Real Time Symbolic Dynamic Programming for Hybrid MDPs

Luis G. R. Vianna and **Leliane N. de Barros**
IME - USP
São Paulo, Brazil

Scott Sanner
NICTA & ANU
Canberra, Australia

Abstract

Symbolic Dynamic Programming (SDP) has been recently proposed as an exact and complete solution for finite-horizon Hybrid Markov Decision Processes (HMDPs), i.e. an MDP with mixed continuous and discrete state-space and actions with continuous parameters. In SDP, piecewise continuous functions are represented as eXtended Algebraic Decision Diagrams (XADDs). The XADD representation of piecewise functions naturally factors the continuous state-space into regions that share the same expression and allows an efficient symbolic update of the value for each region. If the initial state is known, using SDP to update the value of the whole state-space is unnecessary and costly. In this case, the optimal policy can be computed only for regions containing relevant states, as it is done in Real Time Dynamic Programming (RTDP) for MDPs. In this work, we propose combining SDP with RTDP into a new HMDP exact solution, named Real Time Symbolic Dynamic Programming (RTSDP). Unlike conventional RTDP, our proposed algorithm updates are not restricted to a single state but are generalised to the region where the current state belongs, which allows easier reuse of previous updates. Our algorithm is empirically tested on a nonlinear domain in which the action set is finite and a continuous actions domain in which actions have continuous parameters. We empirically show that, given an initial state, RTSDP can solve these HMDP problems faster and using less memory than SDP.

Introduction

Real-world planning applications frequently involve continuous resources (e.g. fuel, energy and money) and can be modelled as Hybrid Markov Decision Processes (HMDPs). Symbolic dynamic programming (SDP) ((**?**; **?**)) provides an exact solution for HMDPs with discrete noise, using the eXtended Algebraic Decision Diagram (XADD) representation. The XADD structure is a generalisation of the algebraic decision diagram (ADD) (**?**) to represent functions of both discrete and continuous variables. Decision nodes may contain boolean variables or inequalities between expressions containing continuous variables. Terminal nodes contain an expression on the continuous variables which could be evaluated to a real number. An example of XADD is shown in Figure 1.

Copyright © 2014, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

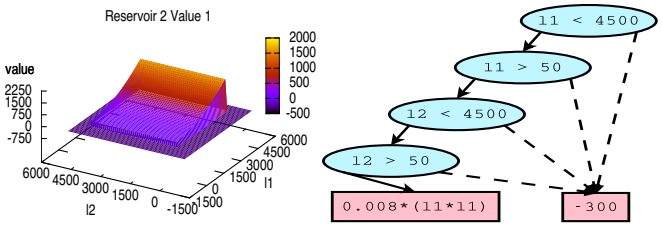


Figure 1: Ilustration of a local reward function from the INVENTORY CONTROLdomain: *(left)* graphical plot; *(right)* XADD representation. Blue ovals represent internal nodes, pink rectangles represent terminal nodes. The $true$ branches are solid lines, $false$ are dashed.

Example 1 - INVENTORY CONTROL (?) *An inventory control problem, consists of determining what itens from the inventory should be ordered and how much to order of each. There is a continuous variables x_i for each item i and a single action $order$ with continuous parameters dx_i for each item. There is a fixed limit L for the maximum number of items in the warehouse and a limit l_i for how much of an item can be ordered in a single action. The items are sold according to a stochastic demand, modelled by boolean variables d_i that represent whether demand is high (Q units) or low (q units) for item i. A reward is given for each sold item, but there are linear costs for items ordered and held in the warehouse. For item i, the reward is as follows:*

$$R_i(x_i, d_i, dx_i) = \begin{cases} \text{if } d_i \wedge x_i + dx_i > Q & : Q - 0.1x_i - 0.3dx_i \\ \text{if } d_i \wedge x_i + dx_i < Q & : 0.7x_i - 0.3dx_i \\ \text{if } \neg d_i \wedge x_i + dx_i > q & : q - 0.1x_i - 0.3dx_i \\ \text{if } \neg d_i \wedge x_i + dx_i < q & : 0.7x_i - 0.3dx_i \end{cases}$$

A drawback of the SDP solution for HMDPs is the overhead of computing a complete optimal policy even for problems where the initial state is known and a partial solution would be sufficient. Though there exist previous solutions to HMDPs that use initial state information, they require monotonic resource consumption and aren't as general as SDP (**?**).

The Real Time Dynamic Programming (RTDP) algorithm is considered a state-of-the-art solver for MDPs ((**?**; **?**)). It combines initial state information and a value function heuristic with asynchronous state value updates to generate an optimal partial policy closed for the relevant states, i.e., states reachable from the initial state following the optimal

policy.

In this work, we propose a new HMDP solver, named Real Time Symbolic Dynamic Programming (RTSDP), which uses the symbolic representation of XADDs combined with RTDP to perform a heuristic search with asynchronous SDP, i.e. dynamic programming updating only some parts of the state space. As the XADD representation naturally factors the state-space into regions with the same continuous expressions, we perform *region based updates* that improve the value function not only for the current state, but *generalize this update to all states in the same region*. We claim that region updates promote better reusability of previous updates, improving the search efficiency.

Our algorithm is empirically tested on two challenging types of domains: nonlinear domains in which the action set is finite but the transition dynamic is nonlinear on state variables; and continuous actions domains in which actions have continuous parameters and the dynamics are linear on both state and parameter variables. We empirically show that, given an initial state, RTSDP can solve these HMDP problems faster and using less memory than SDP.

Hybrid Markov Decision Processes

An MDP (**?**) is defined by a set of states $\mathcal{S}$, actions $\mathcal{A}$, a probabilistic transition function $\mathcal{P}$ and a reward function $\mathcal{R}$. In an HMDP, we consider a model factored in terms of state variables (**?**), i.e. a state $s \in \mathcal{S}$ is an assignment vector of variables $(\vec{b}, \vec{x}) = (b_1, \ldots, b_n, x_1, \ldots, x_m)$, where each $b_i \in \{0, 1\}$ $(1 \leq i \leq n)$ is boolean and each $x_j \in \mathbb{R}$ $(1 \leq j \leq m)$ is continuous. We assume the action set A to be composed of a finite set of parametric actions, i.e. $A = \{a_1(\vec{y}), \ldots, a_K(\vec{y})\}$, where each $\vec{y}$ is the vector of parameter variables. The functions of state and action variables can be compactly represented if structural independencies among variables are exploited in dynamic Bayesian network (DBN) (**?**).

We assume that the factored transition model satisfies the following properties: (i) the next state boolean variables are defined by a probabilistic distribution that only depends on previous state variables and action; (ii) the continuous variables are deterministically dependent on previous state variables, the chosen action and current state boolean variables; (iii) transition functions are piecewise polynomial on the continuous variables. These assumptions allow us to write the transition function in terms of state variables, i.e.:

$$\mathcal{P}(\vec{b'}, \vec{x'} | a(\vec{y}), \vec{b}, \vec{x}) = \prod_{i=1}^{n} \mathcal{P}(b'_i | a(\vec{y}), \vec{b}, \vec{x}) \prod_{j=1}^{m} \mathcal{P}(x'_j | \vec{b'}, a(\vec{y}), \vec{b}, \vec{x}). \tag{1}$$

Assumption (ii) implies the conditional probabilities for continuous variables $\mathcal{P}(x'_j | \vec{b'}, a(\vec{y}), \vec{b}, \vec{x})$ are Dirac Delta functions, which correspond to deterministic transitions, $\vec{x'} \leftarrow \vec{T}_{a(\vec{y})}(\vec{b}, \vec{x}, \vec{b'})$. While this restricts stochasticity only to boolean variables, continuous transitions depend on their sampled values, thus allowing the representation general finite distributions. This is a common restriction in exact HMDP solutions ((**?**; **?**; **?**)).

The reward function $\mathcal{R}(\vec{b}, \vec{x}, a(\vec{y}))$ specifies the immediate reward obtained by taking action $a(\vec{y})$ in state $(\vec{b}, \vec{x})$ and also is a piecewise polynomial function on the continuous variables.

In this paper, we consider HMDPs with an initial state $s_0 \in \mathcal{S}$ and a finite horizon $H \in \mathbb{N}$. A non-stationary policy π is a function which specifies the action $a(\vec{y}) = \pi(s, h)$ to take in state $s \in \mathcal{S}$ at step $h \leq H$. The solution of an HMDP planning problem is an optimal policy π^* that maximizes the expected accumulated reward, $V^{\pi^*}(s_0)$:

$$V^{\pi^*}(s_0) = \mathbb{E}\left[\sum_{t=1}^{H} \mathcal{R}(s_t, a_t(\vec{y_t})) \Big| s_{t+1} \sim \mathcal{P}(s' | s_t, a_t(\vec{y_t}))\right], \tag{2}$$

where $a_t = \pi^*(s_t, t)$ is action chosen by π^* at step t.

Symbolic Dynamic Programming

Before explaining RTSDP, we will first describe the SDP algorithm from (**?**) and then explain how the XADD representation allows it to be efficiently performed.

SDP Algorithm

The symbolic dynamic programming (SDP) algorithm is a generalisation of the classical *value iteration* dynamic programming algorithm (**?**) for constructing optimal policies for MDPs. It proceeds by constructing a series of h stages-to-go optimal value functions $V_h(\vec{b}, \vec{x})$. The pseudocode of SDP is shown in Algorithm 1. Beginning with $V_0(\vec{b}, \vec{x}) = 0$ (line 2), it obtains the *quality* $Q_{a(\vec{y})}(\vec{b}, \vec{x})$ for each action $a(\vec{y}) \in \mathcal{A}$ (lines 5-8) in state $(\vec{b}, \vec{x})$ by regressing the expected reward for $h-1$ future stages $V_{h-1}(\vec{b}, \vec{x})$ along with the immediate reward $R(\vec{b}, \vec{x}, a(\vec{y}))$ as the following:

$$Q_{a(\vec{y})}(\vec{b}, \vec{x}) = R(\vec{b}, \vec{x}, a(\vec{y})) + \sum_{\vec{b'}} \int_{\vec{x'}} \left[\prod_i \mathcal{P}(b'_i | a(\vec{y}), \vec{b}, \vec{x}) \cdot \prod_j \mathcal{P}(x'_j | \vec{b'}, a(\vec{y}), \vec{b}, \vec{x}) \cdot V_{h-1}(\vec{b'}, \vec{x'}) \right]. \tag{3}$$

Given $Q_{a(\vec{y})}(\vec{b}, \vec{x})$ for each $a(\vec{y}) \in A$, we can define the h stages-to-go value function as the quality of the best action in each state $(\vec{b}, \vec{x})$ as follows (line 7):

$$V_h(\vec{b}, \vec{x}) = \max_{a(\vec{y}) \in A} \left\{ Q_{a(\vec{y})}(\vec{b}, \vec{x}) \right\}. \tag{4}$$

The optimal value function $V_h(\vec{b}, \vec{x})$ and optimal policy π^*_h for each stage h are calculated after h steps. Note that the policy is obtained from the maximisation in Equation 4, $\pi^*_h(\vec{b}, \vec{x}) = \arg\max_{a(\vec{y})} Q_{a(\vec{y})}(\vec{b}, \vec{x})$ (line 8). The regression step (Procedure 1.1) computes Equation 3 using the XADD structure.

XADD representation

The XADD data structure (**?**) is used to efficiently represent and manipulate piecewise polynomial functions. An XADD is a directed acyclic graph with two kinds of nodes, internal

Algorithm 1: SDP(HMDP M, H) $\rightarrow (V^h, \pi_h^*)$ **(?)**

$V_0 := 0, h := 0$
while $h < H$ **do**
  $h := h + 1$
  foreach $a \in A$ **do**
    $Q_{a(\vec{y})} \leftarrow$ Regress $(V_{h-1}, a(\vec{y}))$
  $V_h \leftarrow \max_{a(\vec{y})} Q_{a(\vec{y})}$ *// Maximize all* $Q_{a(\vec{y})}$
  $\pi_h^* \leftarrow \arg\max_{a(\vec{y})} Q_{a(\vec{y})}$
return (V_h, π_h^*)

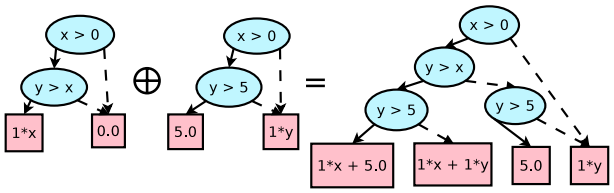
Procedure 1.1: Regress $(V_{h-1}, a(\vec{y}))$
$Q =$ Prime (V_{h-1}) *// Rename all symbolic variables -* *// $b_i \rightarrow b_i'$ and all $x_i \rightarrow x_i'$*
foreach b_i' *in* Q **do**
  // Discrete marginal summation, also denoted $\sum_{b'}$
  $Q \leftarrow \left[Q \otimes P(b_i'|\vec{b}, \vec{x}, a(\vec{y}))\right]|_{b_i'=1}$
    $\oplus \left[Q \otimes P(b_i'|\vec{b}, \vec{x}, a(\vec{y}))\right]|_{b_i'=0}$
foreach x_j' *in* Q **do**
  //Continuous marginal integration
  $Q \leftarrow subst_{x_j'=T_{a(\vec{y})}(\vec{b},\vec{x},\vec{b'})} Q$
return $Q \oplus R(\vec{b}, \vec{x}, a(\vec{y}))$

and terminal. Internal nodes are decision nodes that contain two edges, namely *true* and *false* branches, and a decision that can be either a boolean variable or a polynomial inequality on the continuous variables. Terminal nodes contain a polynomial expression on the continuous variables. XADDs represent piecewise functions by using decision nodes to separate regions and terminal nodes for the local functions (i.e. a expression valid inside a region).

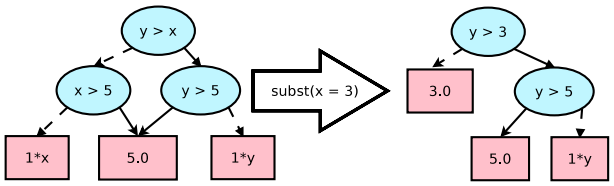
For example, Figure 1(left) shows the plot of a piecewise function and Figure 1(right) its XADD representation. Every path in an XADD defines a partial assignment of boolean variables and a continuous region that satisfies all inequalities in its internal nodes.

The SDP algorithm uses four kinds of XADD operations: (i) Algebraic operations (sum $\oplus$, product $\otimes$) are naturally defined within each region, so the resulting partition is a cross-product of the original partitions and within each region the operation is straightforward. (See Figure 2a), (ii) Substitution operations (replacing or instantiating variables) affect both internal and terminal expressions and any decision that is no longer symbolic is replaced by the corresponding branch.(See Figure 2b), (iii) Comparison operations (maximisation) cannot be performed directly between terminal expressions so a new decision node comparing terminal expressions is created.(See Figure 2c), (iv) Parameter maximization operations modify expressions substituting the maximal values and modify the structure adding comparisons between different branches (See Figure 2d). These XADD operations have been defined previously and we refer to (**?**; **?**) for details.

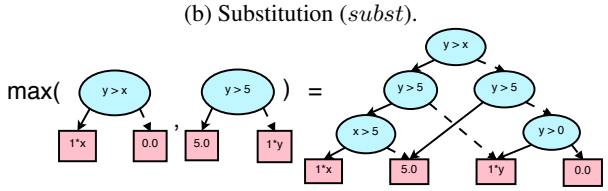
Thus, the SDP Bellman update (Equations 3 & 4) is performed synchronously as a sequence of XADD operations [1], in two steps:



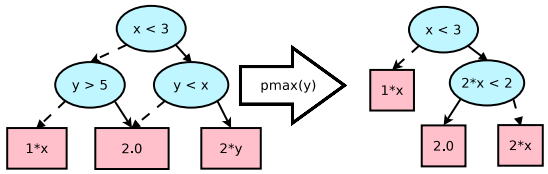
(a) Addition ($\oplus$).



(b) Substitution ($subst$).



(c) Maximisation (max)



(d) Parameter Maximisation (pmax).

Figure 2: XADD Operations.

- Regression of all states:

$$Q_a^{DD(\vec{b},\vec{x},\vec{y})} = R_a^{DD(\vec{b},\vec{x},\vec{y})} \oplus \sum_{\vec{b'}} P_a^{DD(\vec{b},\vec{x},\vec{y},\vec{b'})} \otimes \left[subst_{(x'=T_a^{DD(\vec{b},\vec{x},\vec{y},\vec{b'})})} V_{h-1}^{\prime DD(b',x')} \right]. \quad (5)$$

- Maximization for all states w.r.t. parametrised actions:

$$V_h^{DD(\vec{b},\vec{x})} = \max_a \left(\operatorname*{pmax}_{\vec{y}} Q_a^{DD(\vec{b},\vec{x},\vec{y})} \right). \quad (6)$$

As these operations are performed the complexity of the value function increases and so does the number of regions and nodes in the XADD. A goal of this work is to propose an efficient solution that avoids exploring unreachable or unnecessary regions.

Real Time Symbolic Dynamic Programming

In this section we describe the RTSDP algorithm (Section 4.2), an extension of the RTDP algorithm for MDPs (Section 4.1) and the first asynchronous exact solution for HMDPs.

[1] $F^{DD(vars)}$ is a notation to indicate that F is an XADD symbolic function of $vars$.

The RTDP algorithm

The RTDP algorithm (**?**) (Algorithm 2) uses an initial admissible heuristic value function V and performs trials to improve it iteratively. A trial, described in Procedure 2.1, is a simulated execution of a policy interleaved with local Bellman updates on the visited states. Starting from a known initial state, an update is performed, an action is chosen and a next state is sampled and the trial continues from this new state. Trials are stopped when they reach the horizon H.

The update operation (Procedure 2.2) uses the current value function V to estimate the future reward and evaluates every action a on the current state in terms of its quality Q_a. The action with greatest quality, called greedy action a_g, is chosen and the state's value is set to Q_{a_g}.

RTDP performs updates on one state at a time and only on relevant states, i.e. states that are reachable under the greedy policy. Nevertheless, if the initial heuristic is admissible, there is an optimal policy whose relevant states are visited infinitely often in RTDP trials. Thus the value function converges to V^* on these states and an optimal partial policy closed for the initial state is obtained.

Algorithm 2: `RTDP`(MDP M, s_0, H, V) $\longrightarrow V^*_H(s_0)$

```
while ¬ Solved(M, s_0, V) ∧ ¬TimeOut() do
    V ← MakeTrial(M, s_0, V, H)
return V
```

Procedure 2.1: `MakeTrial` (*MDP M, s, V, h*)

```
if s ∈ GoalStates or h == 0 then
    return V
V_h(s), a_g ← Update(M, s, V, h)
// Sample next state from P(s, a, s')
s^new ← Sample(s, a_g, M)
return MakeTrial(M, s^new, V, h − 1)
```

Procedure 2.2: `Update` (*MDP M, s, V, h*)

```
foreach a ∈ A do
    Q_a ← R(s, a) + Σ_{s'∈S} P(s, a, s') · [V_{h−1}(s')]
a_g ← arg max_a Q_a
V_h(s) ← Q_{a_g}
return V_h(s), a_g
```

Procedure 2.3: `Sample` (*MDP M, s, a_g*)

```
s' ~ P(s, a_g)
return s'
```

The RTSDP Algorithm

We will now generalise the RTDP algorithm for HMDPs. The main `RTDP` (Algorithm 2) and `makeTrial` (Procedure 2.1) functions are trivially extended as they are independent of the state or action representation. The procedure that requires a non-trivial extension is the `Update` function (Procedure 2.2). Considering the continuous state space, it is inefficient to represent values for individual states, therefore we define a new type of update, named *region update*, which is one of the contributions of this work.

While the synchronous SDP update (Equations 5 and 6) modifies the expressions for all paths of $V^{DD(\vec{b},\vec{x})}$, the region update only modifies a single path, the one that corresponds to the region containing the current state $(\vec{b}_c, \vec{x}_c)$, denoted by Ω. This can be achieved by using a "mask", which restricts the update operation to region Ω. The "mask" is an XADD symbolic indicator of whether any $(\vec{b}, \vec{x})$ belongs to Ω, i.e.:

$$I[(\vec{b},\vec{x}) \in \Omega] = \begin{cases} 0, & \text{if } (\vec{b},\vec{x}) \in \Omega \\ +\infty, & otherwise \end{cases}, \tag{7}$$

where 0 indicates valid regions and $+\infty$ invalid ones. The mask is applied in an XADD with a sum ($\oplus$) operation, the valid regions stay unchanged whereas the invalid regions change to $+\infty$. The mask is applied to probability, transition and reward functions restricting these operations to Ω, i.e.:

$$T^{DD(\vec{b},\vec{x},\vec{y},\vec{b'},\vec{x'})}_{a,\Omega} = T^{DD(\vec{b},\vec{x},\vec{y},\vec{b'},\vec{x'})}_{a} \oplus I[(b,x) \in \Omega] \tag{8}$$

$$P^{DD(\vec{b},\vec{x},\vec{y},\vec{b'})}_{a,\Omega} = P^{DD(\vec{b},\vec{x},\vec{y},\vec{b'})}_{a} \oplus I[(b,x) \in \Omega] \tag{9}$$

$$R^{DD(\vec{b},\vec{x},\vec{y})}_{a,\Omega} = R^{DD(\vec{b},\vec{x},\vec{y})}_{a} \oplus I[(b,x) \in \Omega] \tag{10}$$

Finally we can define the region update using the symbolic XADD operations with these restricted functions. In the following steps:

- Regression of region Ω:

$$Q^{DD(\vec{b},\vec{x},\vec{y})}_{a,\Omega} = R^{DD(\vec{b},\vec{x},\vec{y})}_{a,\Omega} \oplus \sum_{\vec{b'}} P^{DD(\vec{b},\vec{x},\vec{y},\vec{b'})}_{a,\Omega} \otimes \left[subst_{(x'=T^{DD(\vec{b},\vec{x},\vec{y},\vec{b'})}_{a,\Omega})} V'^{DD(b',x')}_{h-1} \right]. \tag{11}$$

- Maximization on region Ω w.r.t. parametrised actions:

$$V^{DD(\vec{b},\vec{x})}_{h,\Omega} = \max_a \left(\operatorname*{pmax}_{\vec{y}} \; Q^{DD(\vec{b},\vec{x},\vec{y})}_{a,\Omega} \right). \tag{12}$$

- Update of V restricted to region Ω:

$$V^{DD(b,x)}_{h} = \begin{cases} V^{DD(b,x)}_{h,\Omega}, & \text{if } (\vec{b},\vec{x}) \in \Omega \\ V^{DD(b,x)}_{h}, & otherwise \end{cases} \tag{13}$$

This case update is performed by a XADD minimisation:

$$V^{DD(b,x)}_{h} \leftarrow \min(V^{DD(b,x)}_{h,\Omega}, V^{DD(b,x)}_{h}), \tag{14}$$

where $V_{h,\Omega}$ is $+\infty$ for the invalid regions, and is lower on the valid regions, so that the minimum is indeed the correctly updated V_h.

This new update procedure is described in Algorithm 3. Note that the current state is denoted by $(\vec{b_c}, \vec{x_c})$ to distinguish it from symbolic variables $(\vec{b}, \vec{x})$.

Algorithm 3:

`Region-update`(HMDP,$(\vec{b_c}, \vec{x_c})$,V, h) $\longrightarrow V_h, a_g, \vec{y}_g$

1 //`GetRegion` *extracts XADD path for* $(\vec{b_c}, \vec{x_c})$)
2 $I[(\vec{b}, \vec{x}) \in \Omega] \leftarrow$ `GetRegion`$(V_h, (\vec{b_c}, \vec{x_c}))$
3 $V'^{DD(b',x')} \leftarrow$`Prime` $(V_{h-1}^{DD(b,x)})$ *// Prime variable names*
4 **foreach** $a(\vec{y}) \in A$ **do**
5 $\quad P_{a,\Omega}^{DD(\vec{b},\vec{x},\vec{y},\vec{b'})} \leftarrow P_a^{DD(\vec{b},\vec{x},\vec{y},\vec{b'})} \oplus I[(b,x) \in \Omega]$
6 $\quad T_{a,\Omega}^{DD(\vec{b},\vec{x},\vec{y},\vec{b'},\vec{x'})} \leftarrow T_a^{DD(\vec{b},\vec{x},\vec{y},\vec{b'},\vec{x'})} \oplus I[(b,x) \in \Omega]$
7 $\quad R_{a,\Omega}^{DD(\vec{b},\vec{x},\vec{y})} \leftarrow R_a^{DD(\vec{b},\vec{x},\vec{y})} \oplus I[(b,x) \in \Omega]$
8 $\quad Q_{a,\Omega}^{DD(\vec{b},\vec{x},\vec{y})} \leftarrow R_{a,\Omega}^{DD(\vec{b},\vec{x},\vec{y})} \oplus \sum_{\vec{b'}} P_{a,\Omega}^{DD(\vec{b},\vec{x},\vec{y},\vec{b'})} \otimes \left[subst_{(x'=T_{a,\Omega}^{DD(\vec{b},\vec{x},\vec{y},\vec{b'})})} V'^{DD(b',x')} \right]$
9 $\quad \vec{y}_g^{\,a} \leftarrow \arg\,\text{pmax}_{\vec{y}}\, Q_{a,\Omega}^{DD(\vec{b},\vec{x},\vec{y})}$ *//Maximizing parameter*
10 $V_{h,\Omega}^{DD(\vec{b},\vec{x})} \leftarrow \max_a \left(\text{pmax}_{\vec{y}}\ Q_{a,\Omega}^{DD(\vec{b},\vec{x},\vec{y})} \right)$
11 $a_g \leftarrow \arg\max_a \left(Q_{a,\Omega}^{DD(\vec{b},\vec{x})}(\vec{y}_g^{\,a}) \right)$
12 //The value is updated through a minimisation.
13 $V_h^{DD(b,x)} \leftarrow \min(V_h^{DD(b,x)}, V_{h,\Omega}^{DD(b,x)})$
14 **return** $V_h(b_c, x_c), a_g, \vec{y}_g^{\,a_g}$

Empirical Evaluation

In this section, we evaluate RTSDP for two domains: INVENTORY CONTROL (Example 1) and MARS ROVER (Example 2). These domains show different challenges for planning as INVENTORY CONTROL contains continuously parametrised actions whereas MARS ROVER has a Nonlinear reward function. We show the efficiency of our algorithm by comparing it with synchronous SDP. The space efficiency is measured by the number of nodes in the XADD solution. The time efficiency is measured by the convergency of the value of the initial state s_0.

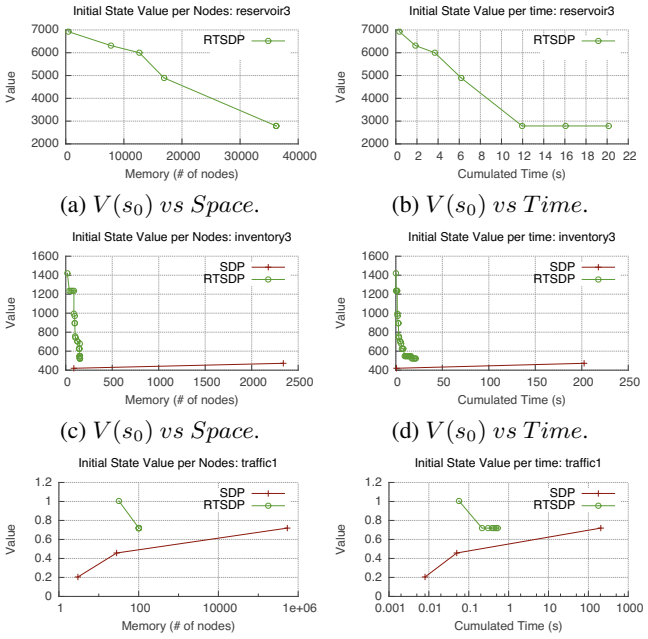
Example 2 (MARS ROVER NonLinear Domain) from (?)

This domain describes a planetary rover moving in a two dimensional plane (x, y), and searching for k marked points from where it must take pictures. There are also k boolean tp_i variables, one for each point, indicating whether the picture of point i was already taken. There is a single move action in this domain that simply reduces the distance from the rover to a specific point by $\frac{1}{3}$ of the current distance. For all experiments, this target point was set to (0, 0). The intent of this action is to represent the fact that a rover may move progressively more slowly as it approaches a target position in order to reach the position with high accuracy. There are k $take - pic_i$ actions that provide reward according to nonlinear expressions over the continuous x and y variables, e.g. it is quadratically proportional to the distance from the picture point. Hence for various points, the rover has to trade-off whether to take each picture at its current position or to get a larger reward by first moving and potentially getting closer before taking the picture. The reward for the $take - pic$ actions for a point in (0,0) is:

$$R_{take-pic_i}(x, y, tp_i) = \begin{cases} \text{if } \neg tp_i \wedge x^2 + y^2 < 4 & : 4 - x^2 + y^2 \\ \text{otherwise} & : 0, \end{cases}$$

which shows that reward $r_i > 0$ is received at the first time the rover chooses take-pic and its distance to (0,0) is less than 2.

In Figure 3 the SDP plot shows the optimal initial state value for different stages-to-go, i.e. $V_h(s_0)$ is the value of the *h-th* point in the plot. The RTSDP plot is the estimated initial state value at the end of each trial, until the optimal value $V_H(s_0)$ is found. Note that RTSDP reaches this optimal value much earlier than SDP and its value function has far fewer nodes. Thus, our solver can achieve the optimal value for the initial state much faster and using a significantly smaller amount of memory space than synchronous SDP.



(a) $V(s_0)$ *vs Space.* (b) $V(s_0)$ *vs Time.*
(c) $V(s_0)$ *vs Space.* (d) $V(s_0)$ *vs Time.*
(e) $V(s_0)$ *vs Space.* (f) $V(s_0)$ *vs Time.*

Figure 3: CRTDP and SDP performance comparison on two domains INVENTORY CONTROL (*top*) and MARS ROVER (*bottom*). Initial state value per memory ($left$) and time ($right$). SDP points are complete iterations (horizon) and RTSDP points are complete trials.

Figures 4 and **??** show the value functions for different stages-to-go generated by RTSDP and SDP when solving the INVENTORY CONTROL instance 2, with $H = 3$ and MARS ROVER instance 1, with $H = 4$. The SDP solutions *(right)* are the optimal value functions ($V_h^{DD(\vec{b},\vec{x})}$) shown as a 3D plot (continuous state variables X state value) and as an XADD. The RTSDP solutions *(left)* are the value functions obtained after a number of trials, such that the initial state value has converged to the optimal. The RTSDP solver runs all trials from the initial state, so the value function for

Instance	SDP		RTSDP	
	Time (s)	Nodes	Time (s)	Nodes
Inventory1	4.6	123	77.3	230
Inventory2	2.07	229	1.01	51
Inventory3	202.5	2340	21.4	152

Table 1: Inventory Results

Instance	SDP		RTSDP	
	Time (s)	Nodes	Time (s)	Nodes
Reservoir1	0.17	318	0.19	138
Reservoir2	0.227	440	0.26	184
Reservoir3	0.53	3476	0.47	657

Table 2: Reservoir $H = 2$ Results

H stages-to-go, V_H *(bottom)* is always updated on the region containing the initial state. As the updates refine the partition, the region containing s_0 shrinks. Note that once a region no longer contains the initial state its heuristic value remains the same (See Figures 4i, 4j, **??** and **??**). In both domains, the use of reachability to prune updates is clear, there are plateaus on the unreachable regions that correspond to the initial heuristic, while the regions containing the initial state have a value similar to the optimal.

Another important comparison is how the solution size (number of XADD nodes) changes with the number of stages-to-go. For the SDP (Figures 4d, 4h, 4l, **??**, **??** and **??**), the complexity of the value function and solution size always increases quickly with horizon. However, for RTSDP (Figures 4b, 4f, 4j, **??**, **??** and **??**) there is different pattern: on one hand, as horizon increases the complexity of the solution in general does increase; On the other hand, if the number of performed actions is small, the number of reachable regions is also small and therefore the number of different terminal nodes and overall XADD size is not so large for the greater horizons.

A peculiarity of this algorithm is that even though its solution is focused to obtain the initial state correct value, it provides useful information on many other states and stages-to-go, what makes it possible to be reused.

Instance	SDP		RTSDP	
	Time (s)	Nodes	Time (s)	Nodes
Reservoir1	DNF	–	1.6	1905
Reservoir2	DNF	–	11.7	7068
Reservoir3	DNF	–	160.3	182000

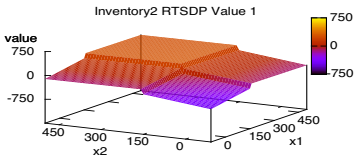
Table 3: Reservoir $H = 3$ Results

Instance	SDP		RTSDP	
	Time (s)	Nodes	Time (s)	Nodes
Traffic1	0.05	28	0.06	28
Traffic2	0.075	63	0.078	48

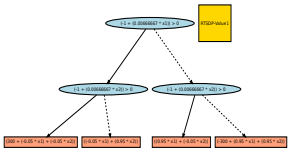
Table 4: Traffic $H = 2$ Results

Instance	SDP		RTSDP	
	Time (s)	Nodes	Time (s)	Nodes
Traffic1	204	533000	0.53	101
Traffic2	DNF	–	1.07	313

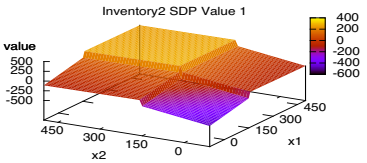
Table 5: Traffic $H = 3$ Results



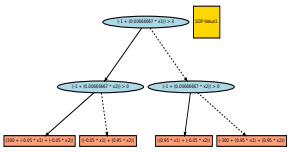
(a) RTSDP - V^1



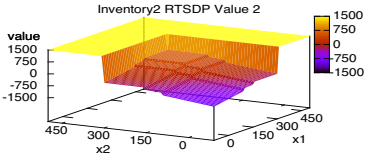
(b) RTSDP - DD^1



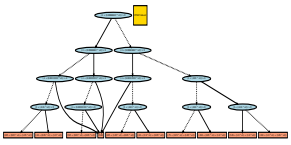
(c) SDP - V^1



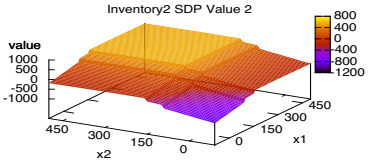
(d) SDP - DD^1



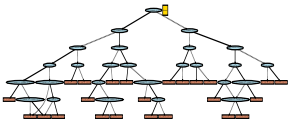
(e) RTSDP - V^2



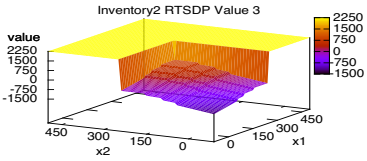
(f) RTSDP - DD^2



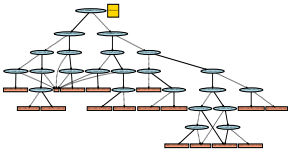
(g) SDP - V^2



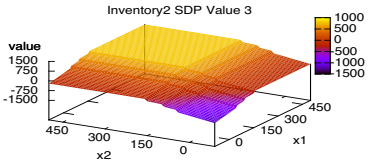
(h) SDP - DD^2



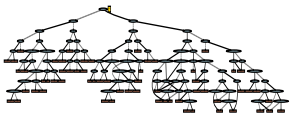
(i) RTSDP - V^3



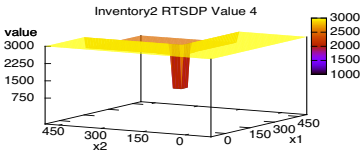
(j) RTSDP - DD^3



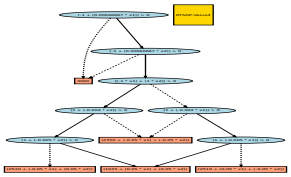
(k) SDP - V^3



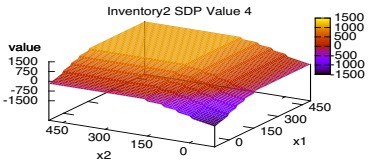
(l) SDP - DD^3



(m) RTSDP - V^4



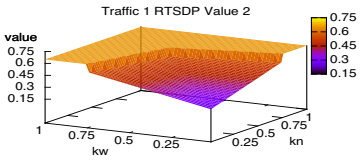
(n) RTSDP - DD^4



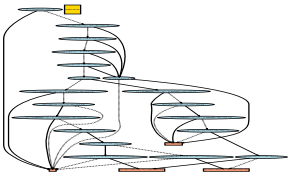
(o) SDP - V^4

(p) SDP - DD^4

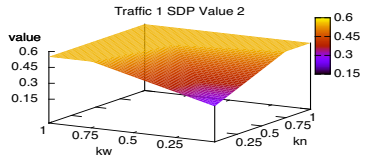
Figure 4: Value functions generated by RTSDP and SDP on INVENTORY CONTROL, with $H = 4$.



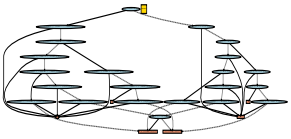
(a) RTSDP - V^2



(b) RTSDP - DD^2

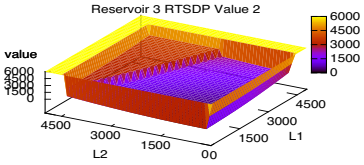


(c) SDP - V^2



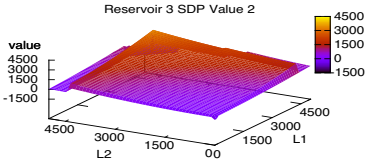
(d) SDP - DD^2

Figure 5: Value 2 functions generated by RTSDP and SDP on TRAFFIC CONTROL, with $H = 2$.

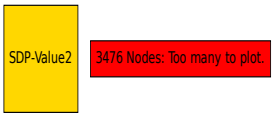


(a) RTSDP - V^2

(b) RTSDP - DD^2



(c) SDP - V^2



(d) SDP - DD^2

Figure 6: Value 2 functions generated by RTSDP and SDP on RESERVOIR, with $H = 2$.