Artificial Intelligence 230 (2016) 192–223

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Real-time dynamic programming for Markov decision processes with imprecise probabilities

Karina V. Delgado *, Leliane N. de Barros, Daniel B. Dias, Scott Sanner

ARTICLE INFO

Article history:
Received 8 February 2014
Received in revised form 19 May 2015
Accepted 17 September 2015
Available online 25 September 2015

Keywords:
Probabilistic planning
Markov decision process
Robust planning

ABSTRACT

Markov Decision Processes have become the standard model for probabilistic planning. However, when applied to many practical problems, the estimates of transition probabilities are inaccurate. This may be due to conflicting elicitations from experts or insufficient state transition information. The Markov Decision Process with Imprecise Transition Probabilities (MDP-IPs) was introduced to obtain a robust policy where there is uncertainty in the transition. Although it has been proposed a symbolic dynamic programming algorithm for MDP-IPs (called SPUDD-IP) that can solve problems up to 22 state variables, in practice, solving MDP-IP problems is time-consuming. In this paper we propose efficient algorithms for a more general class of MDP-IPs, called Stochastic Shortest Path MDP-IPs (SSP MDP-IPs) that use initial state information to solve complex problems by focusing on reachable states. The (L)RTDP-IP algorithm, a (Labeled) Real Time Dynamic Programming algorithm for SSP MDP-IPs, is proposed together with three different methods for sampling the next state. It is shown here that the convergence of (L)RTDP-IP can be obtained by using any of these three methods, although the Bellman backups for this class of problems prescribe a minimax optimization. As far as we are aware, this is the first asynchronous algorithm for SSP MDP-IPs given in terms of a general set of probability constraints that requires non-linear optimization over imprecise probabilities in the Bellman backup. Our results show up to three orders of magnitude speedup for (L)RTDP-IP when compared with the SPUDD-IP algorithm.

1. Introduction

A Markov Decision Process (MDP) encodes the interaction between an agent and its environment: at every stage, the agent decides to execute an action (with probabilistic effects) which leads to a next state and yields a reward. The agent's objective is to maximize the expected reward though a sequence of actions. MDPs [40] have been used as the standard model for probabilistic planning problems where the uncertainty is represented by a state transition probability matrix for each possible action.

There are three important types of MDPs: finite horizon MDPs, where the agent has to act for $H \neq \infty$ steps; infinite horizon MDPs where the agent has to act for $H = \infty$ steps; and indefinite horizon MDPs where the agent has to act for a finite and unknown number of steps (also known as *Stochastic Shortest Path MDPs – SSP MDPs*). Some of the efficient algorithms to MDPs have sought to exploit the factored structure in their representation [11,31,46,9,30], as well as the initial state information of SSP MDPs with a focus on the solution quality of states via asynchronous dynamic programming [3,8].

* Corresponding author.
E-mail addresses: kvd@usp.br (K.V. Delgado), leliane@ime.usp.br (L.N. de Barros), dbdias@ime.usp.br (D.B. Dias), ssanner@nicta.com.au (S. Sanner).

http://dx.doi.org/10.1016/j.artint.2015.09.005
0004-3702/

However, when addressing many real-world problems, it is simply impossible to obtain a precise representation of the transition probabilities in an MDP. There may be many reasons for this, including (a) imprecise or conflicting elicitations from experts [29], (b) insufficient data to form a basis for estimating precise transition models [23], (c) non-stationary transition probabilities due to insufficient state information, or (d) the occurrence of unpredictable events [51]. In all these situations, the agent can be seen making decisions against the adversarial choices of Nature.

Example 1.1. In an automated navigation system, the probabilities of reaching different locations after a movement, may change in the course of time depending on environmental factors (such as weather and road conditions), which can make the navigation more difficult and subject to failure. In general, it is hard to accurately model all these changes since they can include many external dependencies. In view of this, it would be better to have a policy that is optimized for a range of feasible probabilities so that it can be made robust against transition uncertainty.

Example 1.2. In the field of genetics, a genetic regulatory network and a set of actions (therapeutic interventions) can be modeled as an MDP [20,38,13]. This procedure should prevent the network from moving into undesirable states associated with diseases. However, modeling the exact MDP transition probabilities may not be possible when there are few samples or even when exogenous events occur. In this case, a model with imprecise transitions can be used to compute a robust policy for therapeutic interventions.

The *Markov Decision Process with Imprecise transition Probabilities* (MDP-IP) was introduced [44,50] to accommodate optimal models of sequential decision-making in the presence of constraints on the transition probabilities. An MDP-IP is a sequential decision process endowed with a state space, actions and rewards like any MDP, but where transition probabilities may be imprecisely specified via the *parameterized state transition matrices*. For instance, the probability of moving from state s_1 to state s_2, after executing action a_1 can be given by a parameter p_1 that is subject to a constraint such as $0 \leq p_1 \leq 0.75$. That is, instead of a probability measure we have a set of probability measures, for a fixed state-action pair subject to a set of constraints φ over k parameters denoted by $\vec{p} \in [0,1]^k$.

While the MDP-IP establishes a solid framework for the real-world application of decision-theoretic planning, its solution is extremely time-consuming in practice [23] and involves a complex optimization problem rather than an MDP problem: the agent's goal is to maximize its expected reward during a sequence of actions, by taking into account the worst case scenario of imprecise transition probabilities.

The state-of-the-art algorithms for MDP-IPs are based on factored representations and *synchronous dynamic programming* [44,50,23]. The efficiency of the factored MDP-IP algorithm, SPUDD-IP [23], is due to the use of PADDs [23], algebraic decision diagrams with parameterized expressions on its leaves, which can efficiently compute an exact solution that yields speedups of up to two orders of magnitude with the current exact Value Iteration techniques for MDP-IPs. In this study, our concern is to make further improvements in the performance of approaches for solving MDP-IPs. In particular, we seek to explore asynchronous algorithms for a more general class of MDP-IP, called *Stochastic Shortest Path MDP-IP* (SSP MDP-IP).

An asynchronous dynamic programming algorithm for SSP MDPs [4] of particular interest has been the trial-based real-time dynamic programming (RTDP) [3] as is corroborated by a wide range of recent work [8,37,45,43]. Starting from the initial state, this approach updates sampled states during trials (runs), which are the result of simulating a greedy policy. RTDP algorithms have a number of distinct advantages for practical SSP MDP, which are as follows:

(a) *Anytime performance:* RTDP algorithms can be interrupted at any time, and generally yield a better solution the longer they are allowed to run; and
(b) *Optimality without exhaustive exploration:* By focusing on trial-based searches of reachable states from the initial state and using informed heuristics, RTDP algorithms can obtain an optimal policy while visiting (sampling) only a fraction of the state space.

LRTDP [8] is an extension of RTDP that adds a label to states s when all the states s' that are reachable with the greedy policy from s have their utility estimates changed by less than ϵ.

An efficient asynchronous dynamic programming algorithm for SSP MDP-IPs with enumerated states has been previously proposed although it is restricted to interval-based imprecision [14]. However, in general the problem is given in a factored form, i.e., in terms of state variables and in this case even if an interval-based imprecision is assumed for the variables, the previous algorithm is no longer applicable since there are multilinear parameterized joint transition probabilities and a nonlinear optimization over imprecise probabilities in the Bellman backup. This paper outlines the first asynchronous algorithm for SSP MDP-IPs in terms of a general set of constraints on probabilities that can also be applied to factored SSP MDP-IPs. The challenges of extending the RTDP and LRTDP algorithms to solve SSP MDP-IP problems are: (i) How to ensure the convergence of an asynchronous dynamic programming algorithm for SSP MDP-IPs? (ii) How to sample the next state in a trial given the imprecise probabilities? These can be addressed by making the following innovative contributions:

- We propose the first asynchronous algorithm for SSP MDP-IPs given in terms of a general set of probability constraints that requires nonlinear optimization over imprecise probabilities in the Bellman backup, called (L)RTDP-IP.

- We propose three possible methods of choosing the probability measures to sample the next state (Question (ii)) in RTDP-IP: (1) best *worst-case* legal probability measures; (2) random probability measures; and (3) predefined legal probability measures. We seek to show, that despite the imprecision of the transition probabilities, the RTDP-IP algorithm (with any of the three methods) also converges for a robust objective criterion (Question (i)).
- We also propose two symbolic versions of the (L)RTDP-IP algorithm, named factRTDP-IP and factLRTDP-IP that use parameterized decision diagrams to represent and efficiently update the state value function.
- We show in empirical terms our ability to obtain up to three orders of magnitude speedup and compare this with what can be achieved by the state-of-the-art optimal algorithms for SSP MDP-IP domains with sparse transition matrices. This proves that in practice for sparse problems, asynchronous methods are much more efficient than synchronous Value Iteration and allow us to solve larger instances of SSP MDP-IP problems than was previously possible. However, although factored Value Iteration is better than enumerated Value Iteration for MDP-IPs, it turns out that fact(L)RTDP-IP did not outperform the enumerated version, i.e., (L)RTDP-IP.
- We empirically compare the three proposed methods for choosing the probability measures to sample the next state. Our results show that there is no significant difference between the three sampling methods in terms of time.

Sections 2, 3 and 4 review the definitions of SSP MDP, SSP MDP-IP and factored SSP MDP-IP, respectively. Section 5 describes the state-of-the-art MDP-IP exact algorithm, SPUDD-IP [23]. Section 6 provides the first enumerated SSP MDP-IP asynchronous algorithm, called RTDP-IP with proof of its convergence. Section 7 describes two factored versions of RTDP-IP, called factRTDP-IP and factLRTDP-IP. We then analyze our empirical results in Section 8 and compare the performance of the proposed algorithms with SPUDD-IP [23]. In Section 9 we discuss related works. Finally, Section 10 summarizes our conclusions.

2. Indefinite horizon enumerated MDPs

An indefinite horizon enumerated MDP, also called a *Stochastic Shortest Path MDP (SSP MDP)* [6], is a tuple $\langle S, A, C, P, G, s_0\rangle$ where:

- S is a finite set of states;
- A is a finite set of actions;
- $C : S \times A \to \mathbb{R}$ is a cost function;
- $P(s'|s,a)$ defines the conditional transition probability of reaching state s' when starting in state $s \in S$ and executing action $a \in A$;
- $G \subseteq S$ is a set of absorbing goal states. For each $s \in G$, $P(s|s,a) = 1$ and $C(s,a) = 0$ for all $a \in A$; and
- $s_0 \in S$ is an optional initial state.

The solution form for SSP MDP is a function mapping states into actions, called *policy*. Policies of this type are called *stationary*. A *non-stationary policy*, on the other hand, is a function of both state and time. If the mapping is complete for all $s \in S$, this policy is a *complete policy*. However, if only a subset of states is mapped, this policy is called a *partial policy*. If the initial state is unknown, the solution of an SSP MDP must be an optimal stationary complete policy, i.e., a policy whose execution allows the agent to reach a goal state in the least costly way from any state $s \in S$. If the initial state is given, the solution may be an optimal stationary (partial) policy rooted at s_0, i.e., a policy that allows the agent to reach a goal in the least costly way from s_0 and is defined for every state that can be reached from s_0 [35].

Let $\mathbb{P}(\pi)$ be the transition probability matrix $|S| \times |S|$ corresponding to a stationary policy π with elements: $[\mathbb{P}(\pi)]_{ij} = P(s_j|s_i, \pi(s_i)), \forall s_i$ and $s_j \in S$. The probability that the agent is at state s_j at time k, given that the initial state is s_i and using policy π is:

$$[\underbrace{\mathbb{P}(\pi) \times \mathbb{P}(\pi) \times \cdots \times \mathbb{P}(\pi)}_{k}]_{ij}. \tag{1}$$

Definition 2.1 *(SSP MDP proper policy).* A stationary policy π is called *proper* if it leads to a goal with probability 1 for any possible initial state, i.e., $\lim_{k\to\infty}[\mathbb{P}^k(\pi)]_{ij} = 1$ for a $s_j \in G$ and for all $s_i \in S$ [6].

A policy that is not proper is called *improper*. Thus, the following conditions must be satisfied for SSP MDPs:

Assumption 2.2. There exists at least one proper policy.

Assumption 2.3. Every improper policy must incur the cost of ∞ from all the states from which it cannot reach the goal with probability 1.

Notice that if the cost function is strictly positive for all states $s \in S \setminus G$, the Assumption 2.3 is always true.

We define a value function as a mapping $V : S \to \mathbb{R} \cup \{\infty\}$. The value of a policy π at a state s, $V_\pi(s)$, is defined as the expectation of the total cost that the policy π incurs when starting in s, i.e.:

$$V_\pi(s) = E_s^\pi \left[\sum_{t=0}^{\infty} c_t \right], \tag{2}$$

where c_t is the cost incurred at time t when the agent is in state s_t and takes action $\pi(s_t)$.

Finding an optimal solution of an SSP MDP means finding a policy π^* that minimizes V_π. The optimal value function $V^* = V_{\pi^*}$ defined as $V^* = \min_\pi V_\pi$, also satisfies the following condition for all $s \in S$:

$$V^*(s) = \min_{a \in \mathcal{A}} \left\{ C(s,a) + \sum_{s' \in S} P(s'|s,a) V^*(s') \right\}, \tag{3}$$

known as the Bellman equation.

SSP MDPs, which satisfy Assumption 2.2, disallow the existence of dead ends that are states from which the goal cannot be reached [35,28,48]. Different types of SSP MDPs with different assumptions about the existence of dead ends were defined in [35] plus algorithms for solving them. One class of interest in this paper is the SSP where dead ends exist but they are avoidable.

2.1. SSP MDPs with avoidable dead ends

A *Stochastic Shortest Path MDP with avoidable dead ends* [35], is a tuple $\langle S, A, C, P, G, s_0 \rangle$ where S, A, C, P, G and s_0 are defined as in the SSP MDP definition replacing Assumption 2.2 and 2.3 by the following assumptions:

Assumption 2.4. The initial state s_0 is known.

Assumption 2.5. There exists at least one proper policy rooted at s_0.

Assumption 2.6. Every improper policy π has $V_\pi(s_0) = \infty$.

2.2. Value Iteration: SSP MDP synchronous algorithm

Value Iteration (VI) [34] is a classical *synchronous* dynamic programming (DP) algorithm of an SSP MDP. Starting with an arbitrary $V^0(s)$, VI performs value updates for *all* states s, computing the next value function $V^{t+1}(s)$ by applying the B operator:

$$V^{t+1}(s) = (BV^t)(s) = \min_{a \in A} \left\{ Q^{t+1}(s,a) \right\}, \tag{4}$$

where $Q^{t+1}(s,a)$ is the value of state s when applying action a, and is given by:

$$Q^{t+1}(s,a) = C(s,a) + \sum_{s' \in S} P(s'|s,a) V^t(s'). \tag{5}$$

This update is known as a *Bellman update* in V^t. A greedy policy $\pi(s)$ w.r.t. V^t and state s is defined as follows:

$$\pi(s) \in \arg\min_{a \in \mathcal{A}} \left\{ C(s,a) + \sum_{s' \in S} P(s'|s,a) V^t(s') \right\}. \tag{6}$$

$V^t(s)$ converges to the unique optimal value function $V^*(s)$ in the infinite limit of updates, i.e., $\lim_{t \to \infty} \max_s |V^t(s) - V^*(s)| = 0$ [5].

In practice, Value Iteration is stopped when:

$$BE = \max_{s \in S'} |V(s) - (BV)(s)| \leq \epsilon, \tag{7}$$

where S' is the set of states reachable from s_0 under some greedy policy π w.r.t. V and BE is known as the *Bellman residual*.

Value Iteration does not converge on SSP MDPs with avoidable dead ends since, in general, it is not possible to detect divergence of state values [35]. However, when dead ends are easy to be detected, a simple modification of the VI algorithm, converges.

2.3. Admissibility

We say that the value function V^0 is *admissible* iff $V^0(s) \leq V^*(s), \forall s \in S$. The operator B *preserves admissibility*, i.e., if $V(s) \leq V^*(s), \forall s \in S$, then $(B^k V)(s) \leq V^*(s), \forall s \in S$ and $k \in \mathbb{N}^*$ [7]. In this definition B^k is the composition of the operator B, i.e., $(B^k V)(s) = (B(B^{k-1})V)(s), \forall s \in S$.

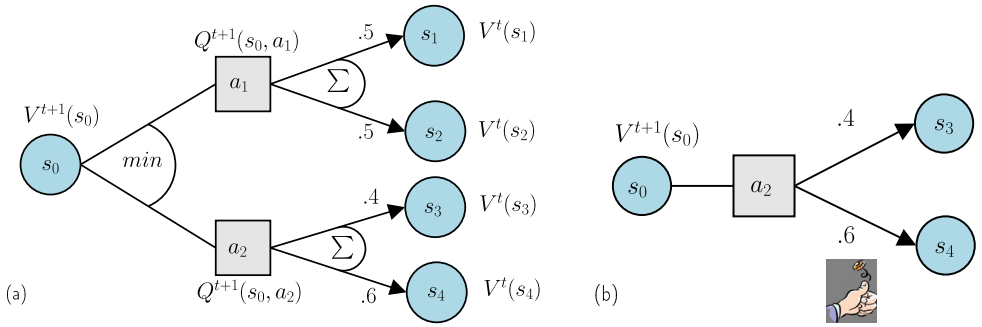


Fig. 1. a) RTDP Bellman update and b) RTDP next state choice.

Algorithm 1: LRTDP(V^0, *SSPMDP*, ϵ, *maxtime*) $\rightarrow V$.

```
1  // Algorithm based on [8]
2  begin
3      // Initialize V with admissible value function V^0
4      V := V^0
5      while (not out of maxtime ∧ ¬s_0.SOLVED()) do
6          visited.CLEAR() // Clear visited states stack
7          s = s_0
8          while (¬s.SOLVED()) do
9              visited.PUSH(s)
10             if (s ∈ G) then
11                 break
12             V(s) := s.UPDATE(V)
13             a := s.GREEDYACTION(V)
14             s := s.CHOOSENEXTSTATE(a) // According to Equation (8)
15
16         // Check state convergence
17         while (¬visited.EMPTY()) do
18             s := visited.POP()
19             if (¬s.CHECKSOLVED(V, ε)) then
20                 break
21
22
23
24     return V
25 end
```

2.4. RTDP and LRTDP: SSP MDP asynchronous algorithms

It should be noted that while Value Iteration updates all states at each iteration, an *asynchronous* dynamic programming method [4] applies the Bellman update to states of an SSP MDP problem in an arbitrary order, while still retaining convergence properties under certain conditions.

The real-time dynamic programming (RTDP) algorithm [3] exploits its knowledge of the initial state by updating states encountered during trial-based simulations of a greedy policy, starting from the initial state. This DP variant explores the state space in trials and performs Bellman updates at each visited state (Fig. 1a). RTDP selects the next state to visit by drawing next state samples s' from the transition distribution for the current greedy action a and current state s (Fig. 1b), i.e.,

$$\text{s.ChooseNextState}(a) = s' \sim P(\cdot|s, a). \tag{8}$$

Since we are now looking for partial policies, i.e., policies that only involve reachable states from s_0, we can also define the idea of *relevant states*.

Definition 2.7 *(Relevant states).* The states reachable from s_0 by any optimal policy are called relevant states [3].

Thus, we need to ensure convergence for optimal values only for the relevant states. The convergence of RTDP is guaranteed if all the states are initialized with an admissible value function V^0.

However, the convergence of RTDP may be slow because states with low transition probability will be less visited and their successor states will not receive many updates. The algorithm Labeled Real-Time Dynamic Programming (LRTDP) [8] has been proposed (Algorithm 1) to improve the time of convergence of RTDP by labeling states s as solved when all

the states s' that are reachable with the greedy policy from s have their utility estimates changed by less than ϵ. And thus it terminates trials that attain labeled states early. LRTDP first initializes V with an admissible value function and applies trial-based simulations that are almost the same as RTDP trials, except that *they stop when a solved state is reached* (Algorithm 1, Line 10). At the end of each trial the CheckSolved algorithm [8] is employed for each state visited during this trial, in reverse order, to determine if the state can be labeled as solved (Algorithm 1, Lines 16–22). A state is marked as *solved* if its greedy graph (the states reachable from s with a greedy policy) is also solved. It should be noted that by stopping a trial when a solved state has been reached, these trials are shortened and the states that have not been solved will be visited more often.

In Algorithm 1, Update (V) computes Equations (4) and (5), while GreedyAction(V) computes Equation (6).

LRTDP can also be used to solve SSP MDPs with avoidable dead ends, defined in Section 2.1, with little adaptation [35].

2.5. RTDP convergence for SSP MDPs

In this section there is a general outline of Barto's RTDP convergence proof for SSP MDPs, which will be used as the basis of the RTDP-IP convergence proof for SSP MDP-IPs given in Section 6. Since RTDP is a special case of asynchronous DP algorithm, its convergence proof is built on the convergence proof of asynchronous DP.

Definition 2.8 *(Asynchronous dynamic programming).* (See [47].) In asynchronous DP, the updates of a value function do not have to be executed in a systematically arranged way. The states are updated in any order using values of other states that are available. Before the values of some states are updated, other states may be updated several times.

Notice that Definition 2.8 is general and does not make a commitment to a particular updating operator or initial value function. Bertsekas [4] has provided evidence to show that a sufficient condition for the convergence of asynchronous DP is that the updating operator must have a contraction property w.r.t. the sup-norm [4].

Theorem 2.9. *Asynchronous DP for SSP MDPs converges to the optimal value if the value of each state is updated infinitely often [4].*

The key to the proof of Theorem 2.9 is that the B operator, as defined by Equation (4), has a contraction property w.r.t. the sup-norm [4]. Note that RTDP is a special case of asynchronous dynamic programming where a single state is updated at each time step. If we consider a set of initial states I, as in the original proof, and if $I = S$, it can be guaranteed that each state is updated infinitely often [3]. However, for $I = \{s_0\}$, as considered in this paper, where only a subset of S is reachable from s_0, we cannot guarantee that all states are updated infinitely often, and therefore Theorem 2.9 cannot be directly employed to prove the convergence of RTDP.

Thus, in order to prove the convergence of RTDP with $I = \{s_0\}$, we have to guarantee that asynchronous DP is performed for the set of relevant states (Definition 2.7). Theorem 2.10 guarantees that for an SSP MDP with an admissible initial value function, RTDP converges over the set of relevant states.

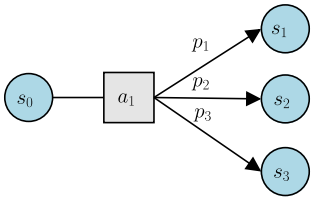
Theorem 2.10 *(Convergence of RTDP). In an SSP MDP, if the initial value function V^0 is admissible, repeated RTDP trials eventually yield optimal values over all relevant states [3].*

The key to the proof of this theorem is the ability of the B operator to preserve admissibility during the RTDP trials since this will eventually cause RTDP to choose the optimal policy.

3. Indefinite horizon enumerated MDP-IPs

An indefinite horizon Markov Decision Process with Imprecise Probabilities, also called a *Stochastic Shortest Path MDP-IP (SSP MDP-IP)* is a sequential decision process with states, actions, a cost function, a set of goal states and an initial state, like an SSP MDP, but with a transition probability function that can be imprecisely defined. That is, an SSP MDP-IP is an extension of an SSP MDP where, instead of a probability distribution $P(\cdot|s,a)$ over the state space S, there is a *set* of probability distributions.

For instance, consider an SSP MDP-IP with four states and two actions, a_1 and a_2. Fig. 2 shows the probability transition function for state s_0 and action a_1 with three possible next states, s_1, s_2 and s_3, and with the transition probabilities given by the parameters p_1, p_2 and p_3, i.e., $p_1 = P(s_1|s_0,a_1)$, $p_2 = P(s_2|s_0,a_1)$ and $p_3 = P(s_3|s_0,a_1)$ (Fig. 2a). We denote $P(\cdot|s_0,a_1)$ for the whole set of probability transition values, i.e., the set of particular values for p_1, p_2 and p_3, with $p_1 + p_2 + p_3 = 1$. Other parameters and/or probability values must be also defined for the transitions applying a_1 to states s_1, s_2 and s_3. Fig. 2b shows the complete transition matrix for action a_1, which includes parameters p_i, with $i \leq 5$, all of them subject to the following sets of constraints $\varphi_1 = \{p_1 \leq 2/3;\ p_3 \leq 2/3;\ 2 * p_1 \geq p_2;\ p_1 + p_2 + p_3 = 1\}$ and $\varphi_2 = \{p_4 + p_5 = 1\}$. By analogy, we should define the transition function for action a_2, which may also include other parameters subject to a set of constraints. The set of constraints is called φ; the vector containing the value of each parameter that satisfies the probability constraints is represented by $\vec{p}$; and the set of all probability measures that satisfy a set of constraints φ is referred to as a *transition credal set*.



(a)

a_1	s'_0	s'_1	s'_2	s'_3
s_0	0	p_1	p_2	p_3
s_1	0	0.5	0.3	0.2
s_2	0	0	0.6	0.4
s_3	p_4	0	p_5	0

Set of constraints φ_1:
$p_1 \leq 2/3$
$p_3 \leq 2/3$
$2 * p_1 \geq p_2$
$p_1 + p_2 + p_3 = 1$

Set of constraints φ_2:
$p_4 + p_5 = 1$

(b)

Fig. 2. An example of an SSP MDP-IP parameterized probabilistic function. a) An illustration of $P(\cdot|s_0, a_1)$. b) The complete parameterized transition matrix for action a_1 and the sets of constraints over the parameters, where φ_1 is related to $P(\cdot|s_0, a_1)$ and φ_2 to $P(\cdot|s_3, a_1)$.

Definition 3.1 *(Transition credal set).* (See [23].) A credal set containing conditional probability distributions over the next state s', given a state s and an action a, is referred to as a *transition credal set* [17] and denoted by $K(\cdot|s, a)$. Thus, $K(\cdot|s, a)$ (or simply $K_{s,a}$) is used to define imprecisely specified transition probabilities.

Notice that $P(\cdot|s, a) \in K(\cdot|s, a)$ is a conditional probability distribution and $P(s'|s, a)$ is a probability measure for s' given s and a. As in [15], we assume state-action pair independence.

Assumption 3.2 *(State-action pair independence).* (See [15].) Probability distributions $P(\cdot|s, a)$ are independent from one state-action pair to another.

In this work, we also assume that all credal sets are closed and convex and that they do not depend on stage t, that is, $K(s'|s, a)$ is stationary. In addition, it is assumed that the imprecise probabilities of an SSP MDP-IP are given in terms of a set of probability parameters p_i subject to a set of general constraints φ, where $\forall P(\cdot|s, a) \in K(\cdot|s, a)$, $P(\cdot|s, a)$ satisfies φ. Note that in this definition each $P(s'|s, a)$ is either linked to a constant between 0 and 1 or to a single parameter p_i.

Definition 3.3 *(SSP MDP-IP).* Formally an SSP MDP-IP is defined by $\langle S, A, C, \mathcal{K}, G, s_0 \rangle$ where S, A, C, G and s_0 are defined as for any SSP MDP and $\mathcal{K}$ is a set of transition credal sets, where each transition credal set $K_{s,a}$ is defined for a state-action pair, i.e., $|\mathcal{K}|=|S \times A|$.

There are several optimization criteria that can be used to define the value of a policy in an SSP MDP-IP. In this paper, we adopt the *minimax* criterion, as it is a reasonable approach when robust policies are required [23]. The minimax criterion can be represented as a two-player zero-sum game where a player's gain is exactly balanced by the loss of the other player. In an SSP MDP-IP the agent selects the policies that minimize its future cost on the assumption that the adversarial agent, i.e., Nature, chooses the probability that maximizes the agent's expected cost. Thus, on the basis of Assumption 3.2, an SSP MDP-IP is a sequential Stochastic Shortest Path Game (SSP game) [39]. For an SSP game, the stationary policy π_M for the minimizer is proper if it leads to the goal with probability 1, for any possible initial state and for any arbitrary policy π_N of the maximizer [39]. The following conditions must be satisfied for SSP games:

Assumption 3.4. There exists at least one proper policy for the minimizer.

Assumption 3.5. If a pair of arbitrary policies π_M for the minimizer and π_N for the maximizer does not lead to the goal with probability 1, then the cost to the minimizer must be ∞.

For an SSP game (with Assumptions 3.4 and 3.5), there is a unique equilibrium value and stationary policies which achieve this equilibrium [39].

Let $\mathbb{P}_{IP}(\pi)$ be the parameterized transition probability matrix $|S| \times |S|$ of an SSP MDP-IP, corresponding to a policy π for the minimizer with elements: $[\mathbb{P}_{IP}(\pi)]_{ij} = P(s_j|s_i, \pi(s_i))$, where each $P(s_j|s_i, \pi(s_i))$ is either linked to a constant between 0 and 1 or to a single parameter p_z subject to a set of constraints φ.

Definition 3.6 *(SSP MDP-IP proper policy).* A stationary policy π for the minimizer is called *proper* if it leads to a goal with probability 1 for any possible initial state and for any probability that the maximizer chooses, i.e., $\lim_{k\to\infty}\{[\mathbb{P}^k_{IP}(\pi)]_{ij}\} = 1$ for a $s_j \in G$, for all $s_i \in S$ and for all values of p_z subject to φ, i.e., $\forall P(\cdot|s_i, \pi(s_i)) \in K(\cdot|s_i, \pi(s_i))$.

π		s'_0	s'_1	s'_2	s'_3	s'_4	s'_5
$\leftarrow$	s_0	0	1	0	0	0	0
$\uparrow$	s_1	0	0	0	1	0	0
$\uparrow$	s_2	0	0	$1-p_1$	0	p_1	0
$\uparrow$	s_3	0	0	0	0	0	1
$\uparrow$	s_4	0	0	0	0	1	0
$\rightarrow$	s_5	0	0	0	0	1	0

Set of constraints φ:
$1/3 \leq p_1 \leq 2/3$

(a)

π		s'_0	s'_1	s'_2	s'_3	s'_4	s'_5
$\leftarrow$	s_0	0	0	0	0	1	0
$\uparrow$	s_1	0	0	0	0	1	0
$\uparrow$	s_2	0	0	$(1-p_1)^k$	0	$p_1\sum_{j=0}^{k-1}(1-p_1)^j$	0
$\uparrow$	s_3	0	0	0	0	1	0
$\uparrow$	s_4	0	0	0	0	1	0
$\rightarrow$	s_5	0	0	0	0	1	0

$lim_{k\to\infty}(1-p_1)^k = 0$

Set of constraints φ:
$1/3 \leq p_1 \leq 2/3$

(b)

Fig. 3. An example of a proper policy π for an SSP MDP-IP. a) The parameterized transition matrix $\mathbb{P}_{IP}(\pi)$. b) $\mathbb{P}^k_{IP}(\pi)$ shows that π is a proper policy for $k \to \infty$.

For example, consider an SSP MDP-IP of a robot navigation problem with six states s_i, with $0 \leq i \leq 5$, and four actions $(\uparrow, \downarrow, \leftarrow, \rightarrow)$. Let $G = \{s_4\}$. Let π be a stationary policy such that $\pi(s_0) = \leftarrow$, $\pi(s_1) = \uparrow$, $\pi(s_2) = \uparrow$, $\pi(s_3) = \uparrow$, $\pi(s_4) = \uparrow$ and $\pi(s_5) = \rightarrow$. Fig. 3a shows the parameterized transition matrix for policy π (for simplicity, we have only include one parameter p_1). Fig. 3b shows that π is proper. $[\mathbb{P}^k_{IP}(\pi)]_{ij} = 0$ for any $0 \leq i \leq 5$ and any $j \neq 4$, for $k \geq 6$. In particular $[\mathbb{P}^k_{IP}(\pi)]_{22}$ is 0 for $k \to \infty$, i.e., $\lim_{k\to\infty}(1-p_1)^k = 0$ considering the constraint $1/3 \leq p_1 \leq 2/3$. Consequently, $\lim_{k\to\infty}\{[\mathbb{P}^k_{IP}(\pi)]_{24}\} = 1$. Thus, π is a proper policy since independently of the maximizer choices for p_1, it leads to the goal with probability 1 from any state s_i (probability 1 for the whole column 4).

The following conditions must be satisfied for SSP MDP-IPs:

Assumption 3.7. There exists at least one *proper policy for the minimizer agent*.

Assumption 3.8. If an arbitrary policy π for the minimizer agent and any probability that the maximizer chooses do not lead to the goal with probability 1, then the cost to the minimizer must be ∞.

As in SSP games, for an SSP MDP-IP with Assumptions 3.7 and 3.8, there is a unique equilibrium value of the SSP MDP-IP and a deterministic stationary policy that achieves this equilibrium. This policy induces an optimal value function that is the unique fixed-point solution of [39]:

$$V^*(s) = \min_{a\in A} \max_{P\in K_{s,a}} \left\{ C(s,a) + \sum_{s'\in S} P(s'|s,a) V^*(s') \right\}, \ \forall s \in S. \tag{9}$$

3.1. SSP MDP-IPs with avoidable dead ends

We can also extend the definition of SSP MDP with avoidable dead ends (Section 2.1) for SSP MDP-IPs. A *Stochastic Shortest Path MDP-IP with avoidable dead ends* is a tuple $\langle S, A, C, \mathcal{K}, G, s_0\rangle$ where $S, A, C, \mathcal{K}, G$ and s_0 are defined as in the SSP MDP-IP definition and the following conditions are satisfied:

Assumption 3.9. The initial state s_0 is known.

Assumption 3.10. There exists at least one proper policy for the minimizer agent rooted at s_0.

Assumption 3.11. If an arbitrary policy π for the minimizer agent and any probability that the maximizer chooses do not lead to the goal with probability 1, then the cost to the minimizer must be ∞.

3.2. Value Iteration for SSP MDP-IPs

A popular algorithm for solving enumerated SSP MDP-IPs based on dynamic programming is Value Iteration (VI) [39] that performs *Bellman updates* for *all* states s, by computing the next value function $V^{t+1}(s)$ applying operator T as follows:

$$V^{t+1}(s) = (TV^t)(s) = \min_{a\in A} \left\{ Q^{t+1}(s,a) \right\}, \tag{10}$$

where

$$Q^{t+1}(s,a) = C(s,a) + \max_{P \in K_{s,a}} \sum_{s' \in S} P(s'|s,a) V^t(s'). \tag{11}$$

Equations (10) and (11) optimize the action choice $a \in A$ w.r.t. the *worst-case* distribution $P \in K_{s,a}$ that maximizes the future expected value. Note that this algorithm (which works with an enumerated state space representation) must solve an optimization problem by calling (an LP solver if φ is a set of linear constraints or otherwise, a nonlinear solver) for every state-action pair at each step; hence it is very computationally expensive in large state spaces.

Theorem 3.12 *(Convergence of Value Iteration for SSP MDP-IPs). (See [39].) For SSP MDP-IP problems, $V^t(s)$ converges to the unique equilibrium value function $V^*(s)$ in the infinite limit of updates, i.e.:*

$$\lim_{t \to \infty} \max_s |V^t(s) - V^*(s)| = 0. \tag{12}$$

In practice, Value Iteration for SSP MDP-IPs is stopped when:

$$BE_{IP} = \max_{s \in S'} |V(s) - (TV)(s)| \leq \epsilon, \tag{13}$$

where S' is the set of states reachable from s_0 under some greedy policy π w.r.t. V and BE_{IP} is known as the Bellman residual for SSP MDP-IP.

3.3. Bounded-parameter enumerated MDP

A particular sub-class of enumerated infinite horizon MDP-IP [44] is the *Bounded-parameter Markov Decision Process (BMDP)* [29], where the imprecise probabilities are specified by intervals, for example $\{0.3 \leq p_j \leq 0.5\}$. One algorithm for BMDPs is Interval Value Iteration [29] that can find an optimal policy (under the worst model) without requiring expensive optimization techniques. The algorithm arranges the next states by the value function in descending order and tries to assign valid higher probabilities for states with a higher value function. There are also algorithms to SSP BMDPs that are extensions of RTDP [14] and LAO* [19,52]. However, they do not address the more general case of probability constraints considered in this work.

4. Factored SSP MDP-IPs

A *factored SSP MDP-IP* is an SSP MDP-IP where states $\vec{x}$ are specified as a joint assignment to a vector $\vec{X}$ of n state variables $(X_1, \ldots, X_n)$ and where the *dynamic credal networks (DCNs)* [17,18,23] are used to represent the transition function. x_i is used to represent the value assignment of the state variable X_i. In this paper binary variables are employed to simplify the notation, although the definitions in this paper can be easily extended to cover multivalued variables. The cost function is also given in terms of a subset of $\{X_1, \ldots, X_n\}$.

We call a partial assignment $\vec{w} \in \{0,1\}^m$, where $m < n$, an *abstract state*. E.g., for $n = 3$, the abstract state $\vec{w} = \{X_1 = 1, X_2 = 0\}$ corresponds to the set of states $\{\{X_1 = 1, X_2 = 0, X_3 = 0\}, \{X_1 = 1, X_2 = 0, X_3 = 1\}\}$.

Definition 4.1 *(Parents of a variable).* Given an action a, the value of a variable $X_i{}'$ depends on a set (possibly empty) of variables $\{X_j, X_k, X_l, \ldots\}$. We call this set of variables the parents of $X_i{}'$ w.r.t. action a denoted by $pa_a(X_i{}')$.

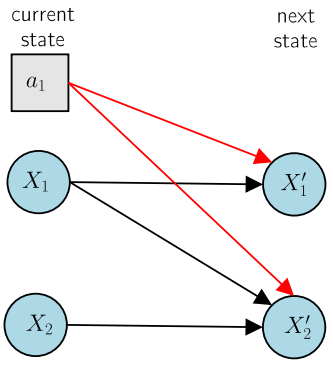
Definition 4.2 *(Factored transition credal set).* (See [23].) Consider an action $a \in A$ and a variable X_i. A credal set containing conditional distributions over the values of X_i given the values of $pa_a(X_i)$ is referred to as a factored transition credal set and denoted by $K_a(x_i|pa_a(X_i), a)$.

The imprecise probabilities of factored SSP MDP-IPs are given in terms of probability parameters subject to a set of constraints φ. Thus the factored transition credal set $K_a(x_i|pa_a(X_i), a)$, for $i = \{1, \cdots, n\}$ and $\forall a \in A$, must satisfy φ. As for enumerated SSP MDP-IPs, we also assume that all the factored transition credal sets are closed and convex and do not depend on stage t.

Definition 4.3 *(Dynamic Credal Network – DCN).* (See [23].) It is a generalization of a Dynamic Bayesian Network (DBN) [21] where, given an action $a \in A$, each variable X_i is associated with factored transition credal sets $K_a(x_i|pa_a(X_i), a)$ for each value of $pa_a(X_i)$. A DCN represents a joint credal set [18,17] over all of its variables consisting of all distributions that satisfy:

$$P(\vec{x}'|\vec{x}, a) = \prod_{i=1}^{n} P(x_i'|pa_a(X_i'), a). \tag{14}$$

For each variable X_i' in a DCN, there is a *conditional probability table (CPT)* with imprecise probabilities, where each line corresponds to a possible combination of values of $pa_a(X_i')$, which defines an abstract state. The entries of a CPT are specified by parameters p_{ij} (i for variable X_i', j for the jth parameter in the CPT for X_i').



(a)

CPT action a_1 for state variable X_1'

X_1	$X_1' = 0$	$X_1' = 1$
0	p_{11}	$1 - p_{11}$
1	p_{12}	$1 - p_{12}$

Set of constraints φ_1:
$0.3 \leq p_{11}$
$p_{12} \leq 0.8$

(b)

CPT action a_1 for state variable X_2'

X_1	X_2	$X_2' = 0$	$X_2' = 1$
0	0	p_{21}	$1 - p_{21}$
0	1	p_{22}	$1 - p_{22}$
1	0	p_{23}	$1 - p_{23}$
1	1	p_{24}	$1 - p_{24}$

Set of constraints φ_2:
$0.5 \leq p_{21} \leq 0.7$
$0.3 \leq p_{22} \leq 0.5$
$0.25 \leq p_{23} \leq 0.35$
$0.1 \leq p_{24} \leq 0.2$

(c)

Fig. 4. Dynamic Credal Network for action $a_1 \in A$. a) Dependence diagram. b) CPT for the state variable X_1' and the set of constraints φ_1. c) CPT for the state variable X_2' and the set of constraints φ_2 related to the probabilities.

Definition 4.4 *(Factored SSP MDP-IP).* Formally a Factored SSP MDP-IP is defined by $\langle \vec{X}, A, C, \mathcal{K}, G, \vec{x}_0 \rangle$ where A is defined as for any SSP MDP-IP, $\vec{X}$ is a vector of n state variables $(X_1, \ldots, X_n)$, C is a factored cost function, $\mathcal{K}$ is a set of factored transition credal sets (Definition 4.2), G is a set of goal states and $\vec{x}_0$ the initial state.

From Equations (9) and (14), the factored optimal value function for SSP MDP-IPs is given by:

$$V^*(\vec{x}) = \min_{a \in A} \left\{ C(\vec{x}, a) + \max_{P \in K_a} \sum_{\vec{x}' \in S} \prod_{i=1}^{n} P(x_i' | pa_a(X_i'), a) V^*(\vec{x}') \right\}, \ \forall \vec{x} \in \{0,1\}^n. \tag{15}$$

To solve Equation (15) we can first compute the product (the joint transition probability) or use variable elimination as SPUDD-IP [23] (as explained in Section 5.3).

4.1. Abstract state and action pair independence and the number of credal sets

For enumerated SSP MDP-IPs we assume state and action pair independence (Assumption 3.2). In the case of factored SSP MDP-IP, instead of assuming independence for single states we now assume independence for abstract states, i.e., a set of states that are defined by the partial assignment of $pa_a(X_i')$.

Assumption 4.5 *(Abstract state and action pair independence).* Probability distributions $P(X_i' | pa_a(X_i'), a)$ are independent from one abstract state and an action pair to another.

An example of a DCN is shown in Fig. 4a. The CPTs for variables X_1' and X_2' are shown in Figs. 4b and 4c. The set of constraints φ on p_{ij} are shown below the CPTs. The CPT of Fig. 4b is defined for two abstract states $\vec{w}_1 = \{X_1 = 0\}$ and $\vec{w}_2 = \{X_1 = 1\}$. Note that the parameters for different lines of the CPTs (abstract states) are different and the constraints are also independent for different abstract states.

a_1	s'_0	s'_1	s'_2	s'_3
s_0	$p_{11}p_{21}$	$p_{11}(1-p_{21})$	$(1-p_{11})p_{21}$	$(1-p_{11})(1-p_{21})$
s_1	$p_{11}p_{22}$	$p_{11}(1-p_{22})$	$(1-p_{11})p_{22}$	$(1-p_{11})(1-p_{22})$
s_2	$p_{12}p_{23}$	$p_{12}(1-p_{23})$	$(1-p_{12})p_{23}$	$(1-p_{12})(1-p_{23})$
s_3	$p_{12}p_{24}$	$p_{12}(1-p_{24})$	$(1-p_{12})p_{24}$	$(1-p_{12})(1-p_{24})$

Set of constraints φ_1:
$0.3 \leq p_{11}$
$p_{12} \leq 0.8$

Set of constraints φ_2:
$0.5 \leq p_{21} \leq 0.7$
$0.3 \leq p_{22} \leq 0.5$
$0.25 \leq p_{23} \leq 0.35$
$0.1 \leq p_{24} \leq 0.2$

Fig. 5. The complete parameterized transition matrix $P(\cdot|s_i, a_1)$ computed from the DCN given in Fig. 4, where $s_0 = \{X_1 = 0, X_2 = 0\}$, $s_1 = \{X_1 = 0, X_2 = 1\}$, $s_2 = \{X_1 = 1, X_2 = 0\}$ and $s_3 = \{X_1 = 1, X_2 = 1\}$.

The number of credal sets $|\mathcal{K}|$ depends on the number of state variables n, the number of actions $|A|$ and the state variable independence, i.e.:

$$|\mathcal{K}| \leq n * 2^{\max_{a,X_i} |pa_a(X_i)|} * |A|, \tag{16}$$

where $\max_{a,X_i} |pa_a(X_i)|$ is the maximum number of parents in the problem for all state variables and actions. For example, for a domain with 30 variables, $\max_{a,X_i} |pa_a(X_i)| = 5$ and $|A| = 10$, $|\mathcal{K}| \leq 30 \times 10 \times 2^5$, while in the enumerated version we have $|\mathcal{K}| = 10 \times 2^{30}$.

4.2. Generating an enumerated SSP MDP-IP from a factored SSP MDP-IP

Different from an enumerated SSP MDP-IP, a factored SSP MDP-IP enables us to specify the independence that exists among state variables. The fact that a variable only depends on its parents allows variable elimination of SPUDD-IP [23] to be more efficient, depending on $|pa_a(X'_i)|$.

Given a factored SSP MDP-IP, we can transform it into its corresponding enumerated SSP MDP-IP by computing the joint transition probability according to Equation (14). Fig. 5 shows the joint credal sets computed for a factored SSP MDP-IP given in Fig. 4. For instance, to compute the probability to be at state $\vec{x}' = \{X'_1 = 0, X'_2 = 0\}$ given that we are at state $\vec{x} = \{X_1 = 1, X_2 = 1\}$ and we take action a_1, we have to multiply the probability parameters p_{12} (Fig. 4b) and p_{24} (Fig. 4c), i.e, $P(X'_1 = 0, X'_2 = 0|X_1 = 1, X_2 = 1, a_1) = p_{12}p_{24}$, which corresponds to $P(s'_0|s_3, a_1)$ in the matrix of Fig. 5. Note that even considering sets of linear constraints φ_1 and φ_2 (Fig. 5), we now have a nonlinear optimization problem due to multiplied parameters.

Thus, while for enumerated SSP MDP-IPs as defined in Section 3, the transition probability can be trivially linear (a single probability parameter p_i for each $P(s'|s, a)$), in an enumerated SSP MDP-IP generated from a factored SSP MDP-IP, there are *multilinear* parameterized joint transition probabilities.

Notice also that if the constraints are simple intervals, as in Figs. 4b and 4c, which could define a factored BMDP (Section 3), when the joint transition probabilities are calculated, we no longer have an interval-based model and thus the Interval Value Iteration algorithm [29] for BMDPs cannot be applied.

Since in practice, problems are commonly described in a factored way, it must be assumed that any SSP MDP-IP generated from a factored description will have to deal with nonlinear optimization problems, even though the set of constraints for the probability parameters are simple intervals.

5. Value Iteration for factored MDP-IPs

SPUDD [31] is an efficient Value Iteration solver for factored infinite horizon MDPs. It uses *Algebraic decision diagrams (ADDs)* [2] to compactly represent the transition and cost function and efficiently compute the optimal policy. An extension of ADDs, called *Parameterized ADDs (PADDs)* was proposed in [23] to efficiently solve factored infinite horizon MDP-IPs in an algorithm called SPUDD-IP that proves to be up to two orders of magnitude faster than Value Iteration when solving the corresponding enumerated infinite horizon MDP-IPs.

5.1. Algebraic decision diagrams

Algebraic decision diagrams (ADDs) [2] are used to compactly represent functions $f : \{0, 1\}^n \to \mathbb{R}$ with context-specific independence [10]. ADDs are an extension of BDDs [12] that represent boolean functions $f : \{0, 1\}^n \to \{0, 1\}$. ADDs are *directed acyclic graphs (DAGs)* whose nodes are associated to decision variables $Y_i \in Y$ $(1 \leq i \leq n)$ and the two out-going arcs

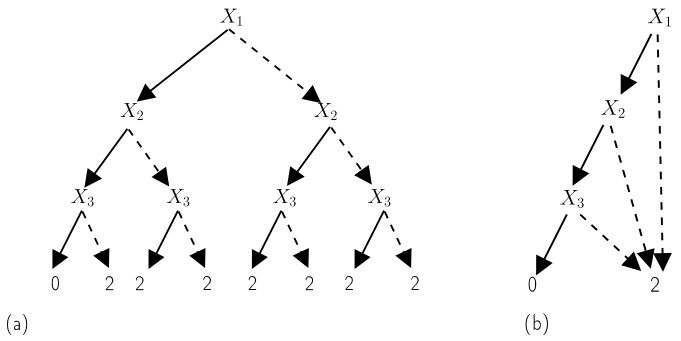


Fig. 6. An example of cost function represented as a) a tree and b) as an ADD. The solid line indicates the true (1) branch of a variable and the dashed line indicates the false (0) branch. The cost is 0 for $\vec{x} = \{X_1 = 1, X_2 = 1, X_3 = 1\}$, otherwise it is 2.

for a node are labeled true (high) and false (low), respectively [2]. Additionally, any path from root to leaf follows a fixed total variable ordering $\mathbb{O}$. In an ADD, identical subgraphs are merged, which can result in a compact representation of f. For example, the cost function $C : \{0,1\}^n \to \mathbb{R}^+$ represented in Fig. 6a as a decision tree can be represented as an ADD in a compact form (Fig. 6b).

Given a boolean function f represented as an ADD, called F, and a set of variables Y with variable assignment $\vec{y} \in \{0,1\}^n$; *EvalADD*$(F, \vec{y})$ returns the real value of the leaf following the corresponding path from the ADD root.

Unary operations such as min, max, restriction, marginalization over variables ($\sum_{x_i \in X_i}$), as well as binary operations such as addition ($\oplus$), subtraction ($\ominus$), multiplication ($\otimes$), min$(\cdot,\cdot)$ and max$(\cdot,\cdot)$ can be performed efficiently on ADDs.

5.2. Parameterized ADDs (PADDs)

PADDs [23] are an extension of ADDs used to compactly represent functions $f : \{0,1\}^n \to \mathbb{E}$, where $\mathbb{E}$ is the space of expressions over a set of parameters Z and where $Z_i \in Z$ $(1 \le i \le k)$ belongs to the interval $[0,1]$ and it is given a set of constraints over these parameters. For example, the CPTs from Figs. 4b and 4c can be represented as PADDs (Fig. 7) that contain in the leaves polynomials involving probability parameters p_{ij} and a set of constraints φ for them.

Since the expressions in $\mathbb{E}$ can be restricted to polynomials, a PADD that represents a boolean function f, called F, can be defined by the following BNF grammar [23]:

$$\begin{aligned} F &::= Poly | \text{if}(F_{var} = 1) \text{ then } F_h \text{ else } F_l \\ Poly &::= const | Z_i * Poly + Poly, \end{aligned} \tag{17}$$

where $Z_i \in Z$ is subject to a given set of constraints. In the if-then-else statement of PADDs we call F_h the true (high) branch, F_l the false (low) branch and F_{var} the test variable of the decision node, which can be any Y_i, $(1 \le i \le n)$. Notice that this definition allows *Poly* to be any polynomial expression.

ADD operations can be easily extended and efficiently performed on PADDs. **Binary operations** are $\oplus$ (sum), $\ominus$ (subtraction), and $\otimes$ (product). **Unary operations** are *restriction*, denoted by $F|_{Y_i = y_i}$, which can be calculated by replacing all decision nodes for variable Y_i with either the true or false branch (see the example in Fig. 8); and *sum out* (also called *marginalization*), which eliminates a variable Y_i from the PADD and is computed by $\sum_{y_i \in \{0,1\}} F = F|_{Y_i=0} + F|_{Y_i=1}$. **N-ary operations** are: $\bigotimes_{i=1}^{n}$ (the product of a sequence) and $\sum_{y_1,\cdots,y_n / y_i \in \{0,1\}}$ (multiple sum out, which computes a sequence of n *sum out* operations, i.e., $\sum_{y_1,\cdots,y_n} F = \sum_{y_n}(\cdots(\sum_{y_2}(\sum_{y_1} F))\cdots)$). All these operations are *closed* for PADDs (i.e., these operations on PADDs yield new PADDs with leaves that can also be expressed as *Poly*, i.e., *Poly* can be any polynomial expression). To avoid notational clutter, instead of $\sum_{y_1,\cdots,y_n / y_i \in \{0,1\}}$ we will use $\sum_{y_1,\cdots,y_n}$ and instead of $\sum_{y_i \in \{0,1\}}$ we will use $\sum_{y_i}$.

Next, five PADD functions are defined (Definitions 5.1, 5.2, 5.3, 5.4 and 5.5) that are going to be used in the algorithms proposed in Section 7 and we refer the reader to [23] for an algorithmic description of the basic PADD operations.

Definition 5.1 *(PADD evaluation – EvalPADD$(F, \vec{y})$).* Given a set of variables Y with a variable assignment $\vec{y} \in \{0,1\}^n$, the polynomial returned by the operation PADD evaluation, called *EvalPADD*$(F, \vec{y})$, can be computed recursively by:

$$EvalPADD(F, \vec{y}) = \begin{cases} Poly & \text{if } F = Poly \\ EvalPADD(F_h, \vec{y}) & \text{if } (F \neq Poly) \wedge (\vec{y}(F_{var}) = 1) \\ EvalPADD(F_l, \vec{y}) & \text{if } (F \neq Poly) \wedge (\vec{y}(F_{var}) = 0), \end{cases} \tag{18}$$

where $\vec{y}(F_{var})$ means the value of F_{var} in the variable assignment $\vec{y}$.

EvalPADD$(F, \vec{y})$ can be also computed as a sequence of PADD restriction operations by $(((F|_{Y_1 = \vec{y}(Y_1)})|_{Y_2 = \vec{y}(Y_2)}) \ldots)|_{Y_n = \vec{y}(Y_n)}$ resulting in a terminal node Poly.

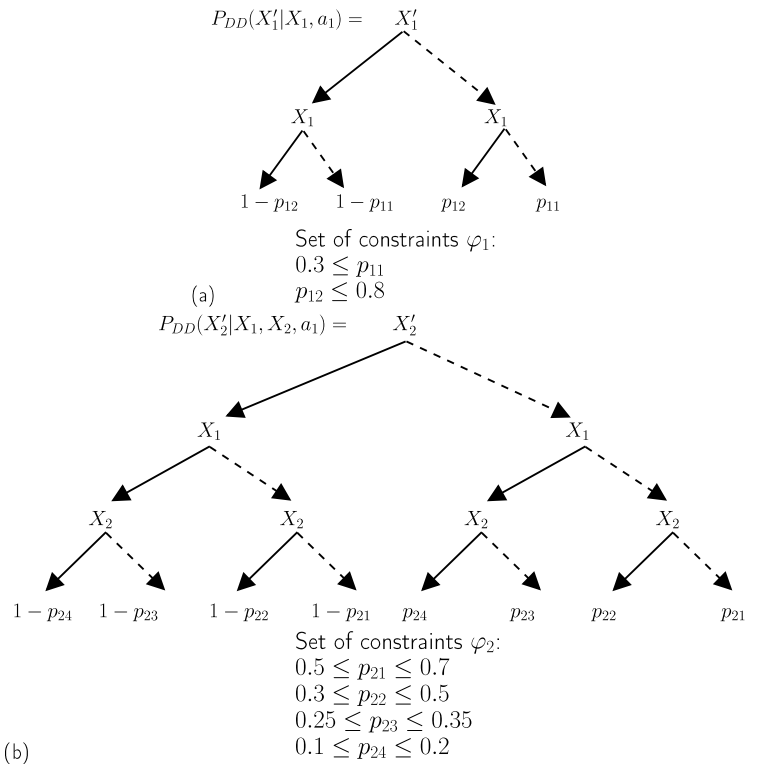


Fig. 7. Parameterized ADD representation for the conditional probability tables from Figs. 4b and 4c denoted by $P_{DD}(X_1'|X_1, a_1)$ and $P_{DD}(X_2'|X_1, X_2, a_1)$, respectively, and the set of constraints $\varphi = \varphi_1 \cup \varphi_2$ over the parameters p_{ij}.

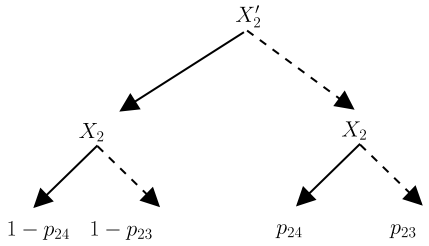


Fig. 8. An example of a restriction operation over the PADD of Fig. 7b for variable $X_1 = 1$.

An example of PADD Evaluation is given in Fig. 9 where F is the probability transition function $P_{DD}(X_2'|X_1, X_2, a_1)$, $n = 4$ and $\vec{y}$ is an assignment for the state variables X_1', X_2', X_1 and X_2, which results in the polynomial $1 - p_{24}$.

Definition 5.2 *(PADD partial evaluation – pEvalPADD$(F, \vec{w})$).* Given a partial assignment, $\vec{w} \in \{0, 1\}^m$, $m < n$, over $W \subset Y$, the partial evaluation of a PADD F, named *pEvalPADD*$(F, \vec{w})$, returns a PADD (instead of a polynomial) and is computed by $(((F|_{W_1=\vec{w}(W_1)})|_{W_2=\vec{w}(W_2)}) \ldots)|_{W_m=\vec{w}(W_m)}$.

Fig. 10 gives an example of PADD partial evaluation, where F is the probability transition function $P_{DD}(X_2'|X_1, X_2, a_1)$ and $\vec{w}$ is an assignment for the state variables X_1 and X_2, which results in a PADD with a single variable X_2'.

Definition 5.3 *(Polynomial evaluation – ValPoly$(Poly, \vec{z})$).* Given a polynomial *Poly* (i.e., a leaf of a PADD), containing a set Z of k parameters and an assignment $\vec{z} \in [0, 1]^k$ over Z, the operation polynomial evaluation, named *ValPoly*$(Poly, \vec{z})$, returns the value of *Poly* according to $\vec{z}$ and can be recursively defined by:

$$ValPoly(Poly, \vec{z}) = \begin{cases} const & \text{if } Poly = const \\ \vec{z}(Z_i) * ValPoly(Poly_1, \vec{z}) + ValPoly(Poly_2, \vec{z}) & \text{if } Poly \neq const, \end{cases} \tag{19}$$

where $\vec{z}(Z_i)$ means the value of Z_i in the parameter assignment $\vec{z}$.

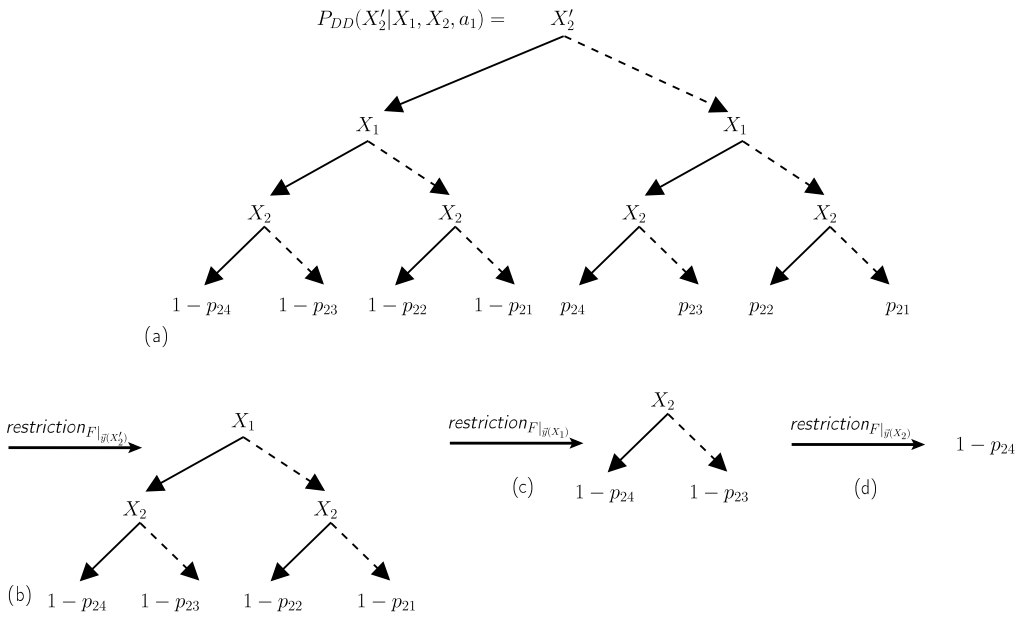


Fig. 9. PADD evaluation of the function $P_{DD}(X_2'|X_1, X_2, a_1)$ of Fig. 7b with $\vec{y} = \{X_1' = 1, X_2' = 1, X_1 = 1, X_2 = 1\}$. a) The PADD before evaluation. b) The resulting PADD after applying *restriction*$(F|_{X_2'=\vec{y}(X_2')})$. c) The resulting PADD after applying *restriction*$(F|_{X_1=\vec{y}(X_1)})$. d) The resulting PADD after applying *restriction*$(F|_{X_2=\vec{y}(X_2)})$.

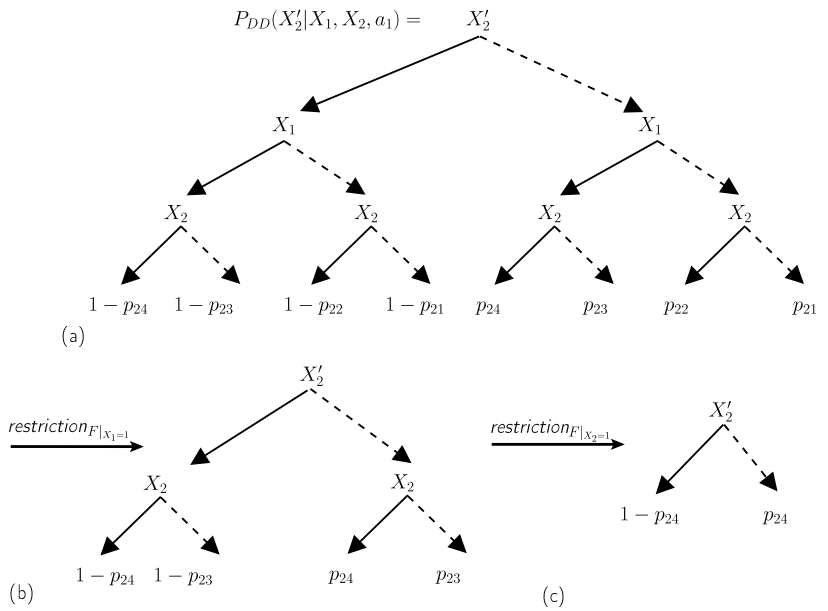


Fig. 10. PADD partial evaluation of the function $P_{DD}(X_2'|X_1, X_2, a_1)$ of Fig. 7b with $\vec{w} = \{X_1 = 1, X_2 = 1\}$. a) The PADD before evaluation. b) The resulting PADD after applying *restriction*$(F|_{X_1=\vec{w}(X_1)})$. c) The resulting PADD after applying *restriction*$(F|_{X_2=\vec{w}(X_2)})$.

For example, given $k = 4$, a polynomial $Poly = p_{12}p_{22}$ and $\vec{z} = \{p_{11} = 0.5, p_{12} = 0.3, p_{21} = 0.65, p_{22} = 0.4\}$, $ValPoly(Poly, \vec{z})$ results in 0.12.

Definition 5.4 *(PADD replacing parameters – ValPADDLeaves$(F, \vec{z})$).* Given a PADD F containing a set of k parameters Z and an assignment $\vec{z} \in [0, 1]^k$ over Z, the operation PADD replacing parameters, named *ValPADD Leaves*$(F, \vec{z})$, returns an ADD, where each polynomial leaf *Poly* is replaced by $ValPoly(Poly, \vec{z})$ (Equation (19)).

ValPADDLeaves$(F, \vec{z})$ can be recursively computed by constructing a unique canonical ADD, from the terminal leaf nodes (replaced by the value returned by $ValPoly(Poly, \vec{z})$) all the way up to the root node.

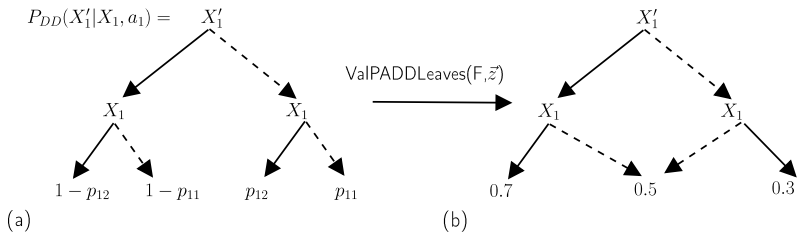


Fig. 11. PADD replacing parameters of the function $P_{DD}(X_1'|X_1, a_1)$ of Fig. 7a with $k = 4$ parameters, $\vec{z} = \{p_{11} = 0.5, p_{12} = 0.3, p_{21} = 0.65, p_{22} = 0.4\}$. a) The PADD before evaluation. b) The resulting ADD after applying *ValPADDLeaves*$(F, \vec{z})$.

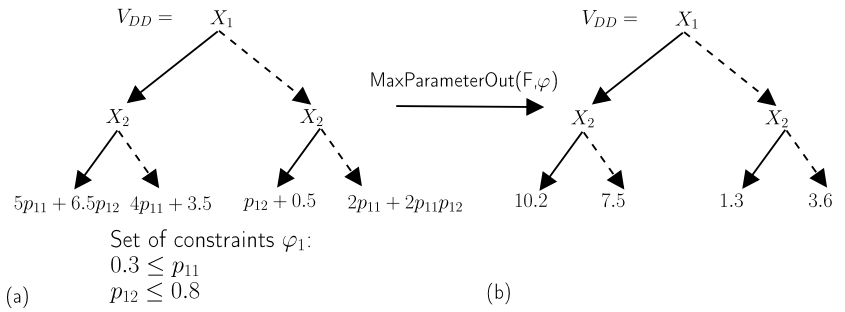


Fig. 12. PADD MaxParameterOut example. a) The PADD V_{DD} before applying the operation and the set of constraints φ_1. b) The resulting ADD after applying *MaxParameterOut*(F, φ).

An example of PADD ReplacingParameters is given in Fig. 11 where $k = 4$, $\vec{z} = \{p_{11} = 0.5, p_{12} = 0.3, p_{21} = 0.65, p_{22} = 0.4\}$ and F is the probability transition function $P_{DD}(X_1'|X_1, a_1)$ which results in an ADD of Fig. 11b.

Definition 5.5 *(PADD MaxParameterOut – MaxParameterOut(F, φ)).* (See [23].) Given a PADD F with a set of parameters Z such that $|Z| = k$ and a set of constraints φ over Z, *MaxParameterOut*(F, φ) returns an ADD, where each polynomial leaf Poly is replaced by the value returned by calling a nonlinear solver with $\max_{Z_1,\cdots,Z_k} Poly\ s.t.\ \varphi$.

MaxParameterOut(F, φ) can be recursively computed by an algorithm similar to *ValPADDLeaves*$(F, \vec{z})$ [23].

An example of PADD MaxParameterOut is given in Fig. 12 where F is the value function V_{DD} and φ_1 is the set of constraints, which results in an ADD of Fig. 12b.

5.3. SPUDD-IP

Satia and Lave, 1970 [44], were the first to propose an exact Value Iteration algorithm for enumerated infinite horizon MDP-IPs. A more efficient algorithm was proposed by Delgado et al., 2011 [23], which could solve factored infinite horizon MDP-IP by using PADDs in all steps of the Value Iteration algorithm, which is called SPUDD-IP. This algorithm is adapted to solve SSP MDP-IP and defines Equations (10) and (11) by means of a factored representation of the transition probability function (Equation (14)) and the cost function. Thus, a factored Bellman update for all states is given by[1]:

$$V_{DD}^{t+1}(\vec{X}) = \min_{a \in A} \left\{ Q_{DD}^{t+1}(\vec{X}, a) \right\}, \tag{20}$$

where

$$Q_{DD}^{t+1}(\vec{X}, a) = C_{DD}(\vec{X}, a) \oplus \max_{\vec{p} \in K_a} \boxed{\sum_{x_1', \cdots, x_n'} \bigotimes_{i=1}^{n} (P_{DD}(X_i'|pa_a(X_i'), a)) \otimes V_{DD}^{t}(\vec{X}').} \tag{21}$$

All the operations in (20) and (21) can be computed through decision diagram operations where the cost and value functions for all the states are ADDs (denoted by C_{DD} and V_{DD}^t); and the transition functions (CPTs) are represented as PADDs, denoted by $P_{DD}(X_i'|pa_a(X_i'), a)$, one for each pair of variable X_i' and action a. Note that the maximization $\max_{\vec{p} \in K_a}$

[1] We use *DD*s for functions represented by ADDs or PADDs.

Algorithm 2: RTDP-IP(V^0, $SSPMDP - IP$, $maxtime$) $\rightarrow V$.

```
1  begin
2      // Initialize V with admissible value function V^0
3      V := V^0
4      // CredalSetVerticesHash maps each credal set to its vertices
5      CredalSetVerticesHash := ∅
6      // Compute vertices and random points of all credal sets if needed
7      if (predefined_parameter_choice) then
8          CredalSetVerticesHash := ComputeAllCredalSetVertices ()
9          p⃗ := ComputeRandomPointForAllCredalSets (
                   CredalSetVerticesHash)
10     while (not out of maxtime) do
11         visited.Clear() // Clear visited states stack
12         s := s_0
13         while (s ∉ G) do
14             visited.Push(s)
15             ⟨V(s), p⃗'⟩ := s.RTDP-IP-Update(V)
16             if (minimax_parameter_choice) then
17                 p⃗ := p⃗'
18             a := s.GreedyAction(V)
19             s := s.RTDP-IP-Sampling(a, p⃗)
20
21         // Optimization: reverse-trial update
22         while (¬visited.Empty()) do
23             s := visited.Pop()
24             ⟨V(s), p⃗'⟩ := s.RTDP-IP-Update(V)
25
26     return V
27 end
```

in Equation (21) is performed by applying the *MaxParameterOut* operation on a PADD, i.e., by calling a nonlinear solver to maximize each PADD leaf, which results in an ADD (this operation corresponds to choosing the worst model for an abstract state and action pair). Therefore Q_{DD}^{t+1} is an ADD. This is important because $\min_{a \in A}$ operation[2] in Equation (20) can only be performed on ADDs. By using PADDs, SPUDD-IP performs operations for sets of states (i.e., abstract states) that have the same value and only need to call a nonlinear solver for PADDs leaves, not necessarily for all the states. This is a key factor in improving efficiency by orders of magnitude. For efficient calculations, SPUDD-IP exploits the variable elimination algorithm [53] by using the *sum out* operation for PADDs (rectangle expression of Equation (21)). The termination criterion of SPUDD-IP is when the Bellman residual BE_{IP} becomes smaller than a bound ϵ (Equation (13)).

Two approximate Value Iteration extensions have been proposed for factored infinite horizon MDP-IPs: (i) APRICODD-IP, a direct extension of the approximate version of SPUDD (APRICODD [46]); and (ii) Objective-IP [23] that seeks to convert the polynomial leaves into constants in order to target the main source of time complexity, i.e., the number of calls to the nonlinear solver during the MDP-IP Value Iteration [23].

6. RTDP-IP: enumerated asynchronous algorithm for SSP MDP-IPs

In this section we propose the first general asynchronous algorithm for SSP MDP-IPs: RTDP-IP, which involves optimization problems and is based on the well-known RTDP [3] algorithm for SSP MDPs and its extension (LRTDP-IP), which is based on the LRTDP [8] algorithm for SSP MDPs.

An asynchronous dynamic programming algorithm for enumerated SSP MDP-IPs, called **RTDP-IP** (Algorithm 2), modifies the RTDP algorithm (Section 2) in two important aspects: (i) the Bellman update and (ii) the next state sampling.

The SSP MDP-IP Bellman backup for the current state that was visited in a trial is carried out through Equations (10) and (11) that are computed by RTDP-IP-Update method (Algorithm 3). This method is called by Algorithm 2 in Line 15 and 24. In contrast with the Value Iteration algorithm, the Bellman update is only performed for the current state in a trial. Notice that the ComputeSuccessors method (called by Algorithm 3 in Line 4) must consider a state s' as a successor of a state s, under a greedy action a, if the parameter $p_i = P(s'|s,a)$ can assume a legal value greater than 0.

The next state sampling is carried out by taking into account the imprecise probabilities, i.e.: given a greedy action, we first have to choose values for each parameter p_i, subject to a set of constraints φ, before finally doing the sampling itself (Algorithm 2, Line 19). The algorithm RTDP-IP-Sampling (Algorithm 4) is explained in the next section.

[2] $\min_{a \in A}$ can be computed by applying a sequence of ADD binary operation $\min(\cdot,\cdot)$.

Algorithm 3: STATE::RTDP-IP-UPDATE(V^t) $\longrightarrow \langle V^{t+1}(s), \vec{p}\rangle$.

```
1  begin
2      minQ := ∞
3      for all a ∈ A do
4          SUCC = s.COMPUTESUCCESSORS(a)
5          p⃗' = arg max_{P∈K_{s,a}} Σ_{s'∈SUCC} P(s'|s,a) V^t(s')
6          Q^{t+1}(s,a) := C(s,a) + max_{P∈K_{s,a}} Σ_{s'∈SUCC} P(s'|s,a) V^t(s')
7          if (Q^{t+1}(s,a) < minQ) then
8              minQ := Q^{t+1}(s,a)
9              p⃗ := p⃗'
10
11     end
12     V^{t+1}(s) := minQ
13     return ⟨V^{t+1}(s), p⃗⟩
```

Algorithm 4: STATE::RTDP-IP-SAMPLING($a, \vec{p}$) $\longrightarrow s'$.

```
1  begin
2      //p⃗ starts with minimax_parameter_choice or predefined_parameter_choice
3      if (rand_parameter_choice) then
4          if (CredalSetVerticesHash.GET(⟨s,a⟩)= NULL) then
5              CredalSetVertices := COMPUTECREDALSETVERTICES (s,a)
6              CredalSetVerticesHash.INSERT(⟨s,a⟩,CredalSetVertices)
7          p⃗ := COMPUTERANDOMPOINTCREDALSET (s, a, CredalSetVerticesHash)
8      s' := s.CHOOSENEXTSTATE (a, p⃗) // According to Equation (23)
9      return s'
10 end
```

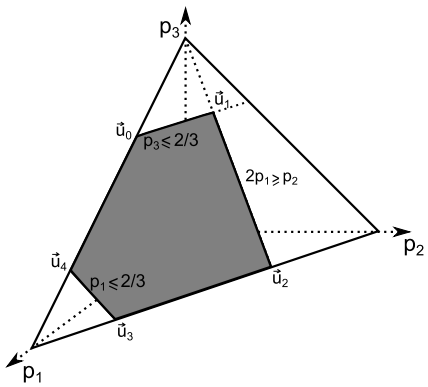


Fig. 13. A transition credal set example represented by the gray area.

6.1. RTDP-IP sampling

In order to sample the next state, three different methods are considered for choosing a value for each probability parameter p_i, subject to a set of constraints φ: (1) the use of the same parameter values computed in the Bellman update called *minimax_parameter_choice*; (2) computing a random legal parameter value at each step during the trial called *rand_parameter_choice*; (3) computing a predefined legal parameter value only once called *predefined_parameter_choice*.

The *minimax_parameter_choice* method uses the same $\vec{p} \in K_{s,a}$, computed in Line 15 (Algorithm 2) by the RTDP-IP-UPDATE method. Thus, $\vec{p}$ can be used for both purposes, i.e., to update a state value and to sample the next state.

The *predefined_parameter_choice* and *rand_parameter_choice* methods compute valid probability distributions for the set of credal sets. Since we assume convex and closed credal sets, we can obtain the vertices $\vec{u_0}, \cdots, \vec{u_l}$ of each credal set $K(\cdot|s,a)$ (each vertex is a distribution over S) by using the LRS software [1].

For instance, consider a problem where for state s_0 and action a_2 there are only three possible next states s_3, s_4 and s_5 with the transition probabilities given by the parameters p_1, p_2 and p_3, i.e., $P(s_3|s_0,a_2) = p_1$, $P(s_4|s_0,a_2) = p_2$ and $P(s_5|s_0,a_2) = p_3$, defined with the following constraints: $\varphi_1 = \{p_1 \leq 2/3, p_3 \leq 2/3, 2, p_1 \geq p_2, p_1 + p_2 + p_3 = 1\}$. The region of all probability measures that satisfy φ_1 is shown in Fig. 13 where the vertices of the polytope (gray region) are: $\vec{u_0} = \{1/3, 0, 2/3\}$, $\vec{u_1} = \{0, 1/3, 2/3\}$, $\vec{u_2} = \{1/3, 2/3, 0\}$, $\vec{u_3} = \{2/3, 1/3, 0\}$ and $\vec{u_4} = \{2/3, 0, 1/3\}$. A valid probability distribution (i.e., a point inside the gray region) could be $\vec{p} = \{0.5, 0.2, 0.3\}$, which is a linear combination of $\vec{u_j}$.

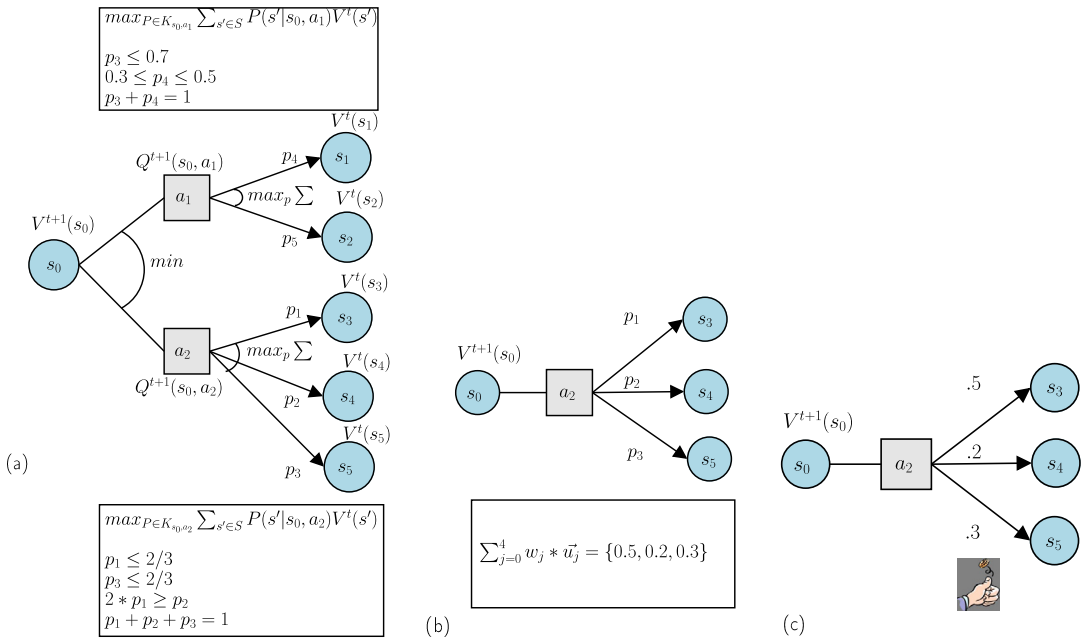


Fig. 14. RTDP-IP Bellman update and next state sampling using the *rand_parameter_choice* method. a) RTDP-IP calls a linear solver twice, for actions a_1 and a_2 and then chooses the greedy action a_2; b) *Rand_parameter_choice* that computes Equation (22) to choose a random point on the polytope, as shown in Fig. 13, i.e., $p_1 = 0.5$, $p_2 = 0.2$ and $p_3 = 0.3$; c) RTDP-IP next state sampling.

Hence, the vertices of the polytope defined by each credal set $K(\cdot|s,a)$ can be used to generate a random point $\vec{p} \in K(\cdot|s,a)$ as a linear combination of $\vec{u_j}$ [24], that is:

$$\sum_{j=0}^{l} w_j * \vec{u}_j, \tag{22}$$

where w_j are generated by a uniform random sample on $[0,1]$ and then normalized to ensure $\sum_{j=0}^{l} w_j = 1$.

The *predefined_parameter_choice* first computes all the credal set vertices (Algorithm 2, Line 8) and then uses the ComputeRandomPointForAll CredalSets function to compute random points $\vec{p} \in K_{s,a}$ (Equation (22)) for all the credal sets (Algorithm 2, Line 9), i.e., for the credal sets $K(\cdot|s,a)$, $\forall s \in S$ and $\forall a \in A$. This method computes them before starting the trials and uses the same $\vec{p}$ for each state-action pair.

The *rand_parameter_choice* method computes a random point (Equation (22)) for the credal set $K(\cdot|s,a)$, where s is the current state in a trial and a is the greedy action (Algorithm 4, Lines 3–7). Thus, while the *predefined_parameter_choice* computes $\vec{p}$ for all state-action pairs before starting the trials, the *rand_parameter_choice* only computes $\vec{p}$ for the visited state and corresponding greedy action. Note that the vertices $\vec{u_0}, \cdots, \vec{u_l}$ for the credal set $K(\cdot|s,a)$ are stored in a hash table called *CredalSetVerticesHash* (Algorithm 4, Line 6). These vertices can be reused if s is revisited and a is still a greedy action, to compute a new $\vec{p}$.

After having chosen the values for $\vec{p}$, with one of the three sampling methods, we can finally carry out the sampling of the next state as follows:

$$\text{s.ChooseNextState}(a, \vec{p}) = s' \sim P(\cdot|s,a), \tag{23}$$

where $P(\cdot|s,a)$ is completely known, i.e., all the probability parameters, $\vec{p}$, are known.

Fig. 14 provides an example of a Bellman update for an SSP MDP-IP and the next state sampling. For the Bellman update, Fig. 14a shows the two calls to a linear solver to choose the probability measure that maximizes the future cost of the agent following action a_1 or a_2, with a_2 being the action that minimizes the results. Fig. 14b illustrates the computation of $\vec{p}$ using the *rand_parameter_choice* method with the greedy action a_2 that results in $p_1 = 0.5$, $p_2 = 0.2$ and $p_3 = 0.3$. Finally, RTDP-IP samples the next state with the chosen parameter values p_1, p_2 and p_3 (Fig. 14c).

We can modify Algorithm 2 to a Labeled RTDP-IP, called LRTDP-IP (Algorithm 5). Unlike CheckSolved [8] for SSP MDP, CheckSolved-IP (called by Algorithm 5, line 27) computes the greedy graph by taking into account all the states that can be reached with the greedy policy with a probability value greater than 0.

Algorithm 5: LRTDP-IP(V^0, $SSPMDP - IP$, $maxtime$, ϵ) $\rightarrow V$.

```
1  begin
2      // Initialize V with admissible value function V^0
3      V := V^0
4      // CredalSetVerticesHash maps each credal set to its vertices
5      CredalSetVerticesHash := ∅
6      // Compute vertices and random points of all credal sets if needed
7      if (predefined_parameter_choice) then
8          CredalSetVerticesHash := ComputeAllCredalSetVertices ()
9          p⃗ := ComputeRandomPointForAllCredalSets(
               CredalSetVerticesHash)
10     while (not out of maxtime ∧ ¬s_0.Solved()) do
11         visited.Clear() // Clear visited states stack
12         s := s_0
13         while (¬s.Solved()) do
14             visited.Push(s)
15             if (s ∈ G) then
16                 break
17
18             ⟨V(s), p⃗'⟩ := s.RTDP-IP-Update(V)
19             if (minimax_parameter_choice) then
20                 p⃗ := p⃗'
21             a := s.GreedyAction(V)
22             s := s.RTDP-IP-Sampling(a, p⃗)
23
24         // Check state convergence
25         while (¬visited.Empty()) do
26             s := visited.Pop()
27             if (¬s.CheckSolved-IP(V, ε)) then
28                 break
29
30
31     return V
32 end
```

6.2. RTDP-IP convergence

Buffet's proof outline [14] is followed for Robust RTDP since it allows us to refer to the imprecision as a set of possible models in the polytope defined by the credal set. However, two points of this proof need to be clarified:

1. The convergence of asynchronous DP for SSP MDP-IPs, which in order to be complete must rely on the fact that the operator T of Equation (10) is monotonic (Theorem 6.1).
2. The preservation of the admissibility of the value function (Proposition 6.2).[3]

Theorem 6.1. *Asynchronous DP for SSP MDP-IP converges to V^* if the value of each state is updated infinitely often.*

Proof. The proof follows directly from the fact that the operator T of SSP MDP-IPs (Equation (10)) can be seen as a particular case of the updating operator for sequential SSP games [39]. Thus, T is monotonic (Lemma A.1 in [39]) and based on the Assumptions 3.7 and 3.8 (Section 3), T has a unique fixed point (Proposition 4.5 in [39]), which is the equilibrium cost of the SSP MDP-IP (Proposition 4.6 in [39]). Also, there exist stationary policies for the minimizer and maximizer that achieve the equilibrium (Proposition 4.6 in [39]). These are sufficient conditions for the convergence of asynchronous DP for SSP MDP-IPs [4]. □

Second, we show that the admissibility of the value function of an SSP MDP-IP is preserved during the RTDP-IP trials.

Proposition 6.2. *If the initial value V^0 is admissible, the admissibility of the value function is preserved during the execution of RTDP-IP, i.e., if $V^0(s) \leq V^*(s), \forall s \in S$ then $V^t(s) \leq V^*(s), \forall s \in S$ and $t \in \mathbb{N}^*$.*

Proof. We will prove by induction that $V^t(s) \leq V^*(s), \forall s \in S$ at any time t.
Base case: For $t = 0$, $V^0(s) \leq V^*(s)$ is true.

[3] This proof was omitted in Buffet's outline.

Induction step: Let s_t be the current state, a_t the greedy action and V^t the value function at time step t, generated by executing RTDP-IP starting from an arbitrary initial state. Since RTDP-IP updates only the state s_t at time t, $V^{t+1}(s) = V^t(s), \forall s \neq s_t$ and by the induction hypothesis if $V^t(s) \leq V^*(s), \forall s \in S$, then $V^{t+1}(s) \leq V^*(s), \forall s \neq s_t$. Now we have to prove for s_t, again by the induction hypothesis if $V^t(s) \leq V^*(s), \forall s' \in S$, then for s_t:

$$V^{t+1}(s_t) = \min_{a \in A}\{C(s_t, a) + \max_{P \in K_{s_t,a}}\{\sum_{s' \in S} P(s'|s_t, a)V^t(s')\}\}$$
$$\leq \min_{a \in A}\{C(s_t, a) + \max_{P \in K_{s_t,a}}\{\sum_{s' \in S} P(s'|s_t, a)V^*(s')\}\} = V^*(s_t). \quad (24)$$

Thus, $V^{t+1}(s) \leq V^*(s), \forall s \in S$ at any time t. □

Note that the only difference from Barto's proof is the addition of $\max_{P \in K_{s_t,a}}$ in Equation (24), which also involves the same inequality. Thus, Barto's proof for the preservation of admissibility by RTDP was easily extended to RTDP-IP.

To complete the convergence proof, we still have to guarantee that the relevant states are visited infinitely often. Thus, it must be ensured that the method of choosing a probability distribution $P \in K_{s,a}$ to sample the next state never eliminates any state to be eventually visited. In our proposal, since *rand_parameter_choice* chooses a different $P \in K_{s,a}$ for each visited state during a trial, it will eventually visit all the reachable states. However, when using the other two methods (*minimax_parameter_choice* and *predefined_parameter_choice*) we must avoid choosing a probability measure equal to 0. For example, the *predefined_parameter_choice* method, could lead us to choose $P(\cdot|s, a) \in K(\cdot|s, a)$ such that $P(s'|s, a) = 0$ for some $s' \in S$ and use it thereafter, which means it would never visit s'. In the worst case scenario, the same thing can happen if *minimax_parameter_choice* chooses $P(\cdot|s, a) \in K(\cdot|s, a)$ such that $P(s'|s, a) = 0$ for some $s' \in S$, every time we visit state s with action a. Thus, convergence can also be guaranteed by using the *minimax_parameter_choice* or *predefined_parameter_choice* methods if we choose $P(\cdot|s, a) \in K(\cdot|s, a)$ such that $P(s'|s, a) \neq 0$ for any $s' \in S$.

In general the convergence theorem of RTDP-IP can be stated as follows:

Theorem 6.3 *(Convergence of RTDP-IP with different methods for sampling the next state). In an SSP MDP-IP, repeated RTDP-IP trials eventually yield optimal values over all relevant states if: (i) the initial value function is admissible, i.e., $V^0(s) \leq V^*(s), \forall s \in S$ and (ii) the method of choosing a probability distribution $P(\cdot|s, a) \in K(\cdot|s, a)$ to sample the next state guarantees choosing $P(\cdot|s, a) \in K(\cdot|s, a)$ such that $P(s'|s, a) \neq 0$ for any $s' \in S$.*

7. factRTDP-IP and factLRTDP-IP: factored asynchronous algorithms for SSP MDP-IPs

In this section the efficiency of SPUDD-IP [31], a factored version of Value Iteration for infinite horizon MDP-IPs, has led us to propose factored versions of RTDP-IP and LRTDP-IP called factRTDP-IP and factLRTDP-IP, respectively.

7.1. ***factRTDP-IP*** *algorithm*

Based on a factored asynchronous algorithm for MDPs [33], we propose the first factored asynchronous algorithm for SSP MDP-IPs, called factRTDP-IP, that combines ideas of SPUDD-IP [23] (Section 5.3) and RTDP-IP (Section 6).

factRTDP-IP update As in an RTDP-like algorithm, the Bellman update is only performed for the currently visited state during a trial, with the following Equation:

$$V^{t+1}(\vec{x}) = \min_{a \in A}\left\{Q^{t+1}(\vec{x}, a)\right\}, \quad (25)$$

where

$$Q^{t+1}(\vec{x}, a) = EvalPADD(C_{DD}(\vec{X}, a), \vec{x}) \oplus$$
$$\max_{\vec{p} \in K_a}\boxed{\sum_{x'_1, \cdots x'_n}\bigotimes_{i=1}^{n}(pEvalPADD(P_{DD}(X'_i|pa_a(X'_i), a), \vec{x})) \otimes V^t_{DD}(\vec{X}')}. \quad (26)$$

It should be noted that, after computing Equation (26), $Q^{t+1}(\vec{x}, a)$ and $V^{t+1}(\vec{x})$ end up having real values, different from the resulting ADD computed by SPUDD-IP backup (Equations (20) and (21)), where $Q^{t+1}_{DD}(\vec{X}, a)$ and $V^{t+1}_{DD}(\vec{X})$ are ADDs that represent the corresponding values for the complete set of states. This is because we are exploring the state space in trials and performing Bellman updates *only for the currently visited state* and as a result only a constant value V^{t+1} is obtained for the state $\vec{x}$.

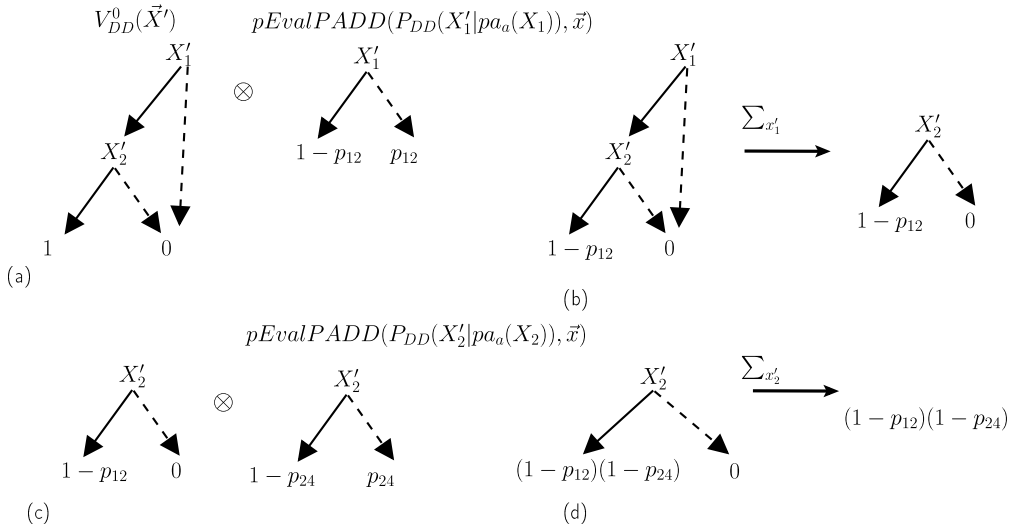


Fig. 15. Example of the PADD operations to compute Equation (28) for the current visiting state $\vec{x} = \{X_1 = 1, X_2 = 1\}$ and action a.

As in the case of SPUDD-IP, the computation of Equation (26) can be performed in an efficient way by eliminating a variable whenever it does not depend on any other variable. For example, if X'_1 is not dependent on any other $X'_i, \forall i \neq 1$ using action a, we can "push" the sum over X'_1 inwards by expanding Equation (26) into:

$$
\begin{aligned}
Q^{t+1}(\vec{x}, a) = {} & EvalPADD(C_{DD}(\vec{X}, a), \vec{x}) \oplus \\
& \max_{\vec{p} \in K_a} \sum_{x'_2, \cdots, x'_n} \bigotimes_{i=2}^{n} pEvalPADD(P_{DD}(X'_i | pa_a(X'_i), a), \vec{x}) \otimes \\
& \sum_{x'_1} pEvalPADD(P_{DD}(X'_1 | pa_a(X'_1), a), \vec{x}) \otimes V^t_{DD}(\vec{X}'),
\end{aligned} \tag{27}
$$

which can be repeated for all variables, obtaining:

$$
\begin{aligned}
Q^{t+1}(\vec{x}, a) = {} & EvalPADD(C_{DD}(\vec{X}, a), \vec{x}) \oplus \\
& \max_{\vec{p} \in K_a} \sum_{x'_n} pEvalPADD(P_{DD}(X'_n | pa_a(X'_n), a), \vec{x}) \otimes \\
& \cdots \\
& \sum_{x'_2} pEvalPADD(P_{DD}(X'_2 | pa_a(X'_2), a), \vec{x}) \otimes \\
& \sum_{x'_1} pEvalPADD(P_{DD}(X'_1 | pa_a(X'_1), a), \vec{x}) \otimes V^t_{DD}(\vec{X}').
\end{aligned} \tag{28}
$$

Since X'_i represents a variable X_i in the next state and $pa_a(X'_i)$ the parents of X'_i, $pEvalPADD(P_{DD}(X'_i|pa_a(X'_i), a), \vec{x})$, which applies the restriction operation for each state variable, results in a PADD with a single variable X'_i.

We show an example of variable elimination in Fig. 15 for the currently visited state $\vec{x} = \{X_1 = 1, X_2 = 1\}$ and action a. Fig. 15a shows the product between the first value function $V^0_{DD}(\vec{X}')$ (obtained after *"priming"* the variables of $V^0_{DD}(\vec{X})$) and the transition probabilities for X'_1 given the current state $\vec{x}$ (i.e., the resulting PADD after applying $pEvalPADD(P_{DD}(X'_1|pa_a\ (X'_1), a), \vec{x})$). Fig. 15b takes the result of the previous product and eliminates variable X'_1 applying summing out over x'_1. After that X'_2 can be eliminated by first multiplying the previous result by the transition probabilities for X'_2 given the current state $\vec{x}$ and action a (Fig. 15c). Finally, in Fig. 15d we apply summing out over x'_2. Notice that the resulting PADD corresponds to a polynomial, which is multilinear as a result of Assumption 4.5. This is important since there are efficient implementations for multilinear optimizations that can be used in practice.

Next, the *MaxParameterOut* PADD operation is employed to perform the maximization over the parameters in Equation (26). Notice that maximizing parameters in PADDs is equivalent to maximizing parameters in the leaves. This is done by calling an optimization solver for a leaf in the PADD, where the result is also an ADD with one leaf, i.e., corresponds to a real value. For the example computed in Fig. 15, the optimization solver is called with $(1 - p_{12})(1 - p_{24})$ (Fig. 15d) and the

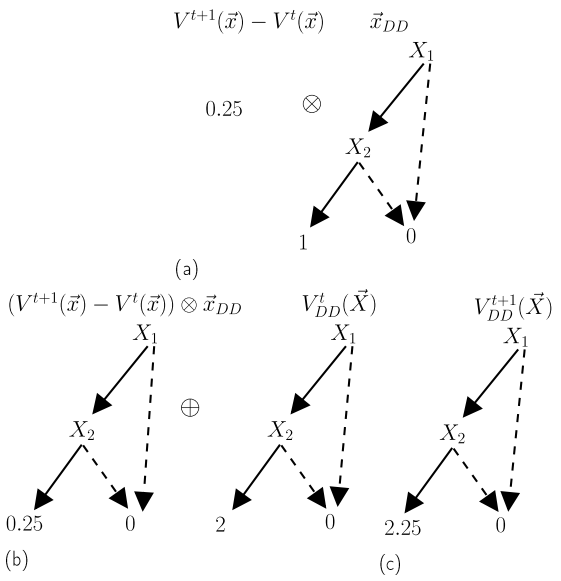


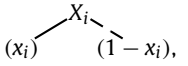
Fig. 16. Insertion of $V^{t+1}(\vec{x})$ into $V^t_{DD}(\vec{X})$: the new value 2.25 for state $\vec{x} = \{X_1 = 1, X_2 = 1\}$ computed by Equation (25), given that the old value was 2; a) Computes the product of the difference between $V^{t+1}(\vec{x})$ and $V^t(\vec{x})$ and a simple indicator function $\vec{x}_{DD}$ that takes the value 1 for state $\vec{x}$ and 0 otherwise; b) Takes the result and sum with the old value function $V^t_{DD}(\vec{X})$; c) Finally $V^{t+1}_{DD}(\vec{X})$ computed by $V^t_{DD} \oplus ((V^{t+1}(\vec{x}) - V^t(\vec{x})) \otimes \vec{x}_{DD})$.

set of constraints φ returning the probability values $\{p_{11} = 0, p_{12} = 0.8, p_{21} = 0.5, p_{22} = 0.5, p_{23} = 0.25, p_{24} = 0.15\}$ and the polynomial value that is used with *EvalPADD*$(C_{DD}(\vec{X}, a), \vec{x})$ to compute $Q^{t+1}(\vec{x}, a)$.

Finally, Equation (25) performs the minimization operation over the Q^{t+1} values for all actions, resulting in a constant value V^{t+1}. *FactRTDP-IP* must insert this new constant V^{t+1} into the current value function V^t_{DD} to obtain V^{t+1}_{DD}. This insertion is carried out by constructing a simple indicator function ADD $\vec{x}_{DD}$ that takes the value 1 for state $\vec{x}$ and 0 otherwise, i.e., given a variable assignment $\vec{v}$:

$$EvalADD(\vec{x}_{DD}, \vec{v}) = \begin{cases} 1 & \text{if } \vec{v} = \vec{x} \\ 0 & \text{otherwise.} \end{cases} \tag{29}$$

One way to construct the indicator function $\vec{x}_{DD}$ is to employ a sequential product operation $\bigotimes_{i=1}^{n} X_{iDD}$, where X_{iDD} is the single variable PADD:



that associates x_i to the true branch and $(1 - x_i)$ to the false branch.

Then, the new value $V^{t+1}(\vec{x})$ can be "masked into" the old value function $V^t_{DD}(\vec{X})$ to obtain the updated function $V^{t+1}_{DD}(\vec{X})$ by computing $V^t_{DD} \oplus ((V^{t+1}(\vec{x}) - V^t(\vec{x})) \otimes \vec{x}_{DD})$. An example is given in Fig. 16. Remember that we use *DD* for functions represented by ADDs or PADDs; $\vec{x}$ for representing a state and $\vec{X}$ for representing a vector of binary state variables.

factRTDP-IP sampling We also consider the three methods described in Section 6 for choosing the value of the probability parameters, i.e., *minimax_parameter_choice*, *predefined_parameter_choice* and *rand_parameter_cho- ice*. Note that for the second and third methods, it is necessary to compute the vertices for the factored transition credal sets (as was done in Section 6.1). After having the values for $\vec{p}$, factRTDP-IP carries out the sampling of each next state variable X'_i based on its CPT for the chosen action a, i.e.:

$$\vec{x}.\textsc{NextStateVar}(X'_i, a, \vec{p}) = x'_i \sim P(\cdot|\vec{x}, a), \tag{30}$$

where the probabilistic distribution $P(\cdot|\vec{x}, a)$ is taken from the leaves of the ADD returned by *ValPADDLeaves*$(pEvalPADD(P_{DD}(X'_i|pa_a(X'_i)), a), \vec{x}), \vec{p})$. Note that this ADD has a single variable X'_i (i.e., the result of evaluating P_{DD} with the variable assignment $\vec{x}$ and replace the probability parameters values $\vec{p}$). Fig. 17 shows an example of the process for choosing the next state $\vec{x}'$, given that the current state is $\vec{x} = \{X_1 = 1, X_2 = 1\}$ and the chosen parameter values are $\{p_{11} = 0,\ p_{12} = 0.8,\ p_{21} = 0.5,\ p_{22} = 0.5,\ p_{23} = 0.25,\ p_{24} = 0.15\}$.

The factRTDP-IP algorithm (Algorithm 6) initializes V_{DD} as an ADD with an admissible value function and runs trials until it achieves a goal state. For each visited state, the factRTDP-IP-Update algorithm is called (Algorithm 6, Line 16) and returns the updated value function and the parameter values which can be reused by the *minimax_parameter_choice*. After

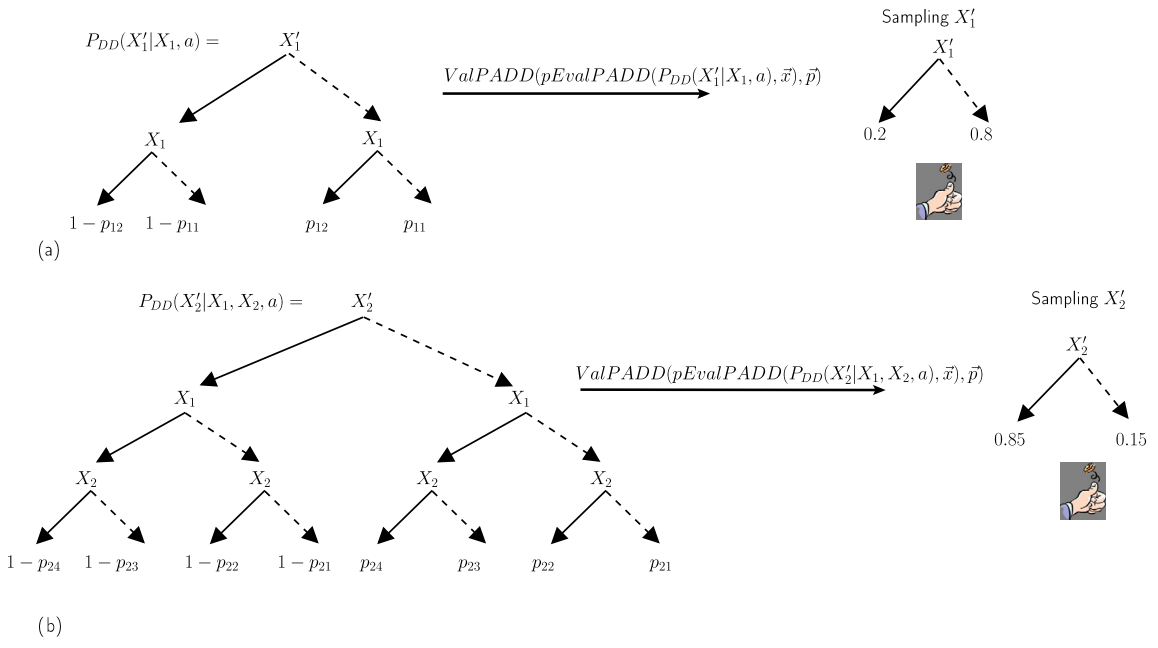


Fig. 17. FACTRTDP-IP sampling of state variables X_1' and X_2' given the visiting state $\vec{x} = \{X_1 = 1, X_2 = 1\}$ and the values for the parameters $\vec{p}$; a) Applying $ValPADD(pEvalPADD(P_{DD}(X_1'|X_1, a_1), \vec{x}), \vec{p})$ and sampling the value for X_1' with 0.2 for $X_1' = 1$ and 0.8 for $X_1' = 0$; b) $ValPADD(pEvalPADD(P_{DD}(X_2'|X_1, X_2, a_1), \vec{x}), \vec{p})$ and sampling X_2' with 0.85 for $X_1' = 1$ and 0.15 for $X_2' = 0$.

Algorithm 6: FACTRTDP-IP(V_{DD}^0, *FactSSPMDP − IP*, *maxtime*) $\rightarrow V_{DD}$.

```
1  begin
2      // Initialize V_DD with admissible value function V_DD^0
3      V_DD := V_DD^0
4      // CredalSetVerticesHash maps each credal set to its vertices
5      CredalSetVerticesHash := ∅
6      // Compute vertices and random points of all credal sets if needed
7      if (predefined_parameter_choice) then
8          CredalSetVerticesHash := ComputeAllCredalSetVertices()
9          p⃗ := ComputeRandomPointForAllCredalSets (
                   CredalSetVerticesHash)
10
11     while (not out of maxtime) do
12         visited.Clear() // Clear visited states stack
13         x⃗ := x⃗_0
14         while (x⃗ ∉ G) do
15             visited.Push(x⃗)
16             ⟨V_DD(x⃗), p⃗'⟩ := x⃗.factRTDP-IP-Update(V_DD)
17             if (minimax_parameter_choice) then
18                 p⃗ := p⃗'
19             a := x⃗.GreedyAction(V_DD)
20             x⃗ := x⃗.factRTDP-IP-Sampling(a, p⃗)
21
22         // Optimization: reverse-trial update
23         while ¬visited.Empty() do
24             x⃗ := visited.Pop()
25             ⟨V_DD(x⃗), p⃗'⟩ := x⃗.factRTDP-IP-Update(V_DD)
26
27     return V_DD
28 end
```

a greedy action (Algorithm 6, Line 19) has been chosen, the next state sampling is carried out (Algorithm 6, Line 20). From Line 23 to 25, the algorithm updates all the states in the trial path, from the end to the initial state.

The FACTRTDP-IP-UPDATE algorithm (Algorithm 7) updates the factored value function for state $\vec{x}$ as follows: after "*priming*" the variables (Algorithm 7, Line 4) of V_{DD}^t (by converting each X_i into X_i'), Q_{DD}^a is derived by computing Equation (26) in terms of PADD operations (Algorithm 7, Lines 5–7). After applying the *MaxParameterOut* PADD operation (Algorithm 7,

Algorithm 7: STATE::FACTRTDP-IP-UPDATE($V_{DD}^t, \vec{x}$) $\longrightarrow$ ($V_{DD}^{t+1}, \vec{p}$).

```
1  begin
2      for all a ∈ A do
3          // Multiply each CPT into V_DD^t(x⃗') and sum out
4          Q_DD^a := ConvertToPrimes(V_DD^t)
5          for all X_i' in Q_DD^a do
6              Q_DD^a := Q_DD^a ⊗ pEvalPADD(P_DD(X_i'|pa_a(X_i'), a), x⃗)
7              Q_DD^a := Σ_{x_i' ∈ X_i'} Q_DD^a
8          //call the non-linear solver for each PADD leaf (here the PADD only have
           one leaf)
9          ⟨Q_DD^a, p⃗_a⟩ := MaxParameterOut (Q_DD^a, φ);
10         Q_DD^a := EvalPADD(C_DD(X⃗, a), x⃗) ⊕ Q_DD^a
11     // Get updated value v for x⃗ and insert into V_DD^t
12     a := arg min_{a∈A} [Q_DD^a]
13     V^{t+1} := min_{a∈A} [Q_DD^a]
14     V_DD^{t+1} := FACTRTDP-IP-MASKEDVINTOADD (V_DD^t, x⃗, V^{t+1})
15     return (V_DD^{t+1}, p⃗_a)
16 end
```

Algorithm 8: FACTRTDP-IP-MASKEDVINTOADD($V_{DD}^t, \vec{x}, V^{t+1}$) $\longrightarrow V_{DD}^{t+1}$.

```
1 begin
2     V^t := EvalADD(V_DD^t, x⃗)
3     x⃗_DD := ⊗_{i=1}^n X_iDD
4     V_DD^{t+1} := V_DD^t ⊕ ((V^{t+1} − V^t) · x⃗_DD)
5     return V_DD^{t+1}
6 end
```

Algorithm 9: STATE::FACTRTDP-IP-SAMPLING($a, \vec{p}$) $\longrightarrow \vec{x}'$.

```
1  begin
2      // Sample each next state variable X_i' from DCN
3      //p⃗ starts with minimax_parameter_choice or predefined_parameter_choice
4      for all X_i' do
5          if (rand_parameter_choice) then
6              if (CredalSetVerticesHash.GET(⟨X_i', x⃗(pa_a(X_i')), a⟩)= NULL) then
7                  CredalSetVertices := COMPUTECREDALSETVERTICES
                   (X_i', x⃗(pa_a(X_i')), a)
8                  CredalSetVerticesHash.INSERT(⟨X_i', x⃗(pa_a(X_i')), a⟩,
                   CredalSetVertices)
9
10             p⃗ := COMPUTERANDOMPOINTCREDALSET (
                       X_i', x⃗(pa_a(X_i')), a, CredalSetVerticesHash)
11
12         x_i' := x⃗.NEXTSTATEVAR (X_i', a, p⃗) // According to Equation (30)
13     end
14     return x⃗' := (x_1', ..., x_n')
```

Line 9) the result is a real value. The computation of $V^{t+1}(\vec{x})$ yields a constant V^{t+1} for a known $\vec{x}$ (Algorithm 7, Line 13) and then factRTDP-IP inserts this new constant V^{t+1} into the current value function V_{DD}^t to obtain V_{DD}^{t+1} calling the method FACTRTDP-IP-MASKEDVINTOADD (Algorithm 8).

The FACTRTDP-IP-SAMPLING algorithm (Algorithm 9) samples a next state $\vec{x}'$ given a current state $\vec{x}$, action a and $\vec{p}$. We first choose values for the parameters $\vec{p}$, using *minimax_parameter_choice*, *predefined_parameter_choice* or *rand_parameter_choice*. If the *minimax_parameter_choice* method is employed, $\vec{p}$ has the same values that have been computed for the last update. If the *predefined_parameter_choice* method is used, $\vec{p}$ is only computed once at the beginning of Algorithm 6 (Lines 7–9) by first computing the vertices of all the credal sets (stored in a hash table called *CredalSetVerticesHash*), and then using those vertices to generate random points. Finally, if the *rand_parameter_choice* is used, $\vec{p}$ is computed for the tuple $\langle X_i', \vec{x}(pa_a(X_i')), a\rangle$ with $i = \{1, \cdots, n\}$, where a is the current greedy action and $\vec{x}(pa_a(X_i'))$ means the values of the parents of X_i' in the variable assignment $\vec{x}$, i.e., the configuration of the parents of X_i' in the current state $\vec{x}$ (Algorithm 9, Lines 5–10). Notice that the size of *CredalSetVerticesHash* will be the number of credal sets for the *predefined_parameter_choice* method and less or equal to this number for the *rand_parameter_choice* method, given the fact that not all the states will be visited. Thus, in the worst case scenario the size is equal to $n * 2^{\max_{a,X_i} |pa_a(X_i)|} * |A|$ (Equation (16)). Since we are considering binary variables and assume closed and convex credal sets, the number of vertices for each credal set is equal to 2. Thus, the space required by *CredalSetVerticesHash* is proportional to a constant times $n * 2^{\max_{a,X_i} |pa_a(X_i)|} * |A|$.

Algorithm 10: factLRTDP-IP(V^0_{DD}, $FactSSPMDP - IP$, $maxtime$, ϵ) $\rightarrow$ V_{DD}.

```
1  begin
2      // Initialize V_DD with admissible value function V^0_DD
3      V_DD := V^0_DD
4      // CredalSetVertices is a map with the credal set and its vertices
5      CredalSetVertices := ∅
6      // Compute vertices and random points of all credal sets if needed
7      if (predefined_parameter_choice) then
8          CredalSetVertices := ComputeAllCredalSetVertices ()
9          p⃗ := ComputeRandomPointForAllCredalSets (CredalSetVertices)
10
11     while (not out of maxtime ∧ ¬x⃗_0.Solved()) do
12         visited.Clear() // Clear visited states stack
13         x⃗ := x⃗_0
14         while (¬x⃗.Solved()) do
15             visited.Push(x⃗)
16             if (x⃗ ∈ G) then
17                 break
18
19             ⟨V_DD(x⃗), p⃗'⟩ := x⃗.factRTDP-IP-Update(V_DD)
20             if (minimax_parameter_choice) then
21                 p⃗ := p⃗'
22             a := x⃗.GreedyAction(V_DD)
23             x⃗ := x⃗.factRTDP-IP-Sampling(a, p⃗)
24
25         // Check state convergence
26         while ¬visited.Empty() do
27             x⃗ := visited.Pop()
28             if ¬x⃗.factCheckSolved-IP(V_DD, ε) then
29                 break
30
31
32     return V_DD
33 end
```

After defining the values for $\vec{p}$ we carry out the sampling of each next state variable X'_i (Algorithm 9, Line 12). This is done by using the ADD (of a single variable X'_i) returned by $ValPADDLeaves(pEvalPADD(P_{DD}(X'_i|pa_a(X'_i), a), \vec{x}), \vec{p})$, where we have the probabilities to do the sampling.

7.2. *factLRTDP-IP* algorithm

The Labeled Factored RTDP-IP algorithm, called factLRTDP-IP, (Algorithm 10) is similar to the factRTDP-IP, but has two main differences: (i) in the trial termination condition (Algorithm 10, Line 14), we determine if the state $\vec{x}$ is marked as solved; (ii) at the end of each trial (Algorithm 10, Lines 25–29) we check if each state in the trial can be labeled as solved through the factCheckSolved-IP procedure [8] (this is also done in a factored way as in [33]).

8. Experimental results

In this section, we empirically evaluate the proposed asynchronous algorithms, RTDP-IP, LRTDP-IP, factRTDP-IP and factLRTDP-IP, as opposed to SPUDD-IP [23], which is a synchronous algorithm that produces optimal policies (see Section 5.3). The algorithms were applied in three well-known planning domains: SysAdmin, proposed by Guestrin et al. (2003); Navigation and Triangle Tire world from the International Probabilistic Planning Competition (IPPC). SSP MDP-IP problems were generated for these domains, and modified by parameterizing the probability transition function and adding a set of constraints, as described below.

In **SysAdmin** [30], there are n computers $c_1, \ldots, c_n$ connected via a given topology (unidirectional ring, bidirectional ring and independent bidirectional rings) and $n+1$ actions: $reboot(c_1), \ldots, reboot(c_n)$ and $notreboot$ (i.e., no machine is rebooted). Let state variable x_i denote whether computer c_i is up and running ($x_i = 1$) or not ($x_i = 0$). Let $Conn(c_j, c_i)$ denote a connection from c_j to c_i. Formally, the transition probabilities have the following form:

$$P(x'_i = 1|\vec{x}, a) = \begin{cases} 1 & \text{if } a = reboot(c_i) \\ p_{i1} \cdot \frac{|\{x_j | j \neq i \wedge x_j = 1 \wedge Conn(c_j, c_i)\}| + 1}{|\{x_j | j \neq i \wedge Conn(c_j, c_i)\}| + 1} & \text{if } a \neq reboot(c_i) \wedge x_i = 1 \\ p_{i2} \cdot \frac{|\{x_j | j \neq i \wedge x_j = 1 \wedge Conn(c_j, c_i)\}| + 1}{|\{x_j | j \neq i \wedge Conn(c_j, c_i)\}| + 1} & \text{if } a \neq reboot(c_i) \wedge x_i = 0 \end{cases} \tag{31}$$

The probability parameters p_{i1} and p_{i2} are restricted by $0.85 + p_{i2} \leq p_{i1} \leq 0.95$. This domain has many transitions with imprecise probabilities and is a dense domain since most states are reachable in one step. There is no goal state, thus this is

an infinite horizon MDP-IP problem. To convert this problem to an SSP MDP-IP problem, we used the conversion described in [7], i.e., we added a goal state g and the transition probability P was transformed into P' as following:

$$P'(s'|s,a) = \begin{cases} \gamma P(s'|s,a) & \text{if } s' \neq g \\ 1-\gamma & \text{if } s' = g \end{cases} \tag{32}$$

Rewards were transformed into positive cost. Thus, the cost is 0 for the goal state and $C(\vec{x}) = n - \sum_{i=1}^{n} x_i$ for the others. The initial state is a single state where half of the computers are running and we consider the discount factor $\gamma = 0.9$ for the transformation.

In **Navigation** [16] (from IPPC 2011), a robot must walk in a grid of $n \times m$ cells and find a path to reach a goal cell, while avoiding dangerous cells that can make it disappear with a certain probability. The robot can walk across one cell per time in four directions (*north*, *south*, *east* and *west*) or can stay in the same place (*noop*). The factored version is given by $n \times m$ variables for representing if the robot is in the cell. The action of walking in any direction is deterministic but there is a probability that the robot will disappear in dangerous cells which is given by the probability parameter p_i subject to: $b_j \leq p_i \leq b_k + 0.1$, where b_j and b_k grow for cells near the goal. The location of the initial and goal cell are, respectively, $[1, m]$ and $[n, m]$. The safe cells are the cells in the row 1 and row n and the cells in the column 1. The cost is 0 when the robot is in the goal cell, otherwise it is 1. To reach the goal in this domain, the robot must walk in the longest path, i.e., walk in the cells in the border of the grid, where all cells are safe. It can be regarded as a sparse domain since, for any domain action, the number of reachable states in one step is a constant for all instances.

In **Triangle Tireworld** [36] (from IPPC 2005), the aim is to move a car from an initial location to a goal location through a sequence of roads. Each road connects two locations. The factored problem has one variable to indicate if the car has a spare tire, one to indicate if the car is flat-tired and n variables, one per location, to show if the car is in this location. There is a probability that the car can become flat-tired when moving from one location to another. In this situation the flat tire can be replaced with a spare tire. In addition, in some locations the car can be loaded with an extra spare tire to be used in the future. The probability for the parameter p_i of an SSP MDP-IP problem for the car to become flat-tired is subject to the constraint $b_j \leq p_i \leq b_k + 0.1$, where b_j and b_k are parameters, with high values in the cells close to the goal; the cost is 0 when the car is in the goal, otherwise it is 1. Like in Navigation domain, the car can reach the goal location safely moving in the longest path, where all locations have an extra spare to avoid the car be flat-tired with no spare in the path. This domain is also considered to be a sparse domain, i.e., the number of reachable states in one step is a constant.

For all the experiments we used a virtual machine running with 2 processors at 3.40 GHz and 5 GB of memory. In the case of SPUDD-IP, LRTDP-IP and factLRTDP-IP, we set $\epsilon = 0.01$ (convergence error) as the residual error used for all domains. All runs that did not complete in two hours or lacked enough memory to solve the problem were marked with DNF (*Did Not Finish*).

Since our Navigation and Triangle Tireworld domains have avoidable dead ends that are easy to be detected (dead-ends in which no action is available), we also implement a dead end detection procedure for all the algorithms. In RTDP-like algorithms a trial is interrupted when a dead-end is reached and its value is updated to ∞. In CheckSolved-IP procedure, if a dead-end is found, we also update its value to ∞ and ignore that state in the CheckSolved-IP's search.

The proposed algorithms were analyzed in terms of both the initial state convergence rate and convergence time. We also analyzed the algorithms w.r.t. the number of calls to a multilinear solver.

8.1. The initial state convergence rate

Analyzing the convergence behavior of SPUDD-IP, RTDP-IP, LRTDP-IP, factRTDP-IP and factLRTDP-IP, we examined the value of the initial state s_0 during the time, until convergence, for three planning problems: instance 18 of Navigation; 17 of Triangle Tireworld; and instance 6 of SysAdmin.[4] Each of these problems corresponds to the largest instance of the corresponding domain for which all of the algorithms were able to return a policy (as will be seen in Section 8.2).

We ran the algorithms to assess the three methods of choosing the probability for sampling the next state, i.e., *minimax_parameter_choice*, *predefined_parameter_choice* and *rand_parameter_choice* methods. Intuitively, we expected that using the *minimax_parameter_choice* method could cause $V(s_0)$ to converge faster. However, in each of the three methods, $V(s_0)$ converges at the same rate. In this section we only show the results for the *minimax_parameter_choice* method and leave comparisons of these methods in terms of time to the next section.

In Fig. 18 it can be seen that for sparse domains (Navigation and Triangle Tireworld), asynchronous algorithms (e.g., with *minimax_para- meter_choice* method) perform much better than SPUDD-IP. For the Navigation instance it is clear that RTDP-IP, LRTDP-IP, factRTDP-IP and factLRTDP-IP quickly reach a near-optimal value in the first 4 seconds, while SPUDD-IP takes more than 5000 seconds. A similar behavior can be observed in the Triangle Tireworld instance. However, for the SysAdmin instance (Fig. 18, bottom), which is a dense domain, SPUDD-IP has a better convergence rate than the asynchronous algorithms, RTDP-IP, LRTDP-IP, factRTDP-IP and factLRTDP-IP. It should be noted that in this domain all states are reachable, as a result of probabilistic exogenous events, and hence asynchronous algorithms are unable to take advantage of

[4] The number of instance represent the number of variables in this problem, so the corresponding number of states is $2^{\#variables}$.

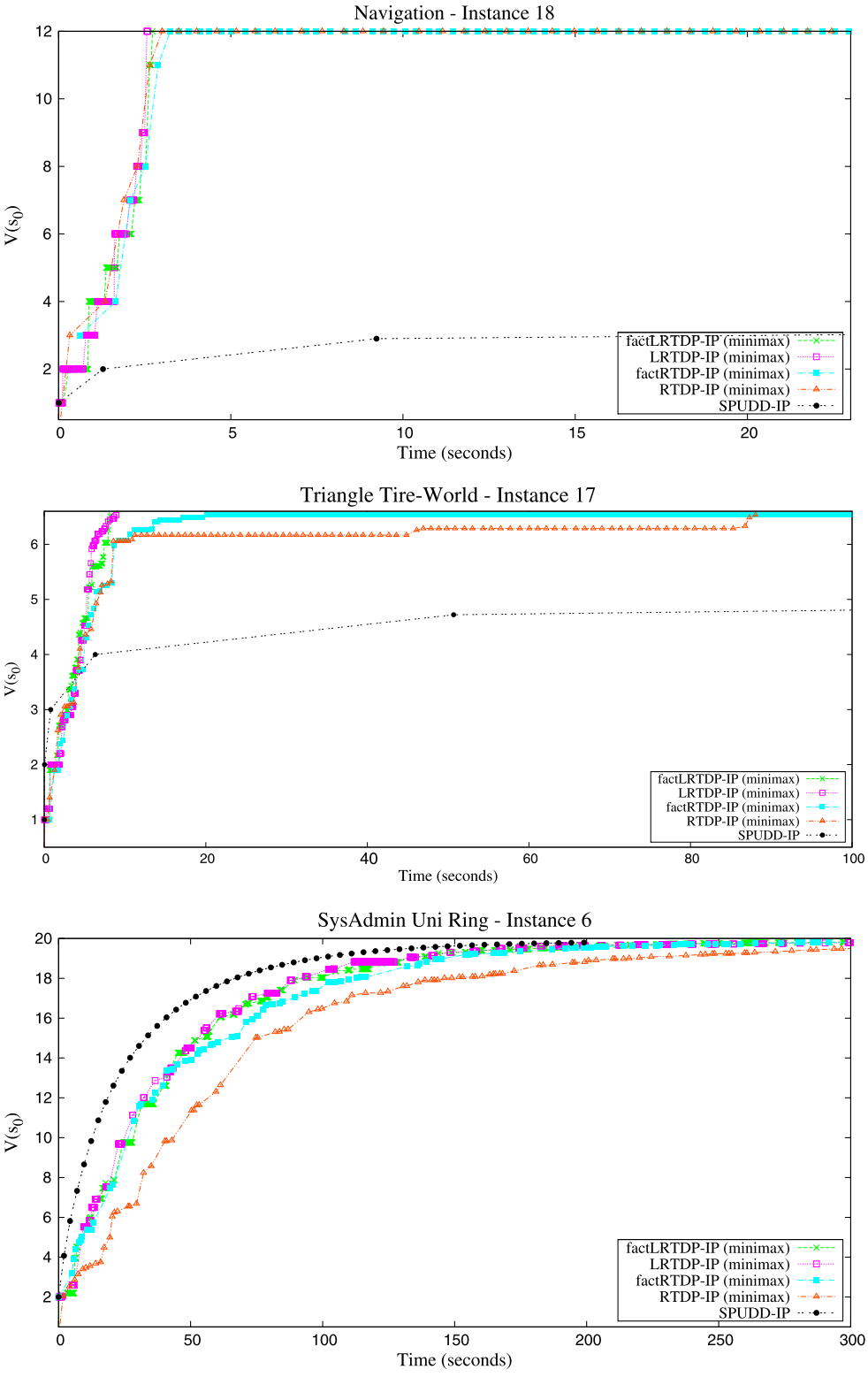


Fig. 18. Value of the initial state for instance 18 of the NAVIGATION domain, instance 17 of the TRIANGLE TIREWORLD domain and instance 6 of the SYSADMIN domain.

the initial state information. In this case, asynchronous algorithms consume more time updating state by state than SPUDD-IP which updates all the states at each step. Moreover, when labeling solved states, LRTDP-IP and factLRTDP-IP exploit the greedy graph structure, which can be very costly when the graph is dense. In fact, in a dense domain, the CheckSolved procedure just wastes time, since all the states may converge together. Note that since SPUDD-IP performs a synchronous dynamic programming, $V(s_0)$ changes uniformly (see the SPUDD-IP curve at the bottom of Fig. 18) at each iteration, while, for the asynchronous algorithms, the value of s_0 may not change during certain time intervals (depending on which states were updated during the trials).

Since LRTDP-IP and factLRTDP-IP converge faster than RTDP-IP and factRTDP-IP for all the domains and instances, in the rest of our analysis we only include the results of LRTDP-IP and factLRTDP-IP.

8.2. Convergence time per problem instance

Fig. 19 shows the global convergence time of SPUDD-IP, LRTDP-IP and factLRTDP-IP for different instances of the Navigation, Triangle Tireworld and SysAdmin domains. We also compare the convergence time for the three methods for sampling next state in factLRTDP-IP, i.e., the *minimax_parameter_choice* method (abbreviated as ***minimax***), the *predefined_parameter_choice* (***pre***) and *rand_ parameter_choice* (***rand***).

For *sparse* domains such as Navigation and Triangle Tireworld, the asynchronous algorithms LRTDP-IP and factLRTDP-IP show up to three orders of magnitude speedup unlike the synchronous dynamic programming approach, SPUDD-IP. Additionally, while SPUDD-IP is not able to return a policy for instances with more than 20 variables, LRTDP-IP and factLRTDP-IP return optimal policies for instances with up to 121 variables, in the Navigation domain and for instances up to 80 variables in the Triangle Tireworld domain. As expected, this happens because in these domains the LRTDP-IP and factLRTDP-IP algorithms only perform the Bellman backup in the set of states reachable from the initial state, while SPUDD-IP has to update all the states at each iteration. Note also that LRTDP-IP shows up to 2 times speedup in comparison to factLRTDP-IP, so the difference is not significant.

We also show that there is no significant difference in terms of time for the *minimax_parameter_choice*, *predefined_parameter_choice* and *rand_parameter_ choice* methods for the three domains we have analyzed. An explanation is that for these domains, $\max_{a,X_i} |pa_a(X_i)| = 2$ or 3 and $|\mathcal{K}| \leq n * 2^3 * |A|$ (Equation (16)). Notice that there is a small number of credal sets and the number of vertices of each credal set is up to 2 (since we are working with boolean variables). Thus, there is no significant extra cost involved in computing the credal set vertices or generating random points using Equation (22) for the *predefined_parameter_choice* and *rand_parameter_choice* methods, which causes no difference in time when sampling in compliance with *minimax_parameter_choice*, *predefined_parameter_choice* or *rand_parameter_choice* methods. Thus we believe that for domains with a large variable dependency and a large number of vertices for the credal sets, the *rand_parameter_choice* method might show a worse performance.

In a dense domain such as SysAdmin, the results are not so promising. It is clear that LRTDP-IP and factLRTDP-IP take more time to converge (approximately $4\times$) than SPUDD-IP. As explained in the previous section, in a dense domain where the convergence depends on all the state values, the synchronous dynamic programming algorithm achieves a better performance.

8.3. Number of calls to a multilinear solver

In showing the relationship between the number of calls to a multilinear solver and the overall solution time (convergence time), a comparison was made between SPUDD-IP, LRTDP-IP and factLRTDP-IP with the *minimax_parameter_choice* method to choose the values for each p_i for different instances of the Triangle Tireworld domain.

Fig. 20 shows that for each of the analyzed approaches, there is a strong correlation between the running time and the number of calls to a multilinear solver which suggests that the number of calls dominates the running time. Note that the number of calls to a multilinear solver of SPUDD-IP grows faster than the number of calls to a multilinear solver of the LRTDP-IP and factLRTDP-IP algorithms. This is because SPUDD-IP updates all the states while LRTDP-IP and factLRTDP-IP only update the reachable states and the number of calls to a multilinear solver is proportional to the number of updates.

9. Related work

As mentioned in Section 3, a particular sub-class of MDP-IP is the *Bounded-parameter Markov Decision Process (BMDP)* [29], where the probabilities and rewards are specified by intervals over a set of enumerated states. Algorithms to solve an (enumerated) BMDP can find an optimal policy without requiring expensive optimization techniques, although they have some limitations: they do not deal either with the general constraints or factored inputs. Remember that the translation of the problems described in factored forms into enumerated MDP-IPs result in polynomial expressions in the transition probabilities.

Another sub-class of MDP-IPs is the *Markov Decision Process with Set-valued Transitions (MDP-ST)* [49], where probability distributions are given for finite sets of states. In this model there are two varieties of uncertainty: a probabilistic selection of a reachable set and a non-deterministic choice of a successor state from the reachable set. Algorithms to solve this

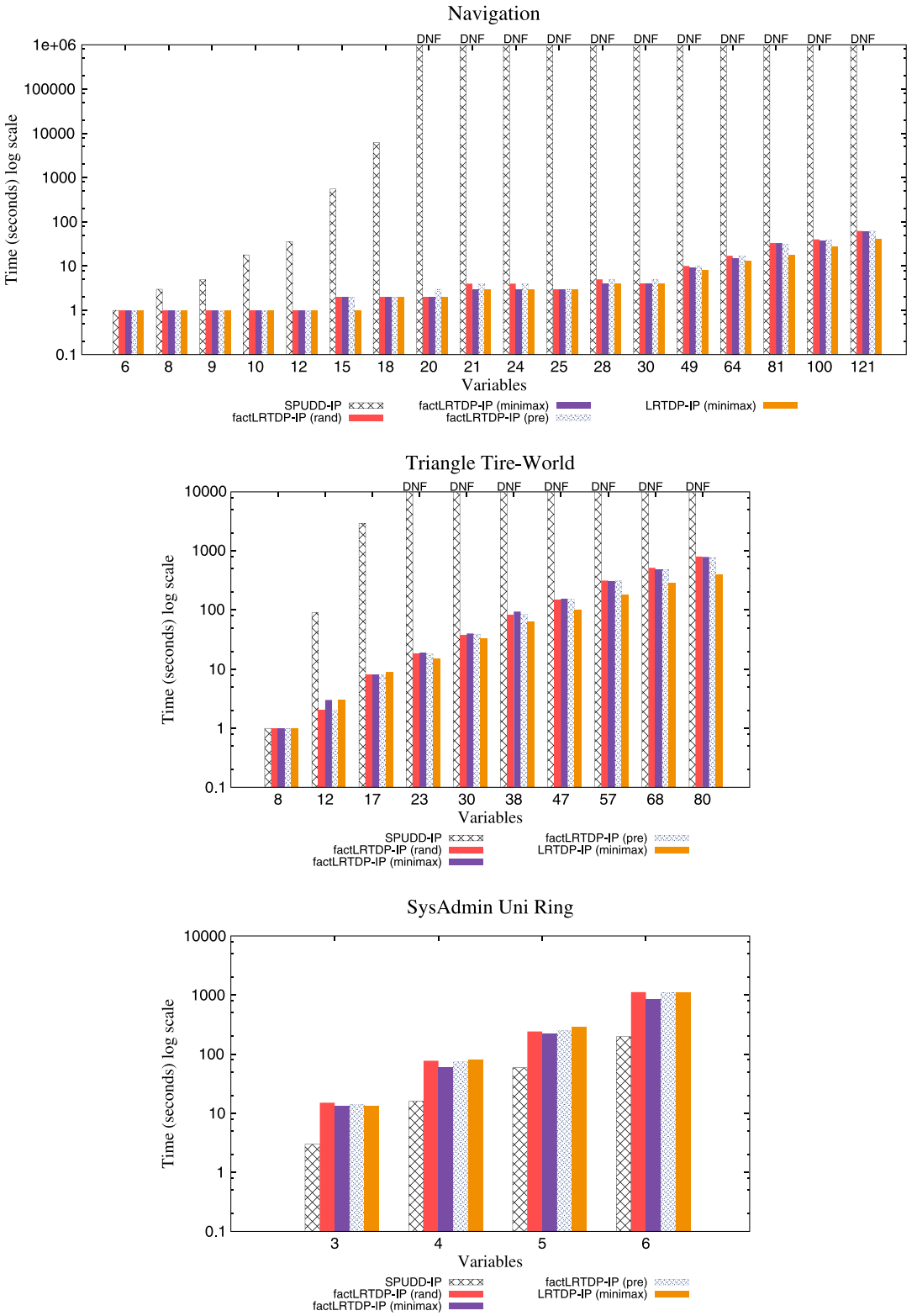


Fig. 19. Domain instances convergence time of the SPUDD-IP, LRTDP-IP and factLRTDP-IP in logarithmic scale, for the NAVIGATION, TRIANGLE TIREWORLD and SYSADMIN UNI-RING domains.

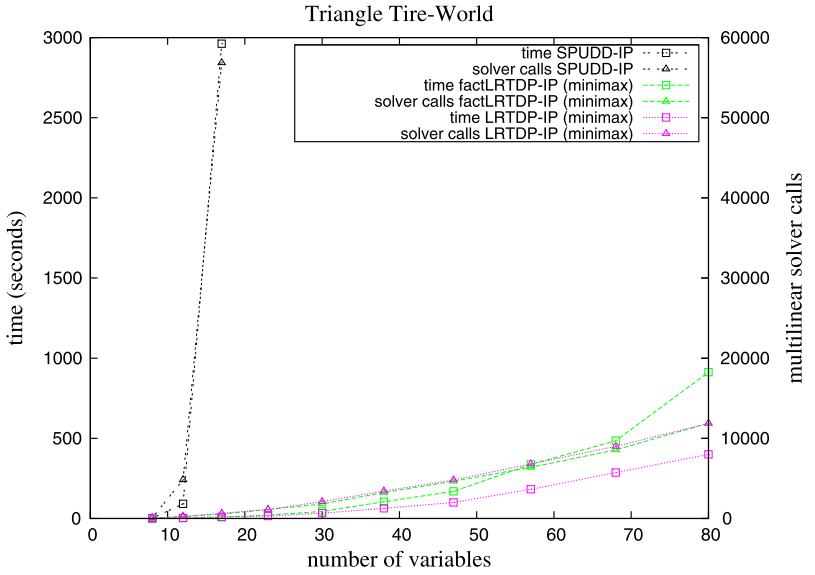


Fig. 20. Performance time and number of calls to the optimization solver for the algorithms SPUDD-IP, LRTDP-IP and factLRTDP-IP with *minimax_parameter_choice* method.

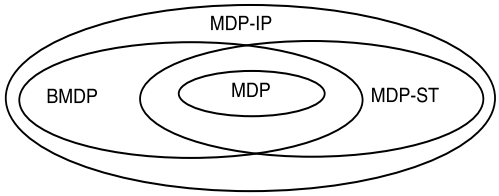


Fig. 21. Relationship between MDP-IP and its sub-classes [49].

problem can also find an optimal policy without requiring expensive optimization techniques since the Bellman principle for MDP-STs can be simplified to a great extent [49].

Fig. 21 shows the relationship between MDP, BMDP, MDP-ST and MDP-IP. Notice that BMDP and MDP-ST do not have the same representational power, i.e., some MDP-ST problems cannot be reduced to BMDP, and vice versa. Note also that since BMDPs and MDP-STs are special cases of MDP-IPs, we can represent them as MDP-IPs; thus the algorithms for MDP-IPs clearly apply to both BMDPs and MDP-STs [49].

Alternative frameworks that consider imprecise transition models that can be captured by MDP-IPs are Contingent planning [32] and Possibilistic approaches [41,25], even though these models are not probabilistic. Contingent planning [32] is the task of generating a conditional plan when there is uncertainty about the initial state and non-deterministic action effects, but with the ability to sense some aspects of the current state. Possibilistic approaches [42,41,25] take a different view about transition uncertainty that is modeled as qualitative possibility distributions over S instead of probability distributions.

Although there is no work on *factored* asynchronous algorithms for SSP MDP-IPs, there are a few studies in the literature on factored asynchronous algorithms for SSP MDPs: sRTDP [27], symbolic LAO* [26], factRTDP [33], factLRTDP [33] and factBRTDP [22]. While sRTDP [27] updates an abstract state (a group of similar states), factRTDP and factLRTDP [33] update a single state in the trial-based simulations. The factBRTDP algorithm is the factored version of the Bounded RTDP algorithm [37] and it is not as efficient as factLRTDP since it is very costly to update both a lower and upper bound value function. The factored algorithms proposed in this paper are based on the algorithms factRTDP and factLRTDP [33].

10. Conclusion

In this paper we have proposed a real-time dynamic programming algorithm that exploits the initial state of knowledge for SSP MDP-IP given in terms of a general set of constraints that can also be applied to factored SSP MDP-IP. To the best of our knowledge, asynchronous algorithms for SSP MDP-IPs have not been proposed before. We first defined an enumerated algorithm, called RTDP-IP, and proved that, despite the imprecision of the transition probabilities, the RTDP-IP algorithm converges for optimal values if the initial value of all the states is admissible and the sampling method does not avoid any state.

We have also defined the LRTDP-IP algorithm and were encouraged by the efficiency of SPUDD-IP [31], to propose two factored algorithms, named factRTDP-IP and factLRTDP-IP. These factored asynchronous algorithms get the best of SPUDD-IP and enumerated RTDP-IP approaches: they efficiently represent the value function and only visit the reachable states from s_0. However, in our experiments LRTDP-IP shows up to 2 times speedup in comparison to factLRTDP-IP.

For *sparse* domains such as NAVIGATION and TRIANGLE TIRE WORLD, LRTDP-IP and factLRTDP-IP yield up to three orders of magnitude speedup in comparison to SPUDD-IP. Additionally, while SPUDD-IP is not able to return a policy for instances with more than 20 variables, LRDP-IP and factLRTDP-IP return optimal policies for instances up to 121 variables in the NAVIGATION domain and for instances up to 80 variables in the TRIANGLE TIRE WORLD domain.

Another important result for the factLRTDP-IP algorithm is that, if the maximum number of parents in the dynamic credal networks is small and the number of vertices of each credal set is small, there is no significant extra cost involved in calculating the credal set vertices and generating random points for the *predefined_parameter_choice* and *rand_parameter_choice* methods. This causes no difference in time when sampling according to *minimax_parameter_choice*, *predefined_parameter_choice* or *rand_parameter_choice* methods.

As expected, in a dense domain, SPUDD-IP has a better convergence rate than our proposed asynchronous algorithms. The reason for this is that for dense domains a large number of states can be reached from the initial state and, in this case, there is no advantage in exploring the state space in an asynchronous fashion.

References

[1] David Avis, LRS: A Revised Implementation of the Reverse Search Vertex Enumeration Algorithm, Birkhäuser-Verlag, 2000, pp. 177–198.
[2] Ruth Iris Bahar, Erica A. Frohm, Charles M. Gaona, Gary D. Hachtel, Enrico Macii, Abelardo Pardo, Fabio Somenzi, Algebraic decision diagrams and their applications, in: Proceedings of the International Conference on Computer-Aided Design, ICCAD, Los Alamitos, CA, USA, IEEE Computer Society Press, 1993, pp. 188–191.
[3] Andrew G. Barto, Steven J. Bradtke, Satinder P. Singh, Learning to act using real-time dynamic programming, Artif. Intell. 72 (1–2) (1995) 81–138.
[4] Dimitri P. Bertsekas, Distributed dynamic programming, IEEE Trans. Automat. Control 27 (1982) 610–617.
[5] Dimitri P. Bertsekas, Dynamic Programming and Optimal Control, vol. 1, Athena Scientific, Belmont, MA, 1995.
[6] Dimitri P. Bertsekas, John N. Tsitsiklis, An analysis of stochastic shortest path problems, Math. Oper. Res. 16 (3) (1991) 580–595.
[7] Dimitri P. Bertsekas, John N. Tsitsiklis, Neuro-Dynamic Programming, Athena Scientific, Belmont, MA, 1996.
[8] Blai Bonet, Hector Geffner, Labeled RTDP: improving the convergence of real-time dynamic programming, in: Proceedings of International Conference on Automated Planning and Scheduling, 2003, pp. 12–21.
[9] Craig Boutilier, Richard Dearden, Moises Goldszmidt, Stochastic dynamic programming with factored representations, Artif. Intell. 121 (1–2) (2000) 49–107.
[10] Craig Boutilier, Nir Friedman, Moises Goldszmidt, Daphne Koller, Context-specific independence in Bayesian networks, in: Proceedings of the 12th International Conference on Uncertainty in Artificial Intelligence, 1996, pp. 115–123.
[11] Craig Boutilier, Steve Hanks, Thomas Dean, Decision-theoretic planning: structural assumptions and computational leverage, J. Artif. Intell. Res. 11 (1999) 1–94.
[12] Randal E. Bryant, Symbolic boolean manipulation with ordered binary-decision diagrams, ACM Comput. Surv. 24 (3) (1992) 293–318.
[13] Daniel Bryce, Michael Verdicchio, Seungchan Kim, Planning interventions in biological networks, ACM Trans. Intell. Syst. Technol. 1 (2) (2010) 111.
[14] Olivier Buffet, Douglas Aberdeen, Robust planning with LRTDP, in: Proceedings of International Joint Conferences on Artificial Intelligence, 2005, pp. 1214–1219.
[15] Olivier Buffet, Douglas Aberdeen, Policy-gradient for robust planning, in: Proceedings of the ECAI'06 Workshop on Planning, Learning and Monitoring with Uncertainty and Dynamic Worlds, PLMUDW'06, 2006.
[16] Amanda Jane Coles, Andrew Coles, Angel García Olaya, Sergio Jiménez, Carlos Linares López, Scott Sanner, Sungwook Yoon, A survey of the seventh international planning competition, AI Mag. 33 (1) (2012).
[17] Fabio G. Cozman, Credal networks, Artif. Intell. 120 (2000) 199–233.
[18] Fabio G. Cozman, Graphical models for imprecise probabilities, Int. J. Approx. Reason. 39 (2–3) (2005) 167–184.
[19] Shulin Cui, Jigui Sun, Minghao Yin, Shuai Lu, Solving uncertain Markov decision problems: an interval-based method, in: Proceedings ICNC (2), 2006, pp. 948–957.
[20] Aniruddha Datta, Ashish Choudhary, Michael L. Bittner, Edward R. Dougherty, External control in Markovian genetic regulatory networks, Mach. Learn. 52 (1–2) (2003) 169–191.
[21] Thomas Dean, Keiji Kanazawa, A model for reasoning about persistence and causation, Comput. Intell. 5 (3) (1990) 142–150.
[22] Karina Valdivia Delgado, Cheng Fang, Scott Sanner, Leliane Nunes de Barros, Symbolic bounded real-time dynamic programming, in: SBIA, 2010, pp. 193–202.
[23] Karina Valdivia Delgado, Scott Sanner, Leliane Nunes de Barros, Efficient solutions to factored MDPs with imprecise transition probabilities, Artif. Intell. 175 (9–10) (2011) 1498–1527.
[24] Luc Devroye, Non-Uniform Random Variate Generation, Springer-Verlag, 1986.
[25] Nicolas Drougard, Florent Teichteil-Konigsbuch, Jean-Loup Farges, Didier Dubois, Qualitative possibilistic mixed-observable MDPs, in: Proceedings of the Conference on Uncertainty in Artificial Intelligence, UAI, Association for Uncertainty in Artificial Intelligence, 2013, pp. 192–201.
[26] Zhengzhu Feng, Eric A. Hansen, Symbolic LAO* search for factored Markov decision processes, in: Proceedings of the AIPS-02 Workshop on Planning via Model Checking, 2002, pp. 49–53.
[27] Zhengzhu Feng, Eric A. Hansen, Shlomo Zilberstein, Symbolic generalization for on-line planning, in: Proceedings of the 19th International Conference on Uncertainty in Artificial Intelligence, 2003, pp. 209–216.
[28] Hector Geffner, Blai Bonet, A Concise Introduction to Models and Methods for Automated Planning, vol. 8, Morgan & Claypool Publishers, 2013.
[29] Robert Givan, Sonia Leach, Thomas Dean, Bounded-parameter Markov decision processes, Artif. Intell. 122 (2000) 71.
[30] Carlos Guestrin, Daphne Koller, Ronald Parr, Shobha Venkataraman, Efficient solution algorithms for factored MDPs, J. Artif. Intell. Res. 19 (2003) 399–468.
[31] Jesse Hoey, Robert St-Aubin, Alan Hu, Craig Boutilier, SPUDD: stochastic planning using decision diagrams, in: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann, 1999, pp. 279–288.
[32] Jörg Hoffmann, Ronen I. Brafman, Contingent planning via heuristic forward search with implicit belief states, in: Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling, ICAPS, 2005, pp. 71–80.

[33] Mijail Gamarra Holguin, Planejamento probabilístico usando programação dinâmica assíncrona e fatorada, Master's thesis, IME-USP, 2013, available in: http://www.teses.usp.br/teses/disponiveis/45/45134/tde-14042013-131306/.
[34] Ronald A. Howard, Dynamic Programming and Markov Process, The MIT Press, 1960.
[35] Andrey Kolobov Mausam, Daniel S. Weld, A theory of goal-oriented mdps with dead ends, in: Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence, Catalina Island, CA, USA, August 14–18, 2012, 2012, pp. 438–447.
[36] Iain Little, Sylvie Thiébaux, Probabilistic planning vs. replanning, in: ICAPS Workshop on IPC: Past, Present and Future, 2007.
[37] H. Brendan McMahan, Maxim Likhachev, Geoffrey J. Gordon, Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees, in: Proceedings of the 22nd International Conference on Machine Learning, ICML, ACM, New York, NY, USA, 2005, pp. 569–576.
[38] Ranadip Pal, Aniruddha Datta, Edward R. Dougherty, Robust intervention in probabilistic boolean networks, IEEE Trans. Signal Process. 56 (3) (2008) 1280–1294.
[39] Stephen D. Patek, Dimitri P. Bertsekas, Stochastic shortest path games, SIAM J. Control Optim. 37 (3) (1999) 804–824.
[40] Martin L. Puterman, Markov Decision Processes, Wiley Series in Probability and Mathematical Statistics, John Wiley and Sons, New York, 1994.
[41] Régis Sabbadin, A possibilistic model for qualitative sequential decision problems under uncertainty in partially observable environments, in: Proceedings of the Conference on Uncertainty in Artificial Intelligence, UAI, 1999, pp. 567–574.
[42] Régis Sabbadin, Hélene Fargier, Jérome Lang, Towards qualitative approaches to multi-stage decision making, Int. J. Approx. Reason. 19 (34) (1998) 441–471.
[43] Scott Sanner, Robby Goetschalckx, Kurt Driessens, Guy Shani, Bayesian real-time dynamic programming, in: Proceedings of International Joint Conferences on Artificial Intelligence, 2009, pp. 1784–1789.
[44] Jay K. Satia, Roy E. Lave Jr., Markovian decision processes with uncertain transition probabilities, Oper. Res. 21 (1970) 728–740.
[45] Trey Smith, Reid G. Simmons, Focused real-time dynamic programming for MDPs: squeezing more out of a heuristic, in: Proceedings of the National Conference on Artificial Intelligence, AAAI, 2006.
[46] Robert St-Aubin, Jesse Hoey, Craig Boutilier, APRICODD: approximate policy construction using decision diagrams, in: Proceedings of Neural Information Processing Systems, NIPS, MIT Press, 2000, pp. 1089–1095.
[47] Richard S. Sutton, Andrew G. Barto, Introduction to Reinforcement Learning, 1st edition, MIT Press, Cambridge, MA, USA, 1998.
[48] Florent Teichteil-Königsbuch, Stochastic safest and shortest path problems, in: Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22–26, 2012, Toronto, Ontario, Canada, 2012.
[49] Felipe W. Trevizan, Fabio G. Cozman, Leliane N. de Barros, Planning under risk and Knightian uncertainty, in: Proceedings of International Joint Conferences on Artificial Intelligence, Hyderabad, India, 2007, pp. 2023–2028.
[50] Chelsea C. White III, Hank K. El-Deib, Markov decision processes with imprecise transition probabilities, Oper. Res. 42 (4) (1994) 739–749.
[51] Stefan J. Witwicki, Francisco S. Melo, Jesus Capitan, Matthijs T.J. Spaan, A flexible approach to modeling unpredictable events in MDPs, in: Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling, ICAPS, 2013, pp. 260–268.
[52] Minghao Yin, Jianan Wang, Wenxiang Gu, Solving planning under uncertainty: quantitative and qualitative approach, in: Proceedings IFSA (2), 2007, pp. 612–620.
[53] Nevin L. Zhang, David Poole, A simple approach to Bayesian network computations, in: Proceedings of the Tenth Canadian Conference on Artificial Intelligence, 1994, pp. 171–178.