Acta Informatica (2017) 54:545–587
DOI 10.1007/s00236-016-0255-4

ORIGINAL ARTICLE

Symblicit algorithms for mean-payoff and shortest path in monotonic Markov decision processes

Aaron Bohy[1] · **Véronique Bruyère**[1] · **Jean-François Raskin**[2] · **Nathalie Bertrand**[3]

Received: 15 April 2015 / Accepted: 9 January 2016 / Published online: 1 February 2016

Abstract When treating Markov decision processes (MDPs) with large state spaces, using explicit representations quickly becomes unfeasible. Lately, Wimmer et al. have proposed a so-called symblicit algorithm for the synthesis of optimal strategies in MDPs, in the quantitative setting of expected mean-payoff. This algorithm, based on the strategy iteration algorithm of Howard and Veinott, efficiently combines symbolic and explicit data structures, and uses binary decision diagrams as symbolic representation. The aim of this paper is to show that the new data structure of pseudo-antichains (an extension of antichains) provides another interesting alternative, especially for the class of monotonic MDPs. We design efficient pseudo-antichain based symblicit algorithms (with open source implementations) for two quantitative settings: the expected mean-payoff and the stochastic shortest path. For two practical applications coming from automated planning and LTL synthesis, we report promising experimental results w.r.t. both the run time and the memory consumption. We also show that a variant of pseudo-antichains allows to handle the infinite state spaces underlying the qualitative verification of probabilistic lossy channel systems.

1 Introduction

Markov decision processes [4,41] (MDPs) are rich models that exhibit both nondeterministic choices and stochastic transitions. Model-checking and synthesis algorithms for MDPs exist for logical properties expressible in the logic PCTL [28], a stochastic extension of CTL [18], and are implemented in tools like PRISM [35], MODEST [29], MRMC [33] …There also exist algorithms for *quantitative properties* such as the long-run average reward (mean-

✉ Véronique Bruyère
Veronique.Bruyere@umons.ac.be

1 Université de Mons, 20 Place du Parc, 7000 Mons, Belgium

2 Université Libre de Bruxelles, Campus de la Plaine, CP212, 1050 Brussels, Belgium

3 INRIA Rennes Bretagne-Atlantique, Campus Universitaire de Beaulieu, 35042 Rennes Cedex, France

payoff) or the stochastic shortest path, that have been implemented in tools like QUASY [17] and PRISM [45].

There are two main families of algorithms for MDPs. First, *value iteration* algorithms assign values to states of the MDPs and refine locally those values by successive approximations. If a fixpoint is reached, the value at a state s represents a probability or an expectation that can be achieved by an optimal strategy that resolves the choices present in the MDP starting from s. This value can be, for example, the maximal probability to reach a set of goal states. Second, *strategy iteration* algorithms start from an arbitrary strategy and iteratively improve the current strategy by local changes up to the convergence to an optimal strategy. Both methods have their advantages and disadvantages. Value iteration algorithms usually lead to easy and efficient implementations, but in general the fixpoint is not guaranteed to be reached in a finite number of iterations, and so only approximations are computed. On the other hand, strategy iteration algorithms have better theoretical properties as convergence towards an optimal strategy in a finite number of steps is usually ensured, but they often require to solve systems of linear equations, and so they are more difficult to implement efficiently.

When considering large MDPs, that are obtained from high-level descriptions or as the product of several components, explicit methods often exhaust available memory and are thus impractical. This is the manifestation of the well-known *state explosion problem*. In non-probabilistic systems, symbolic data structures such as binary decision diagrams (BDDs) have been investigated [16] to mitigate this phenomenon. For probabilistic systems, multi-terminal BDDs (MTBDDs) are useful but they are usually limited to systems with around 10^6 or 10^7 states only. Also, as mentioned above, some algorithms for MDPs rely on solving linear systems, and there is no easy use of BDD-like structures for implementing such algorithms.

Recently, Wimmer et al. [46] have proposed a method that *mixes* symbolic and explicit representations to efficiently implement the Howard [31] and Veinott [43] strategy iteration algorithm to synthesize optimal strategies for mean-payoff objectives in MDPs. Their solution is as follows. First, the MDP is represented and handled symbolically using MTBDDs. Second, a strategy is fixed symbolically and the MDP is transformed into a Markov chain (MC). To analyze this MC, a linear system needs to be constructed from its state space. As this state space is potentially huge, the MC is first reduced by *lumping* [15,34] (bisimulation reduction), and then a (hopefully) compact linear system can be constructed and solved. Solutions to this linear system allow to show that the current strategy is optimal, or to obtain sufficient information to improve it. A new iteration is then started. The main difference between this method and the other methods proposed in the literature is its *hybrid nature*: it is symbolic for handling the MDP and for computing the lumping, and it is explicit for the analysis of the reduced MC. This is why the authors of [46] have coined their approach *symblicit*.

Contributions. In this paper, we build on the symblicit approach described above. Our contributions are threefold. First, we show that the symblicit approach and strategy iteration can also be efficiently applied to the *stochastic shortest path* problem. We start from an algorithm proposed by Bertsekas and Tsitsiklis [7] with a preliminary step of de Alfaro [19], and we show how to cast it in the symblicit approach. Second, we show that alternative data structures can be more efficient than BDDs or MTBDDs for implementing a symblicit approach, both for mean-payoff and stochastic shortest path objectives. In particular, we consider a natural class of MDPs with *monotonic properties* on which our alternative data structure is more efficient. For such MDPs, as for subset constructions in automata theory

[21,47], antichain-based data structures usually behave better than BDDs. The application of antichains to monotonic MDPs requires nontrivial extensions: for instance, to handle the lumping step, we need to generalize existing antichain-based data structures in order to be closed under negation. To this end, we introduce a new data structure called *pseudo-antichain*. Third, we have implemented our algorithms and we show that they are more efficient than existing solutions on natural examples of monotonic MDPs. We show that monotonic MDPs naturally arise in probabilistic planning [9] and when optimizing controllers synthesized from LTL specifications with mean-payoff objectives [12]. To show the variety of applications of pseudo-antichains, we also propose a third completely different application in the context of qualitative verification of probabilistic lossy channel systems [2].

Structure of the paper. In Sect. 2, we recall the useful definitions, and introduce the notion of monotonic MDP. In Sect. 3, we recall strategy iteration algorithms for mean-payoff and stochastic shortest path objectives, and we present the symblicit version of those algorithms. We introduce the notion of pseudo-antichains in Sect. 5, and we describe our pseudo-antichain-based symblicit algorithms in Sect. 6. In Sect. 7, we propose two applications of the symblicit algorithms and give experimental results, we also give a third application in the context of lossy channel systems. Finally in Sect. 8, we summarize our results. This paper is an extended version of [13] with complete proofs, detailed algorithms, and the additional application in the context of probabilistic lossy channel systems.

2 Preliminaries

In this section, we recall useful definitions and we introduce the notion of monotonic Markov decision processes. We also state the problems that we study.

2.1 Functions and probability distributions

For any (partial or total) function f, we denote by $\mathsf{Dom}(f)$ the domain of definition of f. For all sets A, B, we denote by $\mathcal{F}_{\text{tot}}(A, B) = \{f : A \to B \mid \mathsf{Dom}(f) = A\}$ the set of total functions from A to B. A *probability distribution* over a finite set A is a total function $\pi : A \to [0, 1]$ such that $\sum_{a \in A} \pi(a) = 1$. Its *support* is the set $\mathsf{Supp}(\pi) = \{a \in A \mid \pi(a) > 0\}$. We denote by $\mathcal{D}(A)$ the set of probability distributions over A.

2.2 Stochastic models

A *discrete-time Markov chain (MC)* is a tuple $(S, \mathbf{P})$ where S is a finite set of states and $\mathbf{P} : S \to \mathcal{D}(S)$ is a stochastic transition matrix. For all $s, s' \in S$, we often write $\mathbf{P}(s, s')$ for $\mathbf{P}(s)(s')$. A *path* is an infinite sequence of states $\rho = s_0 s_1 s_2 \ldots$ such that $\mathbf{P}(s_i, s_{i+1}) > 0$ for all $i \geq 0$. Finite paths are defined similarly, and $\mathbf{P}$ is naturally extended to finite paths.

A *Markov decision process (MDP)* is a tuple $(S, \Sigma, \mathbf{P})$ where S is a finite set of states, Σ is a finite set of actions and $\mathbf{P} : S \times \Sigma \to \mathcal{D}(S)$ is a partial stochastic transition function. We often write $\mathbf{P}(s, \sigma, s')$ for $\mathbf{P}(s, \sigma)(s')$. For each state $s \in S$, we denote by $\Sigma_s \subseteq \Sigma$ the set of enabled actions in s, where an action $\sigma \in \Sigma$ is *enabled* in s if $(s, \sigma) \in \mathsf{Dom}(\mathbf{P})$. For all states $s \in S$, we require $\Sigma_s \neq \emptyset$, and we thus say that the MDP is Σ*-non-blocking*. For all actions $\sigma \in \Sigma$, we also introduce the notation S_σ for the set of states in which σ is enabled. For $s \in S$ and $\sigma \in \Sigma_s$, we denote by $\mathsf{succ}(s, \sigma) = \mathsf{Supp}(\mathbf{P}(s, \sigma))$ the set of possible successors of s for the enabled action σ.

2.3 Strategies

Let $(S, \Sigma, \mathbf{P})$ be an MDP. A *memoryless strategy* is a total function $\lambda : S \to \Sigma$ mapping each state s to an enabled action $\sigma \in \Sigma_s$. We denote by Λ the set of all memoryless strategies. A memoryless strategy λ induces an MC $(S, \mathbf{P}_\lambda)$ such that for all $s, s' \in S$, $\mathbf{P}_\lambda(s, s') = \mathbf{P}(s, \lambda(s), s')$.

2.4 Costs and value functions

Additionally to an MDP $(S, \Sigma, \mathbf{P})$, we consider a partial *cost function* $\mathbf{C} : S \times \Sigma \to \mathbb{R}$ with $\mathsf{Dom}(\mathbf{C}) = \mathsf{Dom}(\mathbf{P})$ that associates a cost with a state s and an enabled action σ in s. A memoryless strategy λ assigns a total cost function $\mathbf{C}_\lambda : S \to \mathbb{R}$ to the induced MC $(S, \mathbf{P}_\lambda)$, such that $\mathbf{C}_\lambda(s) = \mathbf{C}(s, \lambda(s))$. Given a path $\rho = s_0 s_1 s_2 \ldots$ in this MC, the *mean-payoff* of ρ is $\mathsf{MP}(\rho) = \limsup_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} \mathbf{C}_\lambda(s_i)$. Given a subset $G \subseteq S$ of *goal* states and a finite path ρ reaching a state of G, the *truncated sum up to* G of ρ is $\mathsf{TS}_G(\rho) = \sum_{i=0}^{n-1} \mathbf{C}_\lambda(s_i)$ where n is the first index such that $s_n \in G$.

Given an MDP with a cost function $\mathbf{C}$, and a memoryless strategy λ, we consider two classical value functions of λ defined as follows. For all states $s \in S$, the *expected mean-payoff* of λ is $\mathbb{E}_\lambda^{\mathrm{MP}}(s) = \lim_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} \mathbf{P}_\lambda^i \mathbf{C}_\lambda(s)$. Given a subset $G \subseteq S$, and assuming that λ reaches G from state s with probability 1, the *expected truncated sum up to* G of λ is $\mathbb{E}_\lambda^{\mathsf{TS}_G}(s) = \sum_\rho \mathbf{P}_\lambda(\rho)\mathsf{TS}_G(\rho)$ where the sum is over all finite paths $\rho = s_0 s_1 \ldots s_n$ such that $s_0 = s$, $s_n \in G$, and $s_0, \ldots, s_{n-1} \notin G$. Let λ^* be a memoryless strategy. Given a value function $\mathbb{E}_\lambda^{\cdot} \in \{\mathbb{E}_\lambda^{\mathrm{MP}}, \mathbb{E}_\lambda^{\mathsf{TS}_G}\}$, we say that λ^* is *optimal* if $\mathbb{E}_{\lambda^*}^{\cdot}(s) = \inf_{\lambda\in\Lambda} \mathbb{E}_\lambda^{\cdot}(s)$ for all $s \in S$, and $\mathbb{E}_{\lambda^*}^{\cdot}$ is called the *optimal* value function.[1] Note that we might have considered other classes of strategies but it is known that for these value functions, there always exists a memoryless strategy that minimizes the expected value of all states [4,41].

2.5 Studied problems

In this paper, we study algorithms for solving MDPs for two quantitative settings: the expected mean-payoff and the stochastic shortest path. Let $(S, \Sigma, \mathbf{P})$ be an MDP and $\mathbf{C} : S \times \Sigma \to \mathbb{R}$ be a cost function. (i) The *expected mean-payoff (EMP) problem* is to synthesize an optimal strategy for the expected mean-payoff value function. As explained above, such a memoryless optimal strategy always exists, and the problem is solvable in polynomial time via linear programming [23,41]. (ii) When $\mathbf{C}$ is restricted to *strictly positive* values in $\mathbb{R}_{>0}$, and a subset $G \subseteq S$ of goal states is given, the *stochastic shortest path (SSP) problem* is to synthesize an optimal strategy for the expected truncated sum value function, among the set of strategies that reach G with probability 1, provided such strategies exist. For all $s \in S$, we denote by Λ_s^P the set of *proper strategies* for s that are the strategies that lead from s to G with probability 1. Solving the SSP problem consists in two steps. The first step is to determine the set $S^P = \{s \in S \mid \Lambda_s^P \neq \emptyset\}$ of *proper* states, i.e. states having at least one proper strategy. The second step consists in synthesizing an optimal strategy λ^* such that $\mathbb{E}_{\lambda^*}^{\mathsf{TS}_G}(s) = \inf_{\lambda\in\Lambda_s^P} \mathbb{E}_\lambda^{\mathsf{TS}_G}(s)$ for all states of $s \in S^P$. It is known that memoryless optimal strategies exist for the SSP, and the problem can be solved in polynomial time through linear

[1] An alternative objective might be to maximize the value function, in which case λ^* is optimal if $\mathbb{E}_{\lambda^*}^{\cdot}(s) = \sup_{\lambda\in\Lambda} \mathbb{E}_\lambda^{\cdot}(s)$ for all $s \in S$.

programming [7,8]. Note that the existence of at least one proper strategy for each state is often stated as an assumption on the MDP. It that case, an algorithm for the SSP problem is limited to the second step.

In [46], the authors present a BDD based *symblicit* algorithm for the EMP problem, that is, an algorithm that efficiently combines symbolic and explicit representations. In this paper, we are interested in proposing antichain-based (instead of BDD-based) symblicit algorithms for both the EMP and SSP problems. Due to the use of antichains, our algorithms apply on a particular, but natural, class of MDPs, called *monotonic* MDPs. We first recall the definition of antichains and related notions. We then consider an example to intuitively illustrate the notion of monotonic MDP and we conclude with its formal definition.

2.6 Closed sets and antichains

Let S be a finite set equipped with a partial order $\preceq$ such that $(S, \preceq)$ is a *semilattice*, i.e. for all $s, s' \in S$, their greatest lower bound $s \sqcap s'$ always exists. A set $L \subseteq S$ is *closed* for $\preceq$ if for all $s \in L$ and all $s' \preceq s$, we have $s' \in L$. If $L_1, L_2 \subseteq S$ are two closed sets, then $L_1 \cap L_2$ and $L_1 \cup L_2$ are closed, but $L_1 \backslash L_2$ is not necessarily closed. The *closure* $\downarrow L$ of a set L is the set $\downarrow L = \{s' \in S \mid \exists s \in L \cdot s' \preceq s\}$. Note that $\downarrow L = L$ for all closed sets L. A set α is an *antichain* if all its elements are pairwise incomparable with respect to $\preceq$. For $L \subseteq S$, we denote by $\lceil L \rceil$ the set of its maximal elements, that is $\lceil L \rceil = \{s \in L \mid \forall s' \in L \cdot s \preceq s' \Rightarrow s = s'\}$. This set $\lceil L \rceil$ is an antichain. If L is closed, then $\downarrow \lceil L \rceil = L$, and $\lceil L \rceil$ is called the *canonical representation* of L. The interest of antichains is that they are *compact* representations of closed sets.

Example 1 To illustrate the notion of monotonic MDP in the SSP context, we consider the following example, inspired from [42], where a monkey tries to reach a hanging bunch of bananas. There are several items strewn in the room that the monkey can get and use, individually or simultaneously. There is a *box* on which it can climb to get closer to the bananas, a *stone* that can be thrown at the bananas, a *stick* to try to take the bananas down, and obviously the *bananas* that the monkey wants to eventually obtain. Initially, the monkey possesses no item. The monkey can make actions whose effects are to add and/or to remove items from its inventory. We add stochastic aspects to the problem. For example, using the *stick*, the monkey has probability $\frac{1}{5}$ to obtain the *bananas*, while combining the *box* and the *stick* increases this probability to $\frac{1}{2}$. Additionally, we associate a (positive) cost with each action, representing the time spent executing the action. For example, picking up the *stone* has a cost of 1, while getting the *box* costs 5. The objective of the monkey is then to minimize the expected cost for reaching the *bananas*.

This kind of specification naturally defines an MDP. The set S of states of the MDP is the set of all the possible combinations of items. Initially the monkey is in the state with no item. The available actions at each state $s \in S$ depend on the items of s. For example, when the monkey possesses the *box* and the *stick*, it can decide to try to reach the *bananas* by using one of these two items, or the combination of both of them. If it decides to use the *stick* only, it will reach the state $s \cup \{bananas\}$ with probability $\frac{1}{5}$ whereas it will stay at state s with probability $\frac{4}{5}$. This MDP is monotonic in the following sense. First, the set S is a closed set equipped with the partial order $\supseteq$. Second, the action of trying to reach the *bananas* with the *stick* is also available if the monkey possesses the *stick* together with other items. Moreover, if it succeeds (with probability $\frac{1}{5}$), it will reach a state with the *bananas* and all the items it already had at its disposal. In other words, for all states $s' \in S$ such that $s' \supseteq s = \{stick\}$,

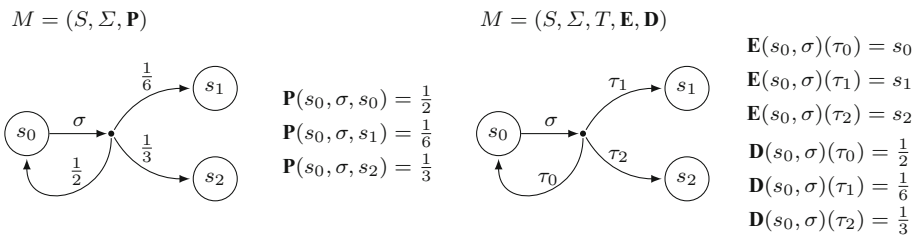

Fig. 1 Illustration of the new definition of MDPs for a state $s_0 \in S$ and an action $\sigma \in \Sigma_{s_0}$

we have that $\Sigma_s \subseteq \Sigma_{s'}$, and $t' \supseteq t = \{bananas, stick\}$ with t' the state reached from s' with probability $\frac{1}{5}$. Finally, note that the set of goal states $G = \{s \in S \mid bananas \in s\}$ is closed.

2.7 New definition of MDPs

To properly define the notion of monotonic MDPs, we need a slightly different, but equivalent, definition of MDPs which is based on a set T of stochastic actions. In this definition, an MDP M is a tuple $(S, \Sigma, T, \mathbf{E}, \mathbf{D})$ where S is a finite set of states, Σ and T are two finite sets of actions such that $\Sigma \cap T = \emptyset$, $\mathbf{E} : S \times \Sigma \to \mathcal{F}_{\text{tot}}(T, S)$ is a partial successor function, and $\mathbf{D} : S \times \Sigma \to \mathcal{D}(T)$ is a partial stochastic function such that $\mathsf{Dom}(\mathbf{E}) = \mathsf{Dom}(\mathbf{D})$. Figure 1 intuitively illustrates the relationship between the two definitions.

Let us explain this relationship more precisely. Let us consider an MDP as given in the new definition. We can then derive from $\mathbf{E}$ and $\mathbf{D}$ the partial transition function $\mathbf{P} : S \times \Sigma \to \mathcal{D}(S)$ such that for all $s, s' \in S$ and $\sigma \in \Sigma_s$,

$$\mathbf{P}(s, \sigma)(s') = \sum_{\substack{\tau \in T \\ \mathbf{E}(s,\sigma)(\tau) = s'}} \mathbf{D}(s, \sigma)(\tau).$$

Conversely, if we consider an MDP $(S, \Sigma, \mathbf{P})$ as in the first definition, then we can choose a set T of stochastic actions of size $|S|$ and adequate functions $\mathbf{E}$, $\mathbf{D}$ to get the second definition $(S, \Sigma, T, \mathbf{E}, \mathbf{D})$ for this MDP (see Fig. 1).

In this new definition of MDPs, for all $s \in S$ and all pair of actions $(\sigma, \tau) \in \Sigma \times T$, there is at most one $s' \in S$ such that $\mathbf{E}(s, \sigma)(\tau) = s'$. We thus say that M is *deterministic*. Moreover, since for all pairs $(s, \sigma) \in \mathsf{Dom}(\mathbf{E})$, $\mathbf{E}(s, \sigma)$ is a total function mapping each $\tau \in T$ to a state $s \in S$, we say that M is T*-complete*.

Notice that the notion of MC induced by a strategy can also be described in this new formalism as follows. Given an MDP $(S, \Sigma, T, \mathbf{E}, \mathbf{D})$ and a memoryless strategy λ, we have the induced MC $(S, T, \mathbf{E}_\lambda, \mathbf{D}_\lambda)$ such that $\mathbf{E}_\lambda : S \to \mathcal{D}(T)$ is the successor function with $\mathbf{E}_\lambda(s) = \mathbf{E}(s, \lambda(s))$, for all $s \in S$, and $\mathbf{D}_\lambda : S \to \mathcal{D}(T)$ is the stochastic function with $\mathbf{D}_\lambda(s) = \mathbf{D}(s, \lambda(s))$, for all $s \in S$.

Depending on the context, we will use both definitions $M = (S, \Sigma, T, \mathbf{E}, \mathbf{D})$ and $M = (S, \Sigma, \mathbf{P})$ for MDPs, assuming that $\mathbf{P}$ is always obtained from some set T and partial functions $\mathbf{E}$ and $\mathbf{D}$. We can now formally define the notion of monotonic MDP.

2.8 Monotonic MDPs

A *monotonic MDP* is an MDP $M_{\preceq} = (S, \Sigma, T, \mathbf{E}, \mathbf{D})$ such that:

1. The set S is equipped with a partial order $\preceq$ such that $(S, \preceq)$ is a semilattice.
2. The partial order $\preceq$ is *compatible* with $\mathbf{E}$, i.e. for all $s, s' \in S$, if $s \preceq s'$, then for all $\sigma \in \Sigma$, $\tau \in T$, for all $t' \in S$ such that $\mathbf{E}(s', \sigma)(\tau) = t'$, there exists $t \in S$ such that $\mathbf{E}(s, \sigma)(\tau) = t$ and $t \preceq t'$.

Note that since $(S, \preceq)$ is a semilattice, we have that S is closed for $\preceq$. With this definition, and in particular by compatibility of $\preceq$, we have the next proposition.

Proposition 1 *The following statements hold for a monotonic MDP $M_{\preceq}$:*

- *For all $s, s' \in S$, if $s \preceq s'$ then $\Sigma_{s'} \subseteq \Sigma_s$*
- *For all $\sigma \in \Sigma$, S_σ is closed.*

Remark 1 In this definition, by monotonic MDPs, we mean MDPs that are built on state spaces *already equipped with a natural partial order*. For instance, this is the case for the two classes of MDPs studied in Sects. 7.1 and 7.2. The same kind of approach has already been proposed in [26].

Note that all MDPs can be seen monotonic. Indeed, let $(S, \Sigma, T, \mathbf{E}, \mathbf{D})$ be a given MDP and let $\preceq$ be a partial order such that all states in S are pairwise incomparable with respect to $\preceq$. Let $t \notin S$ be an additional state such that (1) $t \preceq s$ for all $s \in S$, (2) $\mathbf{E}(s, \sigma)(\tau) \neq t$ for all $s \in S, \sigma \in \Sigma, \tau \in T$, and (3) $\mathbf{E}(t, \sigma)(\tau) = t$ for all $\sigma \in \Sigma, \tau \in T$. Then, we have that $(S \cup \{t\}, \preceq)$ is a semilattice and $\preceq$ is compatible with $\mathbf{E}$. However, such a partial order would not lead to efficient algorithms in the sense studied in this paper.

3 Strategy iteration algorithms

In this section, we present strategy iteration algorithms for synthesizing optimal strategies for the SSP and EMP problems. A *strategy iteration* algorithm [31] consists in generating a sequence of monotonically improving strategies (along with their associated value functions) until converging to an optimal one. Each iteration is composed of two phases: the *strategy evaluation phase* in which the value function of the current strategy is computed, and the *strategy improvement phase* in which the strategy is improved (if possible) at each state, by using the preceding computed value function. The algorithm stops after a finite number of iterations, as soon as no more improvement can be made, and returns the computed optimal strategy.

We now describe two strategy iteration algorithms, for the SSP and the EMP. We follow the presentation of those algorithms as given in [46].

3.1 Stochastic shortest path

We start with the strategy iteration algorithm for the SSP problem [7,31]. Let $M = (S, \Sigma, \mathbf{P})$ be an MDP, $\mathbf{C} : S \times \Sigma \to \mathbb{R}_{>0}$ be a strictly positive cost function, and $G \subseteq S$ be a set of goal states. Recall from the previous section that the solution to the SSP problem is to first compute the set of proper states which are the states having at least one proper strategy.

3.1.1 Computing proper states

An algorithm is proposed in [19] for computing in quadratic time the set $S^P = \{s \in S \mid \Lambda_s^P \neq \emptyset\}$ of proper states. To present it, given two subsets $X, Y \subseteq S$, we define the predicate $\mathsf{APre}(Y, X)$ such that for all $s \in S$,

$$s \models \mathsf{APre}(Y, X) \Leftrightarrow \exists \sigma \in \Sigma_s, (\mathsf{succ}(s, \sigma) \subseteq Y \wedge \mathsf{succ}(s, \sigma) \cap X \neq \emptyset).$$

Then, we can compute the set S^P of proper states by the following μ-calculus expression:

$$S^P = \nu Y \cdot \mu X \cdot (\mathsf{APre}(Y, X) \vee \mathsf{G}),$$

where we denote by G a predicate that holds exactly for the states in G. The algorithm works as follows. Initially, we have $Y_0 = S$. At the end of the first iteration, we have $Y_1 = S \backslash C_0$, where C_0 is the set of states that reach G with probability 0. At the end of the second iteration, we have $Y_2 = Y_1 \backslash C_1$, where C_1 is the set of states that cannot reach G without risking to enter C_0 (i.e. states in C_1 have a strictly positive probability of entering C_0). More generally, at the end of iteration $k > 0$, we have $Y_k = Y_{k-1} \backslash C_{k-1}$, where C_{k-1} is the set of states that cannot reach G without risking to enter $\bigcup_{i=0}^{k-2} C_i$. The correctness and complexity results are proved in [19].

Given an MDP $M = (S, \Sigma, \mathbf{P})$ with a cost function $\mathbf{C}$ and a set $G \subseteq S$, one can restrict M and $\mathbf{C}$ to the set S^P of proper states. We obtain a new MDP $M^P = (S^P, \Sigma, \mathbf{P}^P)$ with cost function $\mathbf{C}^P$ such that $\mathbf{P}^P$ and $\mathbf{C}^P$ are the restriction of $\mathbf{P}$ and $\mathbf{C}$ to S^P. Moreover, for all states $s \in S^P$, we let $\Sigma_s^P = \{\sigma \in \Sigma_s \mid \mathsf{succ}(s, \sigma) \subseteq S^P\}$ be the set of enabled actions in s. Note that by construction of S^P, we have $\Sigma_s^P \neq \emptyset$ for all $s \in S^P$, showing that M^P is Σ-non-blocking. To avoid a change of notation, in the sequel of this subsection, we make the assumption that each state of M is proper.

3.1.2 Strategy iteration algorithm

The strategy iteration algorithm for SSP, named SSP_StrategyIteration, is given in Algorithm 1.[2] This algorithm is applied under the typical assumption that all cycles in the underlying graph of M have strictly positive cost [7]. This assumption holds in our case by definition of the cost function $\mathbf{C}$. The algorithm starts with an arbitrary proper strategy λ_0, that can be easily computed with the algorithm of [19], and improves it until an optimal strategy is found. The expected truncated sum v_n of the current strategy λ_n is computed by solving the system of linear equations in line 3, and used to improve the strategy (if possible) at each state. Note that the strategy λ_n is improved at a state s to an action $\sigma \in \Sigma_s$ only if the new expected truncated sum is strictly smaller than the expected truncated sum of the action $\lambda_n(s)$, i.e. only if $\lambda_n(s) \notin \arg\min_{\sigma \in \Sigma_s} (\mathbf{C}(s, \sigma) + \sum_{s' \in S} \mathbf{P}(s, \sigma, s') \cdot v_n(s'))$. If no improvement is possible for any state, an optimal strategy is found and the algorithm terminates in line 7. Otherwise, it restarts by solving the new equation system, tries to improve the strategy using the new values computed, and so on.

3.2 Expected mean-payoff

We now consider the strategy iteration algorithm for the EMP problem [41,43] (see Algorithm 2[3]). More details can be found in [41]. The algorithm starts with an arbitrary strategy λ_0 (here any initial strategy is appropriate). By solving the equation system of line 3, we obtain the gain value g_n and bias value b_n of the current strategy λ_n. The gain corresponds to the expected mean-payoff, while the bias can be interpreted as the expected total difference between the cost and the expected mean-payoff. The computed gain value is then used

[2] If the expected truncated sum has to be maximized, the cost function is restricted to the strictly negative real numbers and arg min is replaced by arg max in line 4.

[3] If the expected mean-payoff has to be maximized, one has to replace arg min by arg max in lines 4 and 7.

Algorithm 1 SSP_StrategyIteration(MDP M, Strictly positive cost function $\mathbf{C}$, Goal states G)

1: $n := 0$, $\lambda_n :=$ InitialProperStrategy(M, G)
2: **repeat**
3: Obtain v_n by solving
$$\mathbf{C}_{\lambda_n} + (\mathbf{P}_{\lambda_n} - \mathbf{I})v_n = 0$$
4: $\widehat{\Sigma_s} := \underset{\sigma \in \Sigma_s}{\arg\min}(\mathbf{C}(s, \sigma) + \sum_{s' \in S} \mathbf{P}(s, \sigma, s') \cdot v_n(s')), \forall s \in S$
5: Choose λ_{n+1} such that $\lambda_{n+1}(s) \in \widehat{\Sigma_s}, \forall s \in S$, setting $\lambda_{n+1}(s) := \lambda_n(s)$ if possible.
6: $n := n + 1$
7: **until** $\lambda_n = \lambda_{n-1}$
8: return (λ_{n-1}, v_{n-1})

Algorithm 2 EMP_StrategyIteration(MDP M, Cost function $\mathbf{C}$)

1: $n := 0$, $\lambda_n :=$ InitialStrategy(M)
2: **repeat**
3: Obtain g_n and b_n by solving
$$\begin{cases} (\mathbf{P}_{\lambda_n} - \mathbf{I})g_n = 0 \\ \mathbf{C}_{\lambda_n} - g_n + (\mathbf{P}_{\lambda_n} - \mathbf{I})b_n = 0 \\ \mathbf{P}^*_{\lambda_n} b_n = 0 \end{cases}$$
4: $\widehat{\Sigma_s} := \underset{\sigma \in \Sigma_s}{\arg\min} \sum_{s' \in S} \mathbf{P}(s, \sigma, s') \cdot g_n(s'), \forall s \in S$
5: Choose λ_{n+1} such that $\lambda_{n+1}(s) \in \widehat{\Sigma_s}, \forall s \in S$, setting $\lambda_{n+1}(s) := \lambda_n(s)$ if possible.
6: **if** $\lambda_{n+1} = \lambda_n$ **then**
7: Choose λ_{n+1} such that $\lambda_{n+1}(s) \in \underset{\sigma \in \widehat{\Sigma_s}}{\arg\min}(\mathbf{C}(s, \sigma) + \sum_{s' \in S} \mathbf{P}(s, \sigma, s') \cdot b_n(s')), \forall s \in S,$
setting $\lambda_{n+1}(s) = \lambda_n(s)$ if possible.
8: $n := n + 1$
9: **until** $\lambda_n = \lambda_{n-1}$
10: return (λ_{n-1}, g_{n-1})

to locally improve the strategy (lines 4–5). If such an improvement is not possible for any state, the bias value is used to locally improve the strategy (lines 6–7). By improving the strategy with the bias value, only actions that also optimize the gain can be considered (see set $\widehat{\Sigma_s}$). Finally, the algorithm stops at line 10 as soon as none of those improvements can be made for any state, and returns the optimal strategy λ_{n-1} along with its associated expected mean-payoff.

4 Symblicit approach

Explicit-state representations of MDPs like sparse-matrices are often limited to the available memory. When treating MDPs with large state spaces, using explicit representations quickly becomes unfeasible. Moreover, the linear systems of large MDPs are in general hard to solve. Symbolic representations with *(multi-terminal) binary decision diagrams [(MT)BDDs]* are then an alternative solution. A *BDD* [14] is a data structure that permits to compactly represent boolean functions of n boolean variables, i.e. $\{0, 1\}^n \to \{0, 1\}$. An *MTBDD* [27] is a generalization of a BDD used to represent functions of n boolean variables, i.e. $\{0, 1\}^n \to V$, where V is a finite set. A symblicit algorithm for the EMP problem has been studied in [46]. It

combines symbolic techniques based on (MT)BDDs with explicit representations and often leads to a good trade-off between execution time and memory consumption.

In this section, we recall the symblicit algorithm proposed in [46] for solving the EMP problem on MDPs. However, our description is more general to suit also for the SSP problem. We first talk about bisimulation lumping, a technique used by this symblicit algorithm to reduce the state space of the models it works on.

4.1 Bisimulation lumping

The *bisimulation lumping* technique [15,34,36] applies to Markov chains. It consists in gathering certain states of an MC which behave equivalently according to the class of properties under consideration. For the expected truncated sum and the expected mean-payoff, the following definition of equivalence of two states can be used. Let $(S, \mathbf{P})$ be an MC and $\mathbf{C} : S \to \mathbb{R}$ be a cost function on S. Let $\sim$ be an equivalence relation on S and $S_\sim$ be the induced partition. We call any equivalence class of $\sim$ a *block* of $S_\sim$. We say that $\sim$ is a *bisimulation* if for all $s, t \in S$ such that $s \sim t$, we have $\mathbf{C}(s) = \mathbf{C}(t)$ and $\mathbf{P}(s, C) = \mathbf{P}(t, C)$ for all blocks $C \in S_\sim$, where $\mathbf{P}(s, C) = \sum_{s' \in C} \mathbf{P}(s, s')$.

Let $(S, \mathbf{P})$ be an MC with cost function $\mathbf{C}$, and $\sim$ be a bisimulation on S. The *bisimulation quotient* is the MC $(S_\sim, \mathbf{P}_\sim)$ such that $\mathbf{P}_\sim(C, C') = \mathbf{P}(s, C')$, where $s \in C$ and $C, C' \in S_\sim$. The cost function $\mathbf{C}_\sim : S_\sim \to \mathbb{R}$ is transferred to the quotient such that $\mathbf{C}_\sim(C) = \mathbf{C}(s)$, where $s \in C$ and $C \in S_\sim$. The quotient is thus a minimized model equivalent to the original one for our purpose, since it satisfies properties like expected truncated sum and expected mean-payoff as the original model [5]. Usually, we are interested in the unique *largest* bisimulation, denoted $\sim_L$, which leads to the smallest bisimulation quotient $(S_{\sim_L}, \mathbf{P}_{\sim_L})$.

Algorithm LUMP [20] (see Algorithm 3) describes how to compute the partition induced by the largest bisimulation. This algorithm is based on Paige and Tarjan's algorithm for computing bisimilarity of labeled transition systems [39].

For a given MC $M = (S, \mathbf{P})$ with cost function $\mathbf{C}$, Algorithm LUMP first computes the initial partition P such that for all $s, t \in S$, s and t belong to the same block of P iff $\mathbf{C}(s) = \mathbf{C}(t)$. The algorithm holds a list L of potential splitters of P, where a *splitter* of P is a set $C \subseteq S$ such that $\exists B \in P, \exists s, s' \in B$ such that $\mathbf{P}(s, C) \neq \mathbf{P}(s', C)$. Initially, this list L contains the blocks of the initial partition P. Then, while L is non empty, the algorithm takes a splitter C from L and refines each block of the partition according to C. Algorithm SPLITBLOCK splits a block B into non empty sub-blocks $B_1, \ldots, B_k$ according to the probability of reaching the splitter C, i.e. for all $s, s' \in B$, we have $s, s' \in B_i$ for some i iff $\mathbf{P}(s, C) = \mathbf{P}(s', C)$. The block B is then replaced in P by the computed sub-blocks

Algorithm 3 LUMP(MC M, Cost function $\mathbf{C}$)

```
1: P := InitialPartition(M, C)
2: L := P
3: while L ≠ ∅ do
4:    C := Pop(L)
5:    P_new := ∅
6:    for all B ∈ P do
7:       {B_1, ..., B_k} := SplitBlock(B, C)
8:       P_new := P_new ∪ {B_1, ..., B_k}
9:       B_l := some block in {B_1, ..., B_k}
10:      L := L ∪ {B_1, ..., B_k}\{B_l}
11:   P := P_new
12: return P
```

Algorithm 4 SYMBLICIT(MDP $\mathcal{M}$, [Strictly positive] cost function $\mathcal{C}$[, Goal states $\mathcal{G}$])

1: $n := 0$, $\lambda_n :=$ INITIALSTRATEGY($\mathcal{M}$[, $\mathcal{G}$])
2: **repeat**
3: $(\mathcal{M}_{\lambda_n}, \mathcal{C}_{\lambda_n}) :=$ INDUCEDMCANDCOST($\mathcal{M}, \mathcal{C}, \lambda_n$)
4: $(\mathcal{M}'_{\lambda_n}, \mathcal{C}'_{\lambda_n}) :=$ LUMP($\mathcal{M}_{\lambda_n}, \mathcal{C}_{\lambda_n}$)
5: $(M'_{\lambda_n}, \mathbf{C}'_{\lambda_n}) :=$ EXPLICIT($\mathcal{M}'_{\lambda_n}, \mathcal{C}'_{\lambda_n}$)
6: $x_n :=$ SOLVELINEARSYSTEM($M'_{\lambda_n}, \mathbf{C}'_{\lambda_n}$)
7: $\mathcal{X}_n :=$ SYMBOLIC(x_n)
8: $\lambda_{n+1} :=$ IMPROVESTRATEGY($\mathcal{M}, \lambda_n, \mathcal{X}_n$)
9: $n := n + 1$
10: **until** $\lambda_n = \lambda_{n-1}$
11: return $(\lambda_{n-1}, \mathcal{X}_{n-1})$

$B_1, \ldots, B_k$. Finally, we add to L the sub-blocks $B_1, \ldots, B_k$, but one which can be omitted since its power of splitting other blocks is maintained by the remaining sub-blocks [20]. In general, we prefer to omit the largest sub-block since it might be the most costly to process as potential splitter. The algorithm terminates when the list L is empty, which means that the partition is refined w.r.t. all potential splitters, i.e. P is the partition induced by the largest bisimulation $\sim_L$.

4.2 Symblicit algorithm

The algorithmic basis of the symblicit approach is the strategy iteration algorithm (see Algorithm 1 for the SSP and Algorithm 2 for the EMP). In addition, once a strategy λ_n is fixed for the MDP, Algorithm LUMP is applied on the induced MC in order to reduce its size and to produce its bisimulation quotient. The system of linear equations is then solved for the quotient, and the computed value functions are used to improve the strategy for each individual state of the MDP.

The symblicit algorithm is described in Algorithm SYMBLICIT (see Algorithm 4). Note that in line 1, the initial strategy λ_0 is selected arbitrarily for the EMP, while it has to be a proper strategy in case of SSP. It combines symbolic[4] and explicit representations of data manipulated by the underlying algorithm as follows. The MDP $\mathcal{M}$, the cost function $\mathcal{C}$, the strategies λ_n, the induced MCs $\mathcal{M}_{\lambda_n}$ with cost functions $\mathcal{C}_{\lambda_n}$, and the set $\mathcal{G}$ of goal states for the SSP, are symbolically represented. Therefore, the lumping procedure is applied on symbolic MCs and produces a symbolic representation of the bisimulation quotient $\mathcal{M}'_{\lambda_n}$ and associated cost function $\mathcal{C}'_{\lambda_n}$ (line 4). However, since solving linear systems is more efficient using an explicit representation of the transition matrix, the computed bisimulation quotient is converted to a sparse matrix representation (line 5). The quotient being in general much smaller than the original model, there are no memory issues by storing it explicitly. The linear system is thus solved on the explicit quotient. The computed value functions x_n (corresponding to v_n for the SPP, and g_n and b_n for the EMP) are then converted into symbolic representations $\mathcal{X}_n$, and transferred back to the original MDP (line 7). Finally, the update of the strategy is performed symbolically.

In [46], the intermediate symbolic representations use (MT)BDDs. In the sequel, we introduce a new data structure extended from antichains, called *pseudo-antichains*, and we show how it can be used [instead of (MT)BBDs] to solve the SSP and EMP problems for monotonic MDPs under well-chosen assumptions.

[4] We use calligraphic style for symbols denoting a symbolic representation.

5 Pseudo-antichains

In this section, we introduce the notion of pseudo-antichains.[5] We start by recalling properties of antichains [24].

Proposition 2 *Let $(S, \preceq)$ be a semilattice. Let $\alpha_1, \alpha_2 \subseteq S$ be two antichains and $s \in S$. Then:*

- $s \in \downarrow\alpha_1$ *iff* $\exists a \in \alpha_1 \cdot s \preceq a$,
- $\downarrow\alpha_1 \cup \downarrow\alpha_2 = \downarrow\lceil\alpha_1 \cup \alpha_2\rceil$,
- $\downarrow\alpha_1 \cap \downarrow\alpha_2 = \downarrow\lceil\alpha_1 \sqcap \alpha_2\rceil$, *where* $\alpha_1 \sqcap \alpha_2 \stackrel{\text{def}}{=} \{a_1 \sqcap a_2 \mid a_1 \in \alpha_1, a_2 \in \alpha_2\}$,
- $\downarrow\alpha_1 \subseteq \downarrow\alpha_2$ *iff* $\forall a_1 \in \alpha_1 \cdot \exists a_2 \in \alpha_2 \cdot a_1 \preceq a_2$.

For convenience, when α_1 and α_2 are antichains, we use notation $\alpha_1 \dot{\cup} \alpha_2$ (resp. $\alpha_1 \dot{\cap} \alpha_2$) for the antichain $\lceil\downarrow\alpha_1 \cup \downarrow\alpha_2\rceil$ (resp. $\lceil\downarrow\alpha_1 \cap \downarrow\alpha_2\rceil$).

Let $L_1, L_2 \subseteq S$ be two closed sets. Unlike the union or intersection, the difference $L_1 \backslash L_2$ is not necessarily a closed set. There is thus a need for a new structure that "represents" $L_1 \backslash L_2$ in a compact way, as antichains compactly represent closed sets. In this aim, in the next two sections, we begin by introducing the notion of a pseudo-element, and we then introduce the notion of a pseudo-antichain. We also describe some properties that can be used in algorithms using pseudo-antichains.

5.1 Pseudo-elements and pseudo-closures

A *pseudo-element* is a couple (x, α) where $x \in S$ and $\alpha \subseteq S$ is an antichain such that $x \notin \downarrow\alpha$. The *pseudo-closure* of a pseudo-element (x, α), denoted by $\updownarrow(x, \alpha)$, is the set $\updownarrow(x, \alpha) = \{s \in S \mid s \preceq x \text{ and } s \notin \downarrow\alpha\} = \downarrow\{x\} \backslash \downarrow\alpha$. Notice that $\updownarrow(x, \alpha)$ is non empty since $x \notin \downarrow\alpha$ by definition of a pseudo-element. The following example illustrates the notion of pseudo-closure of pseudo-elements.

Example 2 Let $\mathbb{N}^2_{\leq 3}$ be the set of pairs of natural numbers in $[0, 3]$ and let $\preceq$ be a partial order on $\mathbb{N}^2_{\leq 3}$ such that $(n_1, n_1') \preceq (n_2, n_2')$ iff $n_1 \leq n_2$ and $n_1' \leq n_2'$. Then, $(\mathbb{N}^2_{\leq 3}, \preceq)$ is a complete lattice with least upper bound $\sqcup$ such that $(n_1, n_1') \sqcup (n_2, n_2') = (\max(n_1, n_2), \max(n_1', n_2'))$, and greatest lower bound $\sqcap$ such that $(n_1, n_1') \sqcap (n_2, n_2') = (\min(n_1, n_2), \min(n_1', n_2'))$. With $x = (3, 2)$ and $\alpha = \{(2, 1), (0, 2)\}$, the pseudo-closure of the pseudo-element (x, α) is the set $\updownarrow(x, \alpha) = \{(3, 2), (3, 1), (3, 0), (2, 2), (1, 2)\} = \downarrow\{x\} \backslash \downarrow\alpha$ (see Fig. 2).

There may exist two pseudo-elements (x, α) and (y, β) such that $\updownarrow(x, \alpha) = \updownarrow(y, \beta)$. We say that the pseudo-element (x, α) is in *canonical form* if $\forall a \in \alpha \cdot a \preceq x$. The next proposition and its corollary show that the canonical form is unique. Notice that for all pseudo-elements (x, α), there exists a pseudo-element in canonical form (y, β) such that $\updownarrow(x, \alpha) = \updownarrow(y, \beta)$: it is equal to $(x, \{x\} \dot{\cap} \alpha)$. We say that such a couple (y, β) is the *canonical representation* of $\updownarrow(x, \alpha)$.

Proposition 3 *Let (x, α) and (y, β) be two pseudo-elements. Then $\updownarrow(x, \alpha) \subseteq \updownarrow(y, \beta)$ iff $x \preceq y$ and $\forall b \in \beta \cdot b \sqcap x \in \downarrow\alpha$.*

Proof We prove the two implications:

[5] A data structure closely related to our pseudo-antichains has been proposed in [2] in the particular context of probabilistic lossy channel systems. A comparison is given in Sect. 7.3.

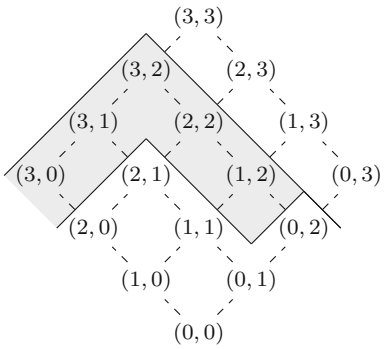

Fig. 2 Pseudo-closure of a pseudo-element over $\left(\mathbb{N}^2_{\leq 3}, \preceq\right)$

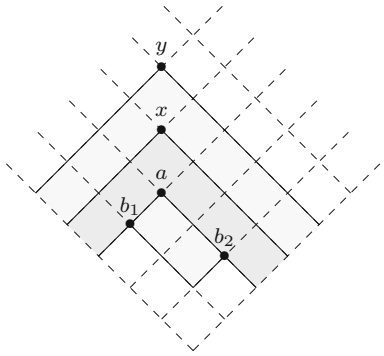

Fig. 3 Inclusion of pseudo-closures of pseudo-elements

- $\underline{\Rightarrow}$: Suppose that $\updownarrow(x, \alpha) \subseteq \updownarrow(y, \beta)$ and let us prove that $x \preceq y$ and $\forall b \in \beta \cdot b \sqcap x \in \downarrow\alpha$. As $x \in \updownarrow(x, \alpha) \subseteq \updownarrow(y, \beta)$, then $x \preceq y$. Consider $s = b \sqcap x$ for some $b \in \beta$. We have $s \notin \updownarrow(y, \beta)$ because $s \preceq b$ and thus $s \notin \updownarrow(x, \alpha)$. As $s \preceq x$, it follows that $s \in \downarrow\alpha$.
- $\underline{\Leftarrow}$: Suppose that $x \preceq y$ and $\forall b \in \beta \cdot b \sqcap x \in \downarrow\alpha$. Let us prove that $\forall s \in \updownarrow(x, \alpha)$, we have $s \in \updownarrow(y, \beta)$. As $s \preceq x$, and $x \preceq y$ by hypothesis, we have $s \preceq y$. Suppose that $s \in \downarrow\beta$, that is $s \preceq b$, for some $b \in \beta$. As $s \preceq x$, we have $s \preceq b \sqcap x$ and thus $s \in \downarrow\alpha$ by hypothesis. This is impossible since $s \in \updownarrow(x, \alpha)$. Therefore, $s \notin \downarrow\beta$, and thus $s \in \updownarrow(y, \beta)$. □

The following example illustrates Proposition 3.

Example 3 Let $(S, \preceq)$ be a semilattice and let $(x, \{a\})$ and $(y, \{b_1, b_2\})$, with $x, y, a, b_1, b_2 \in S$, be two pseudo-elements as depicted in Fig. 3. The pseudo-closure of $(x, \{a\})$ is depicted in dark gray, whereas the pseudo-closure of $(y, \{b_1, b_2\})$ is depicted in (light and dark) gray. We have $x \preceq y$, $b_1 \sqcap x = b_1 \in \downarrow\{a\}$ and $b_2 \sqcap x = b_2 \in \downarrow\{a\}$. Therefore $\updownarrow(x, \{a\}) \subseteq \updownarrow(y, \{b_1, b_2\})$.

The next corollary is a direct consequence of the previous proposition.

Corollary 1 *Let* (x, α) *and* (y, β) *be two pseudo-elements in canonical form. Then* $\updownarrow(x, \alpha) = \updownarrow(y, \beta)$ *iff* $x = y$ *and* $\alpha = \beta$.

Proof We only prove $\updownarrow(x, \alpha) = \updownarrow(y, \beta) \Rightarrow x = y$ and $\alpha = \beta$, the other implication being trivial.

Since $\updownarrow(x,\alpha) = \updownarrow(y,\beta)$, we have $\updownarrow(x,\alpha) \subseteq \updownarrow(y,\beta)$ and $\updownarrow(y,\beta) \subseteq \updownarrow(x,\alpha)$. By Proposition 3, from $\updownarrow(x,\alpha) \subseteq \updownarrow(y,\beta)$, we know that $x \preceq y$ and $\forall b \in \beta \cdot b \sqcap x \in \downarrow\alpha$, and from $\updownarrow(y,\beta) \subseteq \updownarrow(x,\alpha)$, we know that $y \preceq x$ and $\forall a \in \alpha \cdot a \sqcap y \in \downarrow\beta$. As $x \preceq y$ and $y \preceq x$, we thus have $x = y$.

Since $x = y$, by definition of canonical form of pseudo-elements, we have $\forall b \in \beta \cdot b \sqcap x = b \sqcap y = b \in \downarrow\alpha$ and $\forall a \in \alpha \cdot a \sqcap y = a \in \downarrow\beta$. It follows by Proposition 2 that $\downarrow\alpha \subseteq \downarrow\beta$ and $\downarrow\beta \subseteq \downarrow\alpha$, i.e. $\downarrow\alpha = \downarrow\beta$. Since antichains are canonical representations of closed sets, we finally get $\alpha = \beta$, which terminates the proof. □

5.2 Pseudo-antichains

We are now ready to introduce the new structure of pseudo-antichains. A *pseudo-antichain* A is a finite set of pseudo-elements, that is $A = \{(x_i,\alpha_i) \mid i \in I\}$ with I finite. The *pseudo-closure* $\updownarrow A$ of A is defined as the set $\updownarrow A = \bigcup_{i\in I} \updownarrow(x_i,\alpha_i)$. Let $(x_i,\alpha_i), (x_j,\alpha_j) \in A$. We have the two following observations:

1. If $x_i = x_j$, then (x_i,α_i) and (x_j,α_j) can be replaced in A by the pseudo-element $(x_i, \alpha_i \mathbin{\dot\cap} \alpha_j)$.
2. If $\updownarrow(x_i,\alpha_i) \subseteq \updownarrow(x_j,\alpha_j)$, then (x_i,α_i) can be removed from A.

From these observations, we say that a pseudo-antichain $A = \{(x_i,\alpha_i) \mid i \in I\}$ is *simplified* if $\forall i \cdot (x_i,\alpha_i)$ is in canonical form, and $\forall i \neq j \cdot x_i \neq x_j$ and $\updownarrow(x_i,\alpha_i) \not\subseteq \updownarrow(x_j,\alpha_j)$. Notice that two distinct pseudo-antichains A and B can have the same pseudo-closure $\updownarrow A = \updownarrow B$ even if they are simplified. We thus say that A is a *PA-representation*[6] of $\updownarrow A$ (without saying that it is a canonical representation), and that $\updownarrow A$ is *PA-represented* by A. For efficiency purposes, our algorithms always work on simplified pseudo-antichains.

Any antichain α can be seen as the pseudo-antichain $A = \{(x,\emptyset) \mid x \in \alpha\}$. Furthermore, notice that *any* set X can be represented by the pseudo-antichain $A = \{(x,\alpha_x) \mid x \in X\}$, with $\alpha_x = \lceil\{s \in S \mid s \preceq x \text{ and } s \neq x\}\rceil$. Indeed $\updownarrow(x,\alpha_x) = \{x\}$ for all x, and thus $X = \updownarrow A$.

The interest of pseudo-antichains is that given two antichains α and β, the difference $\downarrow\alpha \setminus \downarrow\beta$ is PA-represented by the pseudo-antichain $\{(x,\beta) \mid x \in \alpha, x \notin \downarrow\beta\}$.

Lemma 1 *Let $\alpha, \beta \subseteq S$ be two antichains. Then $\downarrow\alpha \setminus \downarrow\beta = \updownarrow\{(x,\beta) \mid x \in \alpha, x \notin \downarrow\beta\}$.*

The next proposition indicates how to compute pseudo-closures of pseudo-elements w.r.t. the union, intersection and difference operations. This method can be extended for computing the union, intersection and difference of pseudo-closures of pseudo-antichains, by using the classical properties from set theory like $X \setminus (Y \cup Z) = X \setminus Y \cap X \setminus Z$. From the algorithmic point of view, it is important to note that the computations only manipulate (pseudo-)antichains instead of their (pseudo-)closure.

Proposition 4 *Let $(x,\alpha), (y,\beta)$ be two pseudo-elements. Then:*

- $\updownarrow(x,\alpha) \cup \updownarrow(y,\beta) = \updownarrow\{(x,\alpha), (y,\beta)\}$,
- $\updownarrow(x,\alpha) \cap \updownarrow(y,\beta) = \updownarrow\{(x \sqcap y, \alpha \mathbin{\dot\cup} \beta)\}$,
- $\updownarrow(x,\alpha) \setminus \updownarrow(y,\beta) = \updownarrow(\{(x, \{y\} \mathbin{\dot\cup} \alpha)\} \cup \{(x \sqcap b, \alpha) \mid b \in \beta\})$.

Notice that there is an abuse of notation in the previous proposition. Indeed, the definition of a pseudo-element (x,α) requires that $x \notin \downarrow\alpha$, whereas this condition could not be satisfied by couples like $(x \sqcap y, \alpha \mathbin{\dot\cup} \beta)$, $(x, \{y\} \mathbin{\dot\cup} \alpha)$ and $(x \sqcap b, \alpha)$ in the previous definition.[7] When

[6] "PA-representation" means pseudo-antichain-based representation.

[7] This can be easily tested by Proposition 2.

this happens, such a couple should not be added to the related pseudo-antichain. For instance, $\updownarrow(x,\alpha) \cap \updownarrow(y,\beta)$ is either equal to $\updownarrow\{(x \sqcap y, \alpha \dot\cup \beta)\}$ or to $\updownarrow\{\}$. Notice also that the pseudo-antichains computed in the previous proposition are not necessarily simplified. However, our algorithms implementing those operations always simplify the computed pseudo-antichains for the sake of efficiency.

Proof (of Proposition 4) We prove the three statements:

- $\updownarrow(x,\alpha) \cup \updownarrow(y,\beta) = \updownarrow\{(x,\alpha),(y,\beta)\}$: This result comes directly from the definition of pseudo-closure of pseudo-antichains.
- $\updownarrow(x,\alpha) \cap \updownarrow(y,\beta) = \updownarrow\{(x \sqcap y, \alpha \dot\cup \beta)\}$:

$$\begin{aligned} s \in \updownarrow(x,\alpha) \cap \updownarrow(y,\beta) &\Leftrightarrow s \preceq x, s \notin {\downarrow}\alpha \text{ and } s \preceq y, s \notin {\downarrow}\beta \\ &\Leftrightarrow s \preceq x \sqcap y \text{ and } s \notin {\downarrow}\alpha \cup {\downarrow}\beta = {\downarrow}(\alpha \dot\cup \beta) \text{ (by Proposition 2)} \\ &\Leftrightarrow s \in \updownarrow\{(x \sqcap y, \alpha \dot\cup \beta)\} \end{aligned}$$

- $\updownarrow(x,\alpha) \setminus \updownarrow(y,\beta) = \updownarrow(\{(x,\{y\} \dot\cup \alpha)\} \cup \{(x \sqcap b, \alpha) \mid b \in \beta\})$: We prove the two inclusions:
  1. $\subseteq$: Let $s \in \updownarrow(x,\alpha) \setminus \updownarrow(y,\beta)$, i.e. $s \in \updownarrow(x,\alpha)$ and $s \notin \updownarrow(y,\beta)$. Then, $s \preceq x$, $s \notin {\downarrow}\alpha$ and ($s \not\preceq y$ or $s \in {\downarrow}\beta$). Thus, if $s \not\preceq y$, then $s \in \updownarrow(x, \{y\} \dot\cup \alpha)$. Otherwise, $s \in {\downarrow}\beta$, i.e. $\exists b \in \beta$ such that $s \preceq b$. It follows that $s \in \updownarrow(x \sqcap b, \alpha)$.
  2. $\supseteq$: Let $s \in \updownarrow(\{(x,\{y\} \dot\cup \alpha)\} \cup \{(x \sqcap b, \alpha) \mid b \in \beta\})$. Suppose first that $s \in \updownarrow(x,\{y\} \dot\cup \alpha)$. Then $s \preceq x$, $s \not\preceq y$ and $s \notin {\downarrow}\alpha$. We thus have $s \in \updownarrow(x,\alpha)$ and $s \notin \updownarrow(y,\beta)$. Suppose now that $\exists b \in \beta \cdot s \in \updownarrow(x \sqcap b, \alpha)$. We have $s \preceq x$, $s \preceq b$ and $s \notin {\downarrow}\alpha$. It follows that $s \in \updownarrow(x,\alpha)$ and $s \in {\downarrow}\beta$, thus $s \notin \updownarrow(y,\beta)$. □

The following example illustrates the second and third statements of Proposition 4.

Example 4 Let $(S, \preceq)$ be a lower semilattice and let $(x,\{a\})$ and $(y,\{b\})$, with $x, y, a, b \in S$, be two pseudo-elements as depicted in Fig. 4. We have $\updownarrow(x,\{a\}) \cap \updownarrow(y,\{b\}) = \updownarrow(x \sqcap y, \{a,b\})$. We also have $\updownarrow(x,\{a\}) \setminus \updownarrow(y,\{b\}) = \updownarrow\{(x,\{y\} \dot\cup \{a\}), (x \sqcap b, \{a\})\} = \updownarrow\{(x,\{y\}), (b,\{a\})\}$. Note that $(x,\{y\})$ and $(b,\{a\})$ are not in canonical form. The canonical representation of $\updownarrow(x,\{y\})$ (resp. $\updownarrow(b,\{a\})$) is given by $(x,\{x \sqcap y\})$ (resp. $(b,\{b \sqcap a\})$).

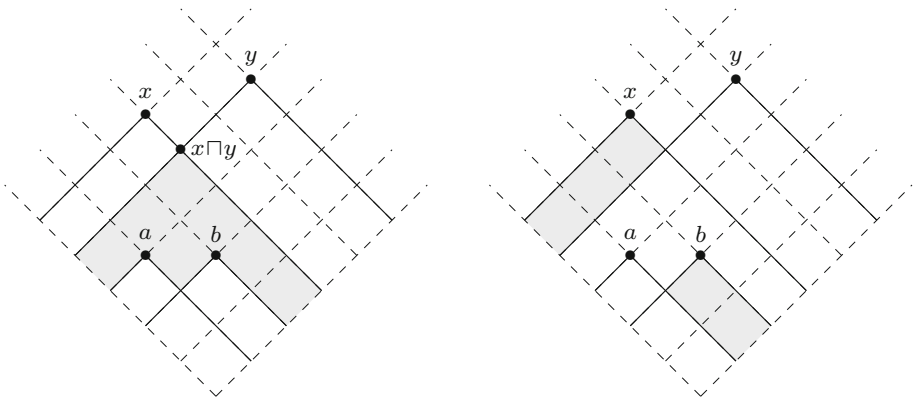

Fig. 4 Intersection (*left*) and difference (*right*) of two pseudo-closures of pseudo-elements

6 Pseudo-antichain-based algorithms

In this section, we propose a pseudo-antichain-based version of the symblicit algorithm described in Sect. 4 for solving the SSP and EMP problems for monotonic MDPs. In our approach, equivalence relations and induced partitions are symbolically represented so that each block is PA-represented. The efficiency of this approach is thus directly linked to the number of blocks to represent, which explains why our algorithm always works with the coarsest equivalence relations. It is also linked to the size of the pseudo-antichains representing the blocks of the partitions.

6.1 Operator $\mathsf{Pre}_{\sigma,\tau}$

We first present an operator denoted $\mathsf{Pre}_{\sigma,\tau}$ that is useful for our algorithms. Let $M_{\preceq} = (S, \Sigma, T, \mathbf{E}, \mathbf{D})$ be a monotonic MDP equipped with a cost function $\mathbf{C} : S \times \Sigma \to \mathbb{R}$. Given $L \subseteq S$, $\sigma \in \Sigma$ and $\tau \in T$, we denote by $\mathsf{Pre}_{\sigma,\tau}(L)$ the set of states that reach L by σ, τ in $M_{\preceq}$, that is

$$\mathsf{Pre}_{\sigma,\tau}(L) = \{s \in S \mid \mathbf{E}(s, \sigma)(\tau) \in L\}.$$

The elements of $\mathsf{Pre}_{\sigma,\tau}(L)$ are called *predecessors* of L for σ, τ in $M_{\preceq}$. The following lemma is a direct consequence of the compatibility of $\preceq$.

Lemma 2 *For all closed sets $L \subseteq S$, and all actions $\sigma \in \Sigma, \tau \in T$, $\mathsf{Pre}_{\sigma,\tau}(L)$ is closed.*

The next lemma indicates the behavior of the $\mathsf{Pre}_{\sigma,\tau}$ operator under boolean operations. The second and last properties follow from the fact that $M_{\preceq}$ is deterministic.

Lemma 3 *Let $L_1, L_2 \subseteq S$, $\sigma \in \Sigma$ and $\tau \in T$. Then,*

- $\mathsf{Pre}_{\sigma,\tau}(L_1 \cup L_2) = \mathsf{Pre}_{\sigma,\tau}(L_1) \cup \mathsf{Pre}_{\sigma,\tau}(L_2)$,
- $\mathsf{Pre}_{\sigma,\tau}(L_1 \cap L_2) = \mathsf{Pre}_{\sigma,\tau}(L_1) \cap \mathsf{Pre}_{\sigma,\tau}(L_2)$,
- $\mathsf{Pre}_{\sigma,\tau}(L_1 \setminus L_2) = \mathsf{Pre}_{\sigma,\tau}(L_1) \setminus \mathsf{Pre}_{\sigma,\tau}(L_2)$.

Proof The first property is immediate. We only prove the second property, since the arguments are similar for the last one. Let $s \in \mathsf{Pre}_{\sigma,\tau}(L_1 \cap L_2)$, i.e. $\exists s' \in L_1 \cap L_2$ such that $\mathbf{E}(s, \sigma)(\tau) = s'$. We thus have $s \in \mathsf{Pre}_{\sigma,\tau}(L_1)$ and $s \in \mathsf{Pre}_{\sigma,\tau}(L_2)$. Conversely let $s \in \mathsf{Pre}_{\sigma,\tau}(L_1) \cap \mathsf{Pre}_{\sigma,\tau}(L_2)$, i.e. $\exists s_1 \in L_1$ such that $\mathbf{E}(s, \sigma)(\tau) = s_1$, and $\exists s_2 \in L_2$ such that $\mathbf{E}(s, \sigma)(\tau) = s_2$. As $M_{\preceq}$ is deterministic, we have $s_1 = s_2$ and thus $s \in \mathsf{Pre}_{\sigma,\tau}(L_1 \cap L_2)$. □

The next proposition indicates how to compute pseudo-antichains w.r.t. the $\mathsf{Pre}_{\sigma,\tau}$ operator.

Proposition 5 *Let (x, α) be a pseudo-element with $x \in S$ and $\alpha \subseteq S$. Let $A = \{(x_i, \alpha_i) \mid i \in I\}$ be a pseudo-antichain with $x_i \in S$ and $\alpha_i \subseteq S$ for all $i \in I$. Then, for all $\sigma \in \Sigma$ and $\tau \in T$,*

- $\mathsf{Pre}_{\sigma,\tau}(\updownarrow(x, \alpha)) = \bigcup_{x' \in \lceil \mathsf{Pre}_{\sigma,\tau}(\downarrow\{x\}) \rceil} \updownarrow(x', \lceil \mathsf{Pre}_{\sigma,\tau}(\downarrow\alpha) \rceil)$,
- $\mathsf{Pre}_{\sigma,\tau}(\updownarrow A) = \bigcup_{i \in I} \mathsf{Pre}_{\sigma,\tau}(\updownarrow(x_i, \alpha_i))$.

Proof For the first statement, we have $\mathsf{Pre}_{\sigma,\tau}(\updownarrow(x, \alpha)) = \mathsf{Pre}_{\sigma,\tau}(\downarrow\{x\} \setminus \downarrow\alpha) = \mathsf{Pre}_{\sigma,\tau}(\downarrow\{x\}) \setminus \mathsf{Pre}_{\sigma,\tau}(\downarrow\alpha)$ by definition of the pseudo-closure and by Lemma 3. The sets $\mathsf{Pre}_{\sigma,\tau}(\downarrow\{x\})$ and $\mathsf{Pre}_{\sigma,\tau}(\downarrow\alpha)$ are closed by Lemma 2 and thus respectively represented by the antichains $\lceil \mathsf{Pre}_{\sigma,\tau}(\downarrow\{x\}) \rceil$ and $\lceil \mathsf{Pre}_{\sigma,\tau}(\downarrow\alpha) \rceil$. By Lemma 1 we get the first statement.

The second statement is a direct consequence of Lemma 3. □

From Proposition 5, we can efficiently compute pseudo-antichains w.r.t. the $\mathsf{Pre}_{\sigma,\tau}$ operator if we have an efficient algorithm to compute antichains w.r.t. $\mathsf{Pre}_{\sigma,\tau}$ (see the first statement). We make the following assumption that we can compute the predecessors of a closed set by only considering the antichain of its maximal elements. Together with Proposition 5, it implies that the computation of $\mathsf{Pre}_{\sigma,\tau}(\updownarrow A)$, for all pseudo-antichains A, does not need to treat the whole pseudo-closure $\updownarrow A$.

Assumption 1 There exists an algorithm taking any state $x \in S$ as input and returning $\lceil \mathsf{Pre}_{\sigma,\tau}(\downarrow\{x\}) \rceil$ as output.

Remark 2 Assumption 1 is a realistic and natural assumption when considering partially ordered state spaces. For instance, it holds for the two classes of MDPs considered in Sects. 7.1 and 7.2 for which the given algorithm is straightforward. Assumptions in the same flavor are made in ([26], see Definition 3.2).

6.2 Symbolic representations

Before giving a pseudo-antichain-based algorithm for the symblicit approach of Sect. 4 (see Algorithm 4), we detail in this section the kind of symbolic representations based on pseudo-antichains that we are going to use. Recall from Sect. 5 that PA-representations are not unique. For efficiency reasons, it will be necessary to work with PA-representations as *compact* as possible, as suggested in the sequel.

6.2.1 Representation of the stochastic models

We begin with symbolic representations for the monotonic MDP $M_{\preceq} = (S, \Sigma, T, \mathbf{E}, \mathbf{D})$ and for the MC $M_{\preceq,\lambda} = (S, T, \mathbf{E}_\lambda, \mathbf{D}_\lambda)$ induced by a strategy λ. For algorithmic purposes, in addition to Assumption 1, we make the following assumption[8] on $M_{\preceq}$.

Assumption 2 There exists an algorithm taking as input any state $s \in S$ and actions $\sigma \in \Sigma_s$, $\tau \in T$, and returning as output $\mathbf{E}(s,\sigma)(\tau)$ and $\mathbf{D}(s,\sigma)(\tau)$.

By definition of $M_{\preceq}$, the set S of states is closed for $\preceq$ and can thus be canonically represented by the antichain $\lceil S \rceil$, and thus represented by the pseudo-antichain $\{(x, \emptyset) \mid x \in \lceil S \rceil\}$. In this way, it follows by Assumption 2 that we have a PA-representation of $M_{\preceq}$, in the sense that S is PA-represented and we can compute $\mathbf{E}(s,\sigma)(\tau)$ and $\mathbf{D}(s,\sigma)(\tau)$ on demand.

Let $\lambda : S \to \Sigma$ be a strategy on $M_{\preceq}$ and $M_{\preceq,\lambda}$ be the induced MC with cost function $\mathbf{C}_\lambda$. We denote by $\sim_\lambda$ the equivalence relation on S such that $s \sim_\lambda s'$ iff $\lambda(s) = \lambda(s')$. We denote by $S_{\sim_\lambda}$ the induced partition of S. Given a block $B \in S_{\sim_\lambda}$, we denote by $\lambda(B)$ the unique action $\lambda(s)$, for all $s \in B$. As any set can be represented by a pseudo-antichain, each block of $S_{\sim_\lambda}$ is PA-represented. Therefore by Assumption 2, we have a PA-representation of $M_{\preceq,\lambda}$.

6.2.2 Representation of a subset of goal states

Recall that a subset $G \subseteq S$ of goal states is required for the SSP problem. Our algorithm will manipulate G when computing the set of proper states. If G is *closed*, then we have a compact representation of G as the one proposed above for S. Otherwise, we take for G any PA-representation. Notice that G is closed for the two classes of monotonic MDPs studied in Sects. 7.1 and 7.2.

[8] Remark 2 also holds for Assumption 2.

6.2.3 Representation for $\mathbf{D}$ and $\mathbf{C}$

For the needs of our algorithm, we introduce symbolic representations for $\mathbf{D}_\lambda$ and $\mathbf{C}_\lambda$. Similarly to $\sim_\lambda$, let $\sim_{\mathbf{D},\lambda}$ be the equivalence relation on S such that $s \sim_{\mathbf{D},\lambda} s'$ iff $\mathbf{D}_\lambda(s) = \mathbf{D}_\lambda(s')$. We denote by $S_{\sim_{\mathbf{D},\lambda}}$ the induced partition of S. Given a block $B \in S_{\sim_{\mathbf{D},\lambda}}$, we denote by $\mathbf{D}_\lambda(B)$ the unique probability distribution $\mathbf{D}_\lambda(s)$, for all $s \in B$. We use similar notations for the equivalence relation $\sim_{\mathbf{C},\lambda}$ on S such that $s \sim_{\mathbf{C},\lambda} s'$ iff $\mathbf{C}_\lambda(s) = \mathbf{C}_\lambda(s')$. As any set can be represented by a pseudo-antichain, each block of $S_{\sim_{\mathbf{D},\lambda}}$ and $S_{\sim_{\mathbf{C},\lambda}}$ is PA-represented.

We will also need to use the next two equivalence relations. For each $\sigma \in \Sigma$, we introduce the equivalence relation $\sim_{\mathbf{D},\sigma}$ on S such that $s \sim_{\mathbf{D},\sigma} s'$ iff $\mathbf{D}(s,\sigma) = \mathbf{D}(s',\sigma)$. Similarly, we introduce relation $\sim_{\mathbf{C},\sigma}$ such that $s \sim_{\mathbf{C},\sigma} s'$ iff $\mathbf{C}(s,\sigma) = \mathbf{C}(s',\sigma)$. Recall that $\mathbf{D}$ and $\mathbf{C}$ are partial functions, there may thus exist one block in their corresponding relation gathering all states s such that $\sigma \notin \Sigma_s$. Each block of the induced partitions $S_{\sim_{\mathbf{D},\sigma}}$ and $S_{\sim_{\mathbf{C},\sigma}}$ is PA-represented.

For the two classes of MDPs studied in Sects. 7.1 and 7.2, both functions $\mathbf{D}$ and $\mathbf{C}$ are independent of S. It follows that the previous equivalence relations have only one or two blocks, leading to compact symbolic representations of these relations.

Now that the operator $\mathsf{Pre}_{\sigma,\tau}$ and the symbolic representations have been introduced, we come back to the different steps of the symblicit approach of Sect. 4 (see Algorithm 4) and show how to derive a pseudo-antichain-based algorithm. We will use Propositions 2, 4 and 5, and Assumptions 1 and 2, for which we know that boolean and $\mathsf{Pre}_{\sigma,\tau}$ operations can be performed efficiently on pseudo-closures of pseudo-antichains, by limiting the computations to the related pseudo-antichains. Whenever possible, we will work with partitions with few blocks whose PA-representation is compact. This aim will be reached for the two classes of monotonic MDPs studied in Sects. 7.1 and 7.2.

6.3 Initial strategy

Algorithm 4 needs an initial strategy λ_0 (line 1). This strategy can be selected arbitrarily among the set of strategies for the EMP, while it has to be a proper strategy for the SSP. We detail how to choose the initial strategy in these two quantitative settings.

6.3.1 Expected mean-payoff

For the EMP, we propose an arbitrary initial strategy λ_0 with a compact PA-representation for the induced MC $M_{\preceq,\lambda_0}$. We know that S is PA-represented by $\{(x,\emptyset) \mid x \in \lceil S \rceil\}$, and that for all $s, s' \in S$ such that $s \preceq s'$, we have $\Sigma_{s'} \subseteq \Sigma_s$ (Proposition 1). This means that for $x \in \lceil S \rceil$ and $\sigma \in \Sigma_x$ we could choose $\lambda_0(s) = \sigma$ for all $s \in \downarrow\{x\}$. However we must be careful with states that belong to $\downarrow\{x\} \cap \downarrow\{x'\}$ with $x, x' \in \lceil S \rceil$, $x \neq x'$. Therefore, let us impose an arbitrary ordering on $\lceil S \rceil = \{x_1, \ldots, x_n\}$, i.e. $x_1 < x_2 < \cdots < x_n$. We then define λ_0 arbitrarily on $\lceil S \rceil$ such that $\lambda_0(x_i) = \sigma_i$ for some $\sigma_i \in \Sigma_{x_i}$, and we extend it to all $s \in S$ by $\lambda_0(s) = \lambda_0(x)$ with $x = \min_i\{x_i \mid s \preceq x_i\}$. This makes sense in view of the previous remarks. Notice that given $\sigma \in \Sigma$, the block B of the partition $S_{\sim_{\lambda_0}}$ such that $\lambda_0(B) = \sigma$ is PA-represented by $\bigcup_i \{(x_i, \alpha_i) \mid \alpha_i = \{x_1, \ldots, x_{i-1}\}, \lambda_0(x_i) = \sigma\}$.

6.3.2 Proper states

Before explaining how to compute an initial proper strategy λ_0 for the SSP, we need to propose a pseudo-antichain-based version of the algorithm of [19] for computing the set S^P

of proper states. Recall from Sect. 3.1 that this algorithm is required for solving the SSP problem.

Let $M_{\preceq}$ be a monotonic MDP and G be a set of goal states. Recall that S^P is computed as $S^P = \nu Y \cdot \mu X \cdot (\mathsf{APre}(Y, X) \vee \mathsf{G})$, such that for all states s,

$$s \models \mathsf{APre}(Y, X) \Leftrightarrow \exists \sigma \in \Sigma_s, (\mathsf{succ}(s, \sigma) \subseteq Y \wedge \mathsf{succ}(s, \sigma) \cap X \neq \emptyset).$$

Our purpose is to define the set of states satisfying $\mathsf{APre}(Y, X)$ thanks to the operator $\mathsf{Pre}_{\sigma,\tau}$. The difficulty is to limit the computations to strictly positive probabilities as required by the operator succ. In this aim, given the equivalence relation $\sim_{\mathbf{D},\sigma}$ defined in Sect. 6.2, for each $\sigma \in \Sigma$ and $\tau \in T$, we define $\mathcal{D}^{>0}_{\sigma,\tau}$ being the set of blocks $\{D \in S_{\sim_{\mathbf{D},\sigma}} \mid \mathbf{D}(s, \sigma)(\tau) > 0 \text{ with } s \in D\}$. For each $D \in \mathcal{D}^{>0}_{\sigma,\tau}$, notice that $\sigma \in \Sigma_s$ for all $s \in D$ (since $\mathbf{D}(s, \sigma)$ is defined). Given two sets $X, Y \subseteq S$, the set of states satisfying $\mathsf{APre}(Y, X)$ is equal to:

$$R(Y, X) = \bigcup_{\sigma \in \Sigma} \bigcup_{D \in S_{\sim_{\mathbf{D},\sigma}}} \left(\bigcap_{\substack{\tau \in T \\ D \in \mathcal{D}^{>0}_{\sigma,\tau}}} \left(\mathsf{Pre}_{\sigma,\tau}(Y) \cap D\right) \cap \bigcup_{\substack{\tau \in T \\ D \in \mathcal{D}^{>0}_{\sigma,\tau}}} \left(\mathsf{Pre}_{\sigma,\tau}(X) \cap D\right) \right)$$

Lemma 4 *For all $X, Y \subseteq S$ and $s \in S$, $s \models \mathsf{APre}(Y, X) \Leftrightarrow s \in R(Y, X)$.*

Proof Let $s \models \mathsf{APre}(Y, X)$. Then there exists $\sigma \in \Sigma_s$ such that $\mathsf{succ}(s, \sigma) \subseteq Y$ and $\mathsf{succ}(s, \sigma) \cap X \neq \emptyset$. Let $D \in S_{\sim_{\mathbf{D},\sigma}}$ be such that $s \in D$. Let us prove that $s \in \bigcap_{\tau \in T, D \in \mathcal{D}^{>0}_{\sigma,\tau}} \mathsf{Pre}_{\sigma,\tau}(Y)$ and $s \in \bigcup_{\tau \in T, D \in \mathcal{D}^{>0}_{\sigma,\tau}} \mathsf{Pre}_{\sigma,\tau}(X)$. It will follow that $s \in R(Y, X)$. As $\mathsf{succ}(s, \sigma) \cap X \neq \emptyset$, there exists $x \in X$ such that $\mathbf{P}(s, \sigma, x) > 0$, that is, $\mathbf{E}(s, \sigma)(\tau) = x$ and $\mathbf{D}(s, \sigma)(\tau) > 0$ for some $\tau \in T$. Thus $s \in \mathsf{Pre}_{\sigma,\tau}(X)$ and $D \in \mathcal{D}^{>0}_{\sigma,\tau}$. As $\mathsf{succ}(s, \sigma) \subseteq Y$, then for all s' such that $\mathbf{P}(s, \sigma, s') > 0$, we have $s' \in Y$, or equivalently, for all $\tau \in T$ such that $\mathbf{D}(s, \sigma)(\tau) > 0$, we have $\mathbf{E}(s, \sigma)(\tau) = s' \in Y$. Therefore, $s \in \mathsf{Pre}_{\sigma,\tau}(Y)$ for all τ such that $D \in \mathcal{D}^{>0}_{\sigma,\tau}$.

Suppose now that $s \in R(Y, X)$. Then we have that there exists $\sigma \in \Sigma$ and $D \in S_{\sim_{\mathbf{D},\sigma}}$ such that $s \in \bigcap_{\tau \in T, D \in \mathcal{D}^{>0}_{\sigma,\tau}} \mathsf{Pre}_{\sigma,\tau}(Y)$ and $s \in \bigcup_{\tau \in T, D \in \mathcal{D}^{>0}_{\sigma,\tau}} \mathsf{Pre}_{\sigma,\tau}(X)$. With the same arguments as above, we deduce that $\mathsf{succ}(s, \sigma) \subseteq Y$ and $\mathsf{succ}(s, \sigma) \cap X \neq \emptyset$. Notice that we have $\sigma \in \Sigma_s$. It follows that $s \models \mathsf{APre}(Y, X)$. □

In the case of the MDPs treated in Sect. 7, G is closed and $\mathcal{D}^{>0}_{\sigma,\tau}$ is composed of at most one block D that is closed. It follows that all the intermediate sets manipulated by the algorithm computing S^P are closed. We thus have an efficient algorithm since it can be based on antichains only.

6.3.3 Stochastic shortest path

For the SSP, the initial strategy λ_0 must be proper. In the previous paragraph, we have presented an algorithm for computing the set $S^P = \nu Y \cdot \mu X \cdot (\mathsf{APre}(Y, X) \vee \mathsf{G})$ of proper states. One can directly extract a PA-representation of a proper strategy from the execution of this algorithm, as follows. During the greatest fix point computation, a sequence $Y_0, Y_1, \ldots, Y_k$ is computed such that $Y_0 = S$ and $Y_{k-1} = Y_k = S^p$. During the computation of Y_k, a least fix point computation is performed, leading to a sequence $X_0, X_1, \ldots, X_l$ such that $X_0 = G$ and $X_i = X_{i-1} \cup \mathsf{APre}(S^P, X_{i-1})$ for all i, $1 \leq i \leq l$, and $X_l = Y_k$. We define a strategy λ_0 incrementally with each set X_i. Initially, $\lambda_0(s)$ is any $\sigma \in \Sigma_s$ for each $s \in X_0$ (since G is reached). When X_i has been computed, then for each $s \in X_i \backslash X_{i-1}$, $\lambda_0(s)$ is any $\sigma \in \Sigma_s$ such

Algorithm 5 SPLIT(B, C, λ)

1: $P[0] := B$
2: **for** i in $[1, m]$ **do**
3: $\quad P_{\text{new}} :=$ INITTABLE(P, τ_i)
4: $\quad$ **for all** $(p, block)$ in P **do**
5: $\quad\quad P_{\text{new}}[p] := P_{\text{new}}[p] \cup (block \setminus \mathsf{Pre}_\lambda(C, \tau_i))$
6: $\quad\quad$ **for all** $D \in S_{\sim_{\mathbf{D},\lambda}}$ **do**
7: $\quad\quad\quad P_{\text{new}}[p + \mathbf{D}_\lambda(D)(\tau_i)] := P_{\text{new}}[p + \mathbf{D}_\lambda(D)(\tau_i)] \cup (block \cap D \cap \mathsf{Pre}_\lambda(C, \tau_i))$
8: $\quad P :=$ REMOVEEMPTYBLOCKS(P_{new})
9: return P

that $\mathsf{succ}(s, \sigma) \subseteq S^P$ and $\mathsf{succ}(s, \sigma) \cap X_{i-1} \neq \emptyset$. Doing so, each proper state is eventually associated with an action by λ_0. Note that this strategy is proper. Indeed, to simplify the argument, suppose G is absorbing.[9] Then by construction of λ_0, the bottom strongly connected components of the MC induced by λ_0 are all in G, and a classical result on MCs (see [4]) states that any infinite path will almost surely lead to one of those components. Finally, as all sets X_i, Y_j manipulated by the algorithm are PA-represented, we obtain a partition $S_{\sim_{\lambda_0}}$ such that each block $B \in S_{\sim_{\lambda_0}}$ is PA-represented.

6.4 Bisimulation lumping

We now consider the step of Algorithm 4 where Algorithm LUMP is called to compute the largest bisimulation $\sim_L$ of an MC $M_{\preceq,\lambda}$ induced by a strategy λ on $M_\preceq$ (line 4 with $\lambda = \lambda_n$). We here detail a pseudo-antichain-based version of Algorithm LUMP (see Algorithm 3) when $M_{\preceq,\lambda}$ is PA-represented. Recall that if $M_\preceq = (S, \Sigma, T, \mathbf{E}, \mathbf{D})$, then $M_{\preceq,\lambda} = (S, T, \mathbf{E}_\lambda, \mathbf{D}_\lambda)$. Equivalently, using the usual definition of an MC, $M_{\preceq,\lambda} = (S, \mathbf{P}_\lambda)$ with $\mathbf{P}_\lambda$ derived from $\mathbf{E}_\lambda$ and $\mathbf{D}_\lambda$ (see Sect. 2). Remember also the two equivalence relations $\sim_{\mathbf{D},\lambda}$ and $\sim_{\mathbf{C},\lambda}$ defined in Sect. 6.2.

The initial partition P computed by Algorithm 3 (line 1) is such that for all $s, s' \in S$, s and s' belong to the same block of P iff $\mathbf{C}_\lambda(s) = \mathbf{C}_\lambda(s')$. The initial partition P is thus $S_{\sim_{\mathbf{C},\lambda}}$.

Algorithm 3 needs to split blocks of partition P (line 7). This can be performed thanks to Algorithm SPLIT that we are going to describe (see Algorithm 5). Given two blocks $B, C \subseteq S$, this algorithm splits B into a partition P composed of sub-blocks $B_1, \ldots, B_k$ according to the probability of reaching C in $M_{\preceq,\lambda}$, i.e. for all $s, s' \in B$, we have $s, s' \in B_l$ for some l iff $\mathbf{P}_\lambda(s, C) = \mathbf{P}_\lambda(s', C)$.

Suppose that $T = \{\tau_1, \ldots, \tau_m\}$. This algorithm computes intermediate partitions P of B such that at step i, B is split according to the probability of reaching C in $M_{\preceq,\lambda}$ when T is restricted to $\{\tau_1, \ldots, \tau_i\}$. To perform this task, it needs a new operator Pre_λ based on $M_\preceq$ and λ. Given $L \subseteq S$ and $\tau \in T$, we define

$$\mathsf{Pre}_\lambda(L, \tau) = \{s \in S \mid \mathbf{E}_\lambda(s)(\tau) \in L\}$$

as the set of states from which L is reached by τ in $M_\preceq$ under the selection made by λ. Notice that when T is restricted to $\{\tau_1, \ldots, \tau_i\}$ with $i < m$, it may happen that $\mathbf{D}_\lambda(s)$ is no longer a probability distribution for some $s \in S$ (when $\sum_{\tau \in \{\tau_1, \ldots, \tau_i\}} \mathbf{D}_\lambda(s)(\tau) < 1$).

Initially, T is restricted to $\emptyset$, and the partition P is composed of one block B (see line 1). At step i with $i \geq 1$, each block B_l of the partition computed at step $i - 1$ is split into several sub-blocks according to its intersection with $\mathsf{Pre}_\lambda(C, \tau_i)$ and each $D \in S_{\sim_{\mathbf{D},\lambda}}$. We take into account intersections with $D \in S_{\sim_{\mathbf{D},\lambda}}$ in a way to know which stochastic function $\mathbf{D}_\lambda(D)$

[9] for all $s \in G$ and $\sigma \in \Sigma_s$, $\sum_{s' \in G} \mathbf{P}(s, \sigma, s') = 1$.

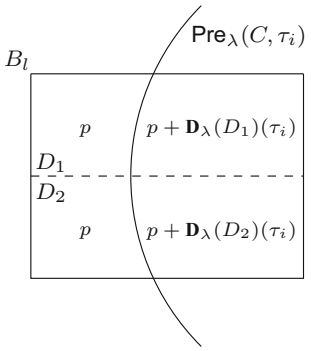

Fig. 5 Step i of Algorithm 5 on a block B_l

is associated with the states we are considering. Suppose that at step $i-1$ the probability for any state of block B_l of reaching C is p. Then at step i, it is equal to $p + \mathbf{D}_\lambda(D)(\tau_i)$ if this state belongs to $D \cap \mathsf{Pre}_\lambda(C, \tau_i)$, with $D \in S_{\sim_{\mathbf{D},\lambda}}$, and to p if it does not belong to $\mathsf{Pre}_\lambda(C, \tau_i)$ (lines 5–7). See Fig. 5 for intuition. Notice that some newly created sub-blocks could have the same probability, they are therefore merged.

The intermediate partitions P (or P_{new}) manipulated by the algorithm are represented by hash tables: each entry $(p, block)$ is stored as $P[p] = block$ such that $block$ is the set of states that reach C with probability p. The use of hash tables permits to efficiently gather sub-blocks of states having the same probability of reaching C, and thus to keep minimal the number of blocks in the partition. Algorithm InitTable is used to initialize a new partition P_{new} from a previous partition P and symbol τ_i: the new hash table is initialized with $P_{\text{new}}[p] := \emptyset$ and $P_{\text{new}}[p + \mathbf{D}_\lambda(D)(\tau_i)] := \emptyset$, for all $D \in S_{\sim_{\mathbf{D},\lambda}}$ and all $(p, block)$ in P. Algorithm RemoveEmptyBlocks(P) removes from the hash table P each pair $(p, block)$ such that $block = \emptyset$.

Theorem 1 *Let λ be a strategy on $M_{\preceq}$ and $M_{\preceq,\lambda} = (S, \boldsymbol{P}_\lambda)$ be the induced MC. Let $B, C \subseteq S$ be two blocks. Then the output of* Split(B, C, λ) *is a partition $\{B_1, \ldots, B_k\}$ of B such that for all $s, s' \in B$, $s, s' \in B_l$ for some l iff $\boldsymbol{P}_\lambda(s, C) = \boldsymbol{P}_\lambda(s', C)$.*

Proof The correctness of Algorithm Split is based on the following invariant. At step i, $0 \leq i \leq m$, with T restricted to $\{\tau_1, \ldots, \tau_i\}$, we have:

- P is a partition of B,
- $\forall s \in B$, if $s \in P[p]$, then $\mathbf{P}_\lambda(s, C) = p$.

Let us first prove that P is a partition. Note that the use of algorithm RemoveEmptyBlocks ensures that P never contains an empty block. Recall that $S_{\sim_{\mathbf{D},\lambda}}$ is a partition of S.

Initially, when $i = 0$, P is composed of the unique block B. Let $i \geq 1$ and suppose that $P = \{B_1, \ldots, B_k\}$ is a partition of B at step $i-1$, and let us prove that P_{new} is a partition of B (see line 7 of the algorithm). Each $B_l \in P$ is partitioned as $\{B_l \setminus \mathsf{Pre}_\lambda(C, \tau_i)\} \cup \{B_l \cap D \cap \mathsf{Pre}_\lambda(C, \tau_i) \mid D \in S_{\sim_{\mathbf{D},\lambda}}\}$. This leads to the finer partition $P' = \{B_l \setminus \mathsf{Pre}_\lambda(C, \tau_i) \mid B_l \in P\} \cup \{B_l \cap D \cap \mathsf{Pre}_\lambda(C, \tau_i) \mid B_l \in P, D \in S_{\sim_{\mathbf{D},\lambda}}\}$ of P. Some blocks of P' are gathered by the algorithm to get P_{new} which is thus a partition.

Let us now prove that at each step i, with T restricted to $\{\tau_1, \ldots, \tau_i\}$, we have: $\forall s \in B$, if $s \in P[p]$, then $\mathbf{P}_\lambda(s, C) = p$.

Initially, when $i = 0$, T is restricted to $\emptyset$, and thus $\mathbf{P}_\lambda(s, C) = 0$ for all $s \in B$. Let $i \geq 1$ and suppose we have that $\forall s \in B$, if $s \in P[p]$ for some p, then $\mathbf{P}_\lambda(s, C) = p$, when T is restricted

to $\{\tau_1, \ldots, \tau_{i-1}\}$. Let us prove that if $s \in P_{\text{new}}[p]$ for some p, then $\mathbf{P}_\lambda(s, C) = p$, when T is restricted to $\{\tau_1, \ldots, \tau_i\}$. Let $s \in B$ be such that $s \in P_{\text{new}}[p]$, we identify two cases: either (i) $s \in P[p]$ and $s \notin \mathsf{Pre}_\lambda(C, \tau_i)$, or (ii) $s \in P[p - \mathbf{D}_\lambda(D)(\tau_i)]$ and $s \in D \cap \mathsf{Pre}_\lambda(C, \tau_i)$, for some $D \in S_{\sim_{\mathbf{D},\lambda}}$. In case (i), by induction hypothesis, we know that $\mathbf{P}_\lambda(s, C) = p$, when $T = \{\tau_1, \ldots, \tau_{i-1}\}$, and since $s \notin \mathsf{Pre}_\lambda(C, \tau_i)$, adding τ_i to T does not change the probability of reaching C from s, i.e. $\mathbf{P}_\lambda(s, C) = p$ when $T = \{\tau_1, \ldots, \tau_i\}$. In case (ii), by induction hypothesis, we know that $\mathbf{P}_\lambda(s, C) = p - \mathbf{D}_\lambda(D)(\tau_i)$, when $T = \{\tau_1, \ldots, \tau_{i-1}\}$. Moreover, since $s \in D \cap \mathsf{Pre}_\lambda(C, \tau_i)$, we have $\mathbf{P}_\lambda(s, C) = \mathbf{D}_\lambda(D)(\tau_i)$, when $T = \{\tau_i\}$. It follows that $\mathbf{P}_\lambda(s, C) = p - \mathbf{D}_\lambda(D)(\tau_i) + \mathbf{D}_\lambda(D)(\tau_i) = p$, when $T = \{\tau_1, \ldots, \tau_i\}$.

Finally, after step $i = m$, we get the statement of Theorem 1 since P is the final partition of B and for all $s \in B$, $s \in P[p]$ iff $\mathbf{P}_\lambda(s, C) = p$.[10] □

Notice that we have a pseudo-antichain version of Algorithm LUMP as soon as the given blocks B and C are PA-represented. Indeed, this algorithm uses boolean operations and the Pre_λ operator. This operator can be computed as follows:

$$\mathsf{Pre}_\lambda(C, \tau) = \bigcup \left\{ \mathsf{Pre}_{\sigma,\tau}(C) \cap B \mid \sigma \in \Sigma, B \in S_{\sim_\lambda}, \lambda(B) = \sigma \right\}.$$

Intuitively, let us fix $\sigma \in \Sigma$ and $B \in S_{\sim_\lambda}$ such that $\lambda(B) = \sigma$. Then $\mathsf{Pre}_{\sigma,\tau}(C) \cap B$ is the set of states of $S \cap B$ that reach C in the MDP $M_{\preceq}$ with σ followed by τ. Finally the union gives the set of states that reach C with τ under the selection made by λ. All these operations can be performed thanks to Propositions 4, 5, and Assumptions 1, 2.

6.5 Solving linear systems

Assume that we have computed the largest bisimulation $\sim_L$ for the MC $M_{\preceq,\lambda} = (S, \mathbf{P}_\lambda)$. Let us now detail lines 5–7 of Algorithm 4 concerning the system of linear equations that has to be solved. We build the Markov chain that is the bisimulation quotient $(S_{\sim_L}, \mathbf{P}_{\lambda,\sim_L})$. We then explicitly solve the linear system of Algorithm 1 for the SSP (resp. Algorithm 2 for the EMP) (line 3). We thus obtain the expected truncated sum v (resp. the gain value g and bias value b) of the strategy λ, for each block $B \in S_{\sim_L}$. By definition of $\sim_L$, we have that for all $s, s' \in S$, if $s \sim_L s'$, then $v(s) = v(s')$ (resp. $g(s) = g(s')$ and $b(s) = b(s')$). Given a block $B \in S_{\sim_L}$, we denote by $v(B)$ the unique expected truncated sum $v(s)$ (resp. by $g(B)$ the unique gain value $g(s)$ and by $b(B)$ the unique bias value $b(s)$), for all $s \in B$.

6.6 Improving strategies

Given an MDP $M_{\preceq}$ with cost function $\mathbf{C}$ and the MC $M_{\preceq,\lambda}$ induced by a strategy λ, we finally present a pseudo-antichain-based algorithm to improve strategy λ for the SSP, with the expected truncated sum v obtained by solving the linear system (see line 8 of Algorithm 4, and Algorithm 1).[11] Starting from the largest bisimulation $\sim_L$ of $M_{\preceq,\lambda}$ and the induced partition $S_{\sim_L}$ (as computed before), we are going to define a new equivalence relation that will allow us to improve strategy λ at the level of each block of this relation.

Recall (see Algorithm 1) that for all $s \in S$, we compute the set $\widehat{\Sigma_s}$ of actions $\sigma \in \Sigma_s$ that minimize the expression

$$l_\sigma(s) = \mathbf{C}(s, \sigma) + \sum_{s' \in S} \mathbf{P}(s, \sigma, s') \cdot v(s'),$$

[10] The "iff" holds since probabilities p are pairwise distinct.

[11] The improvement of a strategy for the EMP, with the gain g or the bias b values (see Algorithm 2), is similar and is thus not detailed.

and then we improve the strategy based on the computed $\widehat{\Sigma_s}$. We give hereafter an approach based on pseudo-antichains which requires the next two steps. The first step consists in computing, for all $\sigma \in \Sigma$, an equivalence relation $\sim_{l_\sigma}$ such that the value $l_\sigma(s)$ is constant on each block of the relation. The second step uses the relations $\sim_{l_\sigma}$, with $\sigma \in \Sigma$, to improve the strategy.

6.6.1 Computing value

l_σ. Let $\sigma \in \Sigma$ be a fixed action. We are looking for an equivalence relation $\sim_{l_\sigma}$ on the set S_σ of states where action σ is enabled, such that

$$\forall s, s' \in S_\sigma : s \sim_{l_\sigma} s' \Rightarrow l_\sigma(s) = l_\sigma(s').$$

Given $\sim_L$ and the induced partition $S_{\sim_L}$, we have for each $s \in S_\sigma$

$$l_\sigma(s) = \mathbf{C}(s, \sigma) + \sum_{C \in S_{\sim_L}} \mathbf{P}(s, \sigma, C) \cdot v(C)$$

since the value v is constant on each block C. Therefore to get relation $\sim_{l_\sigma}$, it is enough to have $s \sim_{l_\sigma} s' \Rightarrow \mathbf{C}(s, \sigma) = \mathbf{C}(s', \sigma)$ and $\mathbf{P}(s, \sigma, C) = \mathbf{P}(s', \sigma, C), \forall C \in S_{\sim_L}$. We proceed by defining the following equivalence relations on S_σ. For the cost part, we use relation $\sim_{\mathbf{C},\sigma}$ defined in Sect. 6.2. For the probabilities part, for each block C of $S_{\sim_L}$, we define relation $\sim_{\mathbf{P},\sigma,C}$ such that $s \sim_{\mathbf{P},\sigma,C} s'$ iff $\mathbf{P}(s, \sigma, C) = \mathbf{P}(s', \sigma, C)$. The required relation $\sim_{l_\sigma}$ on S_σ is then defined as the relation

$$\sim_{l_\sigma} \;=\; \sim_{\mathbf{C},\sigma} \cap \bigcap_{C \in S_{\sim_L}} \sim_{\mathbf{P},\sigma,C} \;=\; \sim_{\mathbf{C},\sigma} \cap \sim_{\mathbf{P},\sigma}$$

Let us explain how to compute $\sim_{l_\sigma}$ with a pseudo-antichain-based approach. Firstly, $M_{\preceq}$ being T-complete, the set S_σ is obtained as $S_\sigma = \mathsf{Pre}_{\sigma,\tau}(S)$ where τ is an arbitrary action of T. Secondly, each relation $\sim_{\mathbf{P},\sigma,C}$ is the output obtained by a call to SPLIT(S_σ, C, λ) where λ is defined on S_σ by $\lambda(s) = \sigma$ for all $s \in S_\sigma$[12] (see Algorithm 5). Thirdly, we detail a way to compute $\sim_{\mathbf{P},\sigma}$ from $\sim_{\mathbf{P},\sigma,C}$, for all $C \in S_{\sim_L}$. Let $S_{\sim_{\mathbf{P},\sigma,C}} = \{B_{C,1}, B_{C,2}, \ldots, B_{C,k_C}\}$ be the partition of S_σ induced by $\sim_{\mathbf{P},\sigma,C}$. For each $B_{C,i} \in S_{\sim_{\mathbf{P},\sigma,C}}$, we denote by $\mathbf{P}(B_{C,i}, \sigma, C)$ the unique value $\mathbf{P}(s, \sigma, C)$, for all $s \in B_{C,i}$. Then, computing a block D of $\sim_{\mathbf{P},\sigma}$ consists in picking, for all $C \in S_{\sim_L}$, one block D_C among $B_{C,1}, B_{C,2}, \ldots, B_{C,k_C}$, such that the intersection $D = \bigcap_{C \in S_{\sim_L}} D_C$ is non empty. Recall that, by definition of the MDP $M_{\preceq}$, we have $\sum_{s' \in S} \mathbf{P}(s, \sigma, s') = 1$. Therefore, if D is non empty, then $\sum_{C \in S_{\sim_L}} \mathbf{P}(D_C, \sigma, C) = 1$. Finally, $\sim_{l_\sigma}$ is obtained as the intersection between $\sim_{\mathbf{C},\sigma}$ and $\sim_{\mathbf{P},\sigma}$.

Relation $\sim_{l_\sigma}$ induces a partition of S_σ that we denote $(S_\sigma)_{\sim_{l_\sigma}}$. For each block $D \in (S_\sigma)_{\sim_{l_\sigma}}$, we denote by $l_\sigma(D)$ the unique value $l_\sigma(s)$, for $s \in D$.

6.6.2 Improving the strategy

We now propose a pseudo-antichain-based algorithm for improving strategy λ by using relations $\sim_L$, $\sim_\lambda$, and $\sim_{l_\sigma}, \forall \sigma \in \Sigma$ (see Algorithm 6).

We first compute for all $\sigma \in \Sigma$, the equivalence relation $\sim_{l_\sigma \wedge L} = \sim_{l_\sigma} \cap \sim_L$ on S_σ. Given $B \in (S_\sigma)_{\sim_{l_\sigma \wedge L}}$, we denote by $l_\sigma(B)$ the unique value $l_\sigma(s)$ and by $v(B)$ the unique value

[12] As Algorithm SPLIT only works on S_σ, it is not a problem if λ is not defined on $S \backslash S_\sigma$.

Algorithm 6 IMPROVESTRATEGY($\mathcal{L}$, $S_{\sim_\lambda}$)

1: **for** $C \in \mathcal{L}$ **do**
2: $S_{\sim_{\lambda'}} := \emptyset$
3: **for** $B \in S_{\sim_\lambda}$ **do**
4: **if** $B \cap C \neq \emptyset$ **then**
5: $S_{\sim_{\lambda'}} := S_{\sim_{\lambda'}} \cup \{B \cap C, B \backslash C\}$
6: **else**
7: $S_{\sim_{\lambda'}} := S_{\sim_{\lambda'}} \cup B$
8: $S_{\sim_\lambda} := S_{\sim_{\lambda'}}$
9: return $S_{\sim_{\lambda'}}$

$v(s)$, for all $s \in B$. Let $\sigma \in \Sigma$, we denote by $(S_\sigma)^{<}_{\sim_{l_\sigma \wedge L}} \subseteq (S_\sigma)_{\sim_{l_\sigma \wedge L}}$ the set of blocks C for which the value $v(C)$ is improved by setting $\lambda(C) = \sigma$, that is

$$(S_\sigma)^{<}_{\sim_{l_\sigma \wedge L}} = \{C \in (S_\sigma)_{\sim_{l_\sigma \wedge L}} \mid l_\sigma(C) < v(C)\}.$$

We then compute an ordered global list $\mathcal{L}$ made of the blocks of all sets $(S_\sigma)^{<}_{\sim_{l_\sigma \wedge L}}$, for all $\sigma \in \Sigma$. It is ordered according to the decreasing value $l_\sigma(C)$. In this way, when traversing $\mathcal{L}$, we have more and more promising blocks to decrease v.

From input $\mathcal{L}$ and $\sim_\lambda$, Algorithm 6 outputs an equivalence relation $\sim_{\lambda'}$ for a new strategy λ' that improves λ. Given $C \in \mathcal{L}$, suppose that C comes from the relation $\sim_{l_\sigma \wedge L}$ (σ is considered). Then for each $B \in S_{\sim_\lambda}$ such that $B \cap C \neq \emptyset$ (line 4), we improve the strategy by setting $\lambda'(B \cap C) = \sigma$, while the strategy λ' is kept unchanged for $B \backslash C$. Algorithm 6 outputs a partition $S_{\sim_{\lambda'}}$ such that $s \sim_{\lambda'} s' \Rightarrow \lambda'(s) = \lambda'(s')$ for the improved strategy λ'. If necessary, for efficiency reasons, we can compute a coarser relation for the new strategy λ' by gathering blocks B_1, B_2 of $S_{\sim_{\lambda'}}$, for all B_1, B_2 such that $\lambda'(B_1) = \lambda'(B_2)$.

The correctness of Algorithm 6 is due to the list $\mathcal{L}$, which is sorted according to the decreasing value $l_\sigma(C)$. It ensures that the strategy is updated at each state s to an action $\sigma \in \widehat{\Sigma_s}$, i.e. an action σ that minimizes the expression $\mathbf{C}(s, \sigma) + \sum_{s' \in S} \mathbf{P}(s, \sigma, s') \cdot v_n(s')$ (cf. line 4 of Algorithm 1).

7 Experiments

In this section, we present two application scenarios of the pseudo-antichain-based symblicit algorithm of the previous section, one for the SSP problem and the other for the EMP problem. In both cases, we first show the reduction to monotonic MDPs that satisfy Assumptions 1 and 2, and we then present some experimental results. We also give a different application of pseudo-antichains to the qualitative analysis of probabilistic lossy channel systems, based on results of [2]. This third application scenario shows that one can manage infinite (rather than finite) MDPs that appear in this context, and illustrates the variety of applications of pseudo-antichains. All our experiments were performed on a Linux platform with a 3.2 GHz CPU (Intel Core i7) and 4 GB of memory. Note that our implementations are single-threaded and thus use only one core. For all those experiments, the timeout is set to 10 h and is denoted by **TO**, and when an execution runs out of memory, we denote it by **MO**.

7.1 Stochastic shortest path on STRIPSs

We consider the following application of the pseudo-antichain-based symblicit algorithm for the SSP problem. In the field of planning, a class of problems called *planning from*

STRIPSs [22] operate with states represented by valuations of propositional variables. Informally, a STRIPS is defined by an *initial* state representing the initial configuration of the system and a set of *operators* that transform a state into another state. The problem of planning from STRIPSs then asks, given a valuation of propositional variables representing a set of *goal* states, to find a sequence of operators that lead from the initial state to a goal one. Let us first formally define the notion of STRIPS and show that each STRIPS can be made monotonic. We will then add stochastic aspects and show how to construct a monotonic MDP from a monotonic stochastic STRIPS.

7.1.1 STRIPSs

A *STRIPS* [22] is a tuple $(P, I, (M, N), O)$ where P is a finite set of *conditions* (i.e. propositional variables), $I \subseteq P$ is a subset of conditions that are initially true (all others are assumed to be false), (M, N), with $M, N \subseteq P$ and $M \cap N = \emptyset$, specifies which conditions are true and false, respectively, in order for a state to be considered a goal state, and O is a finite set of *operators*. An operator $o \in O$ is a pair $((\gamma, \theta), (\alpha, \delta))$ such that (γ, θ) is the *guard* of o, that is, $\gamma \subseteq P$ (resp. $\theta \subseteq P$) is the set of conditions that must be true (resp. false) for o to be executable, and (α, δ) is the *effect* of o, that is, $\alpha \subseteq P$ (resp. $\delta \subseteq P$) is the set of conditions that are made true (resp. false) by the execution of o. For all $((\gamma, \theta), (\alpha, \delta)) \in O$, we have that $\gamma \cap \theta = \emptyset$ and $\alpha \cap \delta = \emptyset$.

From a STRIPS, we derive a transition system as follows. The set of states is 2^P, that is, a state is represented by the set of conditions that are true in it. The initial state is I. The set of goal states are states Q such that $Q \supseteq M$ and $Q \cap N = \emptyset$. There is a transition from state Q to state Q' under operator $o = ((\gamma, \theta), (\alpha, \delta))$ if $Q \supseteq \gamma$, $Q \cap \theta = \emptyset$ (the guard is satisfied) and $Q' = (Q \cup \alpha) \backslash \delta$ (the effect is applied). A standard problem is to ask whether or not there exists a path from the initial state to a goal state.

7.1.2 Monotonic STRIPSs

A *monotonic STRIPS (MS)* is a tuple (P, I, M, O) where P and I are defined as for STRIPSs, $M \subseteq P$ specifies which conditions must be true in a goal state, and O is a finite set of operators. In the MS definition, an operator $o \in O$ is a pair $(\gamma, (\alpha, \delta))$ where $\gamma \subseteq P$ is the guard of o, that is, the set of conditions that must be true for o to be executable, and (α, δ) is the effect of o as in the STRIPS definition. MSs thus differ from STRIPS in the sense that guards only apply on conditions that are true in states, and goal states are only specified by true conditions. The monotonicity will appear more clearly when we will derive hereafter monotonic MDPs from MSs.

Each STRIPS $\mathcal{S} = (P, I, (M, N), O)$ can be made monotonic by duplicating the set of conditions, in the following way. We denote by $\overline{P}$ the set $\{\overline{p} \mid p \in P\}$ containing a new condition $\overline{p}$ for each $p \in P$ such that $\overline{p}$ represents the negation of the propositional variable p. We construct from $\mathcal{S}$ an MS $\mathcal{S}' = (P', I', M', O')$ such that $P' = P \cup \overline{P}$, $I' = I \cup \overline{P \backslash I} \subseteq P'$, $M' = M \cup \overline{N} \subseteq P'$ and $O' = \{(\gamma \cup \overline{\theta}, (\alpha \cup \overline{\delta}, \delta \cup \overline{\alpha})) \mid ((\gamma, \theta), (\alpha, \delta)) \in O\}$. It is easy to check that $\mathcal{S}$ and $\mathcal{S}'$ are equivalent (a state Q in $\mathcal{S}$ has its counterpart $Q \cup \overline{P \backslash Q}$ in $\mathcal{S}'$). In the following, we thus only consider MSs.

Example 5 To illustrate the notion of MS, let us consider the following example of the monkey trying to reach a bunch of bananas (cf. Example 1). Let (P, I, M, O) be an MS such that $P = \{box, stick, bananas\}$, $I = \emptyset$, $M = \{bananas\}$, and $O = \{takebox, takestick, takebananas\}$ where $takebox = (\emptyset, (\{box\}, \emptyset))$, $takestick =$

$(\emptyset, (\{stick\}, \emptyset))$ and $takebananas = (\{box, stick\}, (\{bananas\}, \emptyset))$. In this MS, a condition $p \in P$ is true when the monkey possesses the item corresponding to p. At the beginning, the monkey possesses no item, i.e. I is the empty set, and its goal is to get the *bananas*, i.e. to reach a state $s \supseteq \{bananas\}$. This can be done by first executing the operators *takebox* and *takestick* to respectively get the *box* and the *stick*, and then executing *takebananas*, whose guard is $\{box, stick\}$.

7.1.3 Monotonic stochastic STRIPSs

MSs can be extended with stochastic aspects as follows [9]. Each operator $o = (\gamma, \pi) \in O$ now consists of a guard γ as before, and an effect given as a probability distribution $\pi : 2^P \times 2^P \to [0, 1]$ on the set of pairs (α, δ). An MS extended with such stochastic aspects is called a *monotonic stochastic STRIPS (MSS)*.

Additionally, we associate with an MSS (P, I, M, O) a cost function $C : O \to \mathbb{R}_{>0}$ that associates a strictly positive cost with each operator. The problem of planning from MSSs is then to minimize the expected truncated sum up to the set of goal states from the initial state, i.e. this is a version of the SSP problem.

Example 6 We extend the MS of Example 5 with stochastic aspects to illustrate the notion of MSS. Let (P, I, M, O) be an MSS such that P, I and M are defined as in Example 5, and $O = \{takebox, takestick, takebananaswithbox, takebananaswithstick, takebananaswithboth\}$ where

- $takebox = (\emptyset, (1 : (\{box\}, \emptyset)))$,
- $takestick = (\emptyset, (1 : (\{stick\}, \emptyset)))$,
- $takebananaswithbox = \big(\{box\}, \big(\frac{1}{4} : (\{bananas\}, \emptyset), \frac{3}{4} : (\emptyset, \emptyset)\big)\big)$,
- $takebananaswithstick = \big(\{stick\}, \big(\frac{1}{5} : (\{bananas\}, \emptyset), \frac{4}{5} : (\emptyset, \emptyset)\big)\big)$, and
- $takebananaswithboth = \big(\{box, stick\}, \big(\frac{1}{2} : (\{bananas\}, \emptyset), \frac{1}{2} : (\emptyset, \emptyset)\big)\big)$.

In this MSS, the monkey has a strictly positive probability to fail reaching the *bananas*, whatever the items it uses. However, the probability of success increases when it has both the *box* and the *stick*.

In the following, we show that MSSs naturally define monotonic MDPs on which the pseudo-antichain-based symblicit algorithm of Sect. 6 can be applied.

7.1.4 From MSSs to monotonic MDPs

Let $\mathcal{S} = (P, I, M, O)$ be an MSS. We can derive from $\mathcal{S}$ an MDP $M_{\mathcal{S}} = (S, \Sigma, T, \mathbf{E}, \mathbf{D})$ together with a set of goal states G and a cost function $\mathbf{C}$ such that:

- $S = 2^P$,
- $G = \{s \in S \mid s \supseteq M\}$,
- $\Sigma = O$, and for all $s \in S$, $\Sigma_s = \{(\gamma, \pi) \in \Sigma \mid s \supseteq \gamma\}$,
- $T = \{(\alpha, \delta) \in 2^P \times 2^P \mid \exists (\gamma, \pi) \in O, (\alpha, \delta) \in \mathsf{Supp}(\pi)\}$,
- $\mathbf{E}$, $\mathbf{D}$ and $\mathbf{C}$ are defined for all $s \in S$ and $\sigma = (\gamma, \pi) \in \Sigma_s$, such that:
  - for all $\tau = (\alpha, \delta) \in T$, $\mathbf{E}(s, \sigma)(\tau) = (s \cup \alpha) \setminus \delta$,
  - for all $\tau \in T$, $\mathbf{D}(s, \sigma)(\tau) = \pi(\tau)$, and
  - $\mathbf{C}(s, \sigma) = C(\sigma)$.

Note that we might have that $M_{\mathcal{S}}$ is not Σ-non-blocking, if no operator can be applied on some state of $\mathcal{S}$. In this case, we get a Σ-non-blocking MDP from $M_{\mathcal{S}}$ by eliminating states s with $\Sigma_s = \emptyset$ as long as it is necessary.

Lemma 5 *The MDP $M_{\mathcal{S}}$ is monotonic, G is closed, and functions $\mathbf{D}, \mathbf{C}$ are independent from S.*

Proof First, S is equipped with the partial order $\supseteq$ and $(S, \supseteq)$ is a semilattice. Second, S is closed for $\supseteq$ by definition. Thirdly, we have that $\supseteq$ is compatible with $\mathbf{E}$. Indeed, for all $s, s' \in S$ such that $s \supseteq s'$, for all $\sigma \in \Sigma$ and $\tau = (\alpha, \delta) \in T$, $\mathbf{E}(s, \sigma)(\tau) = (s \cup \alpha)\backslash\delta \supseteq (s' \cup \alpha)\backslash\delta = \mathbf{E}(s', \sigma)(\tau)$. Finally the set $G = \downarrow\{M\}$ of goal states is closed for $\supseteq$, and $\mathbf{D}$, $\mathbf{C}$ are clearly independent from S. □

7.1.5 Symblicit algorithm

In order to apply the pseudo-antichain-based symblicit algorithm of Sect. 6 on the monotonic MDPs derived from MSSs, Assumptions 1 and 2 must hold. Let us show that Assumption 2 is satisfied. For all $s \in S$, $\sigma = (\gamma, \pi) \in \Sigma_s$ and $\tau = (\alpha, \delta) \in T$, we clearly have an algorithm for computing $\mathbf{E}(s, \sigma)(\tau) = (s \cup \alpha)\backslash\delta$, and $\mathbf{D}(s, \sigma)(\tau) = \pi(\tau)$. Let us now consider Assumption 1. An algorithm for computing $\lceil \mathsf{Pre}_{\sigma,\tau}(\downarrow\{x\}) \rceil$, for all $x \in S$, $\sigma \in \Sigma$ and $\tau \in T$, is given by the next proposition.

Proposition 6 *Let $x \in S$, $\sigma = (\gamma, \pi) \in \Sigma$ and $\tau = (\alpha, \delta) \in T$. If $x \cap \delta \neq \emptyset$, then $\lceil \mathsf{Pre}_{\sigma,\tau}(\downarrow\{x\}) \rceil = \emptyset$, otherwise $\lceil \mathsf{Pre}_{\sigma,\tau}(\downarrow\{x\}) \rceil = \{\gamma \cup (x\backslash\alpha)\}$.*

Proof Suppose first that $x \cap \delta = \emptyset$.

We first prove that $s = \gamma \cup (x\backslash\alpha) \in \mathsf{Pre}_{\sigma,\tau}(\downarrow\{x\})$. We have to show that $\sigma \in \Sigma_s$ and $\mathbf{E}(s, \sigma)(\tau) \in \downarrow\{x\}$. Recall that $\sigma = (\gamma, \pi)$. We have that $s = \gamma \cup (x\backslash\alpha) \supseteq \gamma$, showing that $\sigma \in \Sigma_s$. We have that $\mathbf{E}(s, \sigma)(\tau) = (\gamma \cup (x\backslash\alpha) \cup \alpha)\backslash\delta = (\gamma \cup x \cup \alpha)\backslash\delta \supseteq x$ since $x \cap \delta = \emptyset$. We thus have that $\mathbf{E}(s, \sigma)(\tau) \in \downarrow\{x\}$.

We then prove that for all $s \in \mathsf{Pre}_{\sigma,\tau}(\downarrow\{x\})$, $s \in \downarrow\{\gamma \cup (x\backslash\alpha)\}$, i.e. $s \supseteq \gamma \cup (x\backslash\alpha)$. Let $s \in \mathsf{Pre}_{\sigma,\tau}(\downarrow\{x\})$. We have that $\sigma \in \Sigma_s$ and $\mathbf{E}(s, \sigma)(\tau) \in \downarrow\{x\}$, that is, $s \supseteq \gamma$ and $\mathbf{E}(s, \sigma)(\tau) = (s \cup \alpha)\backslash\delta \supseteq x$. By classical set properties, it follows that $(s \cup \alpha) \supseteq x$, and then $s \supseteq x\backslash\alpha$. Finally, since $s \supseteq \gamma$, we have $s \supseteq \gamma \cup (x\backslash\alpha)$, as required.

Suppose now that $x \cap \delta \neq \emptyset$, then $\mathsf{Pre}_{\sigma,\tau}(\downarrow\{x\}) = \emptyset$. Indeed for all $s \in \downarrow\{x\}$, we have $s \cap \delta \neq \emptyset$, and by definition of $\mathbf{E}$, there is no s' such that $\mathbf{E}(s', \sigma)(\tau) = s$. □

Finally, notice that for the class of monotonic MDPs derived from MSSs, the symbolic representations described in Sect. 6.2 are compact, since G is closed and $\mathbf{D}$, $\mathbf{C}$ are independent from S (see Lemma 5). Therefore we have all the required ingredients for an efficient pseudo-antichain-based algorithm to solve the SSP problem for MSSs. The next experiments show its performance.

7.1.6 Experiments

We have implemented in Python and C the pseudo-antichain-based symblicit algorithm for the SSP problem. The C language is used for all the low level operations while the orchestration is done with Python. The binding between C and Python is realized with the ctypes library of Python. The source code is publicly available at http://lit2.ulb.ac.be/STRIPSSolver/, together with the two benchmarks presented in this section. We compared

our implementation with the purely explicit strategy iteration algorithm implemented in the development release 4.1.dev.r7712 of the tool PRISM [35], since to the best of our knowledge, there is no tool implementing an MTBDD based symblicit algorithm for the SSP problem.[13] Note that this explicit implementation exists primarily to prototype new techniques and is thus not fully optimized [40]. Note that value iteration algorithms are also implemented in PRISM. While those algorithms are usually efficient, they only compute approximations. As a consequence, for the sake of a fair comparison, we consider here only the performances of strategy iteration algorithms.

The first benchmark (Monkey) is obtained from Example 6. In this benchmark, the monkey has several items at its disposal to reach the bunch of bananas, one of them being a stick. However, the stick is available as a set of several pieces that the monkey has to assemble. Moreover, the monkey has multiple ways to build the stick as there are several sets of pieces that can be put together. However, the time required to build a stick varies from a set of pieces to another. Additionally, we add useless items in the room: there is always a set of pieces from which the probability of getting a stick is 0. The operators of getting some items are stochastic, as well as the operator of getting the bananas: the probability of success varies according to the owned items (cf. Example 6). The benchmark is parameterized in the number p of pieces required to build a stick, and in the number s of sticks that can be built. Note that the monkey can only use one stick, and thus has no interest to build a second stick if it already has one. Results are given in Table 1.

The second benchmark (Moats and castles) is an adaptation of a benchmark of [37] as proposed in [9].[14] The goal is to build a sand castle on the beach; a moat can be dug before in a way to protect it. We consider up to 7 discrete depths of moat. The operator of building the castle is stochastic: there is a strictly positive probability for the castle to be demolished by the waves. However, the deeper the moat is, the higher the probability of success is. For example, the first depth of moat offers a probability $\frac{1}{4}$ of success, while with the second depth of moat, the castle has probability $\frac{9}{20}$ to resist to the waves. The optimal strategy for this problem is to dig up to a given depth of moat and then repeat the action of building the castle until it succeeds. The optimal depth of moat then depends on the cost of the operators and the respective probability of successfully building the castle for each depth of moat. To increase the difficulty of the problem, we consider building several castles, each one having its own moat. The benchmark is parameterized in the number d of depths of moat that can be dug, and the number c of castles that have to be built. Results are given in Table 2.

On those two benchmarks, we observe that the explicit implementation quickly runs out of memory when the state space of the MDP grows. Indeed, with this method, we were not able to solve MDPs with more than 65536 (resp. 32768) states in Table 1 (resp. Table 2). On the other hand, the symblicit algorithm behaves well on large models: the memory consumption never exceeds 150MB and this even for MDPs with hundreds of millions of states. For instance, the example (5, 5) of the Monkey benchmark is an MDP of more than 17 billions of states that is solved in $<$2 h with only 82 MB of memory.[15]

[13] A comparison with an MTBDD based symblicit algorithm is done in the second application for the EMP problem.

[14] In [9], the authors study a different problem that is to maximize the probability of reaching the goal within a given number of steps.

[15] On our benchmarks, the value iteration algorithm of PRISM performs better than the strategy iteration one w.r.t. the run time and memory consumption. However, it still consumes more memory than the pseudo-antichain-based algorithm, and runs out of memory on several examples.

Table 1 Stochastic shortest path on the Monkey benchmark

(s, p)	$\mathbb{E}_{\lambda}^{\mathrm{TS}_G}$	$\lvert M_{\mathcal{S}}\rvert$	PA						Explicit			
			#it	$\lvert S_{\sim_L}\rvert$	lump	syst	impr	total	mem	constr	strat	total	mem
(1,2)	35.75	256	4	15	0.01	0.00	0.02	0.03	15.6	0.4	0.03	0.43	178.2
(1,3)	35.75	1024	5	19	0.04	0.00	0.03	0.07	15.8	3.42	0.09	3.51	336.7
(1,4)	35.75	4096	6	31	0.17	0.00	0.12	0.29	16.3	55.29	0.16	55.45	1735.1
(1,5)	36.00	16,384	7	39	0.75	0.00	0.62	1.37	17.1				**MO**
(2,2)	34.75	1024	5	19	0.05	0.00	0.03	0.09	15.8	3.2	0.12	3.32	379.9
(2,3)	34.75	8192	5	37	0.32	0.00	0.13	0.45	16.4	240.66	0.30	240.96	3463.2
(2,4)	34.75	65,536	6	45	2.39	0.01	1.04	3.44	18.0				**MO**
(2,5)	35.75	524,288	7	65	27.56	0.02	10.13	37.71	23.4				**MO**
(3,2)	35.75	4096	4	23	0.09	0.00	0.07	0.16	16.0	60.43	0.16	60.59	1625.8
(3,3)	35.75	65,536	5	43	1.14	0.00	0.43	1.57	17.3				**MO**
(3,4)	35.75	1,048,576	6	57	12.89	0.01	4.92	17.83	21.7				**MO**
(3,5)	36.00	16,777,216	7	88	208.33	0.05	63.73	272.13	37.5				**MO**
(4,2)	35.75	16,384	4	29	0.22	0.02	0.14	0.38	16.3	1114.19	0.70	1114.89	1704.3
(4,3)	35.75	524,288	5	50	2.72	0.00	1.26	4.00	18.3				**MO**
(4,4)	35.75	16,777,216	6	87	45.68	0.04	22.41	68.14	25.0				**MO**
(4,5)	36.00	536,870,912	7	114	724.77	0.11	532.46	1257.41	60.9				**MO**
(5,2)	35.75	65,536	4	31	0.36	0.00	0.18	0.54	16.6	20,312.67	3.50	20,316.17	2342.6
(5,3)	35.75	4,194,304	5	56	5.71	0.02	2.47	8.20	19.5				**MO**
(5,4)	35.75	268,435,456	6	97	95.49	0.04	101.27	196.83	31.3				**MO**
(5,5)	36.00	17,179,869,184	7	152	1813.78	0.08	5284.31	7098.40	81.3				**MO**

The column (s, p) gives the parameters of the problem, $\mathbb{E}_{\lambda}^{\mathrm{TS}_G}$ the expected truncated sum of the computed strategy λ, and $\lvert M_{\mathcal{S}}\rvert$ the number of states of the MDP. For the pseudo-antichain-based implementation (PA), #it is the number of iterations of the strategy iteration algorithm, $\lvert S_{\sim_L}\rvert$ the maximum size of computed bisimulation quotients, *lump* the total time spent for lumping, *syst* the total time spent for solving the linear systems, and *impr* the total time spent for improving the strategies. For the explicit implementation (Explicit), *constr* is the time spent for model construction and *strat* the time spent for the strategy iteration algorithm. For both implementations, *total* is the total execution time and *mem* the total memory consumption. All times are given in seconds and all memory consumptions are given in megabytes

Table 2 Stochastic shortest path on the Moats and castles benchmark

(c, d)	$\mathbb{E}_{\lambda}^{\mathrm{TS}_G}$	$\lvert M_{\mathcal{S}}\rvert$	PA							Explicit			
			#it	$\lvert S_{\sim_L}\rvert$	lump	syst	impr	total	mem	constr	strat	total	mem
(2,3)	39.3333	256	3	17	0.05	0.01	0.04	0.10	15.8	0.46	0.03	0.49	206.9
(2,4)	34.6667	1024	3	34	0.41	0.00	0.17	0.58	16.5	6.30	0.08	6.38	483.8
(2,5)	32.2222	4096	3	49	1.36	0.00	0.45	1.82	17.3	133.46	0.20	133.66	1202.5
(2,6)	32.2222	16,384	3	66	9.71	0.01	1.95	11.68	19.3	2966.01	0.79	2966.80	1706.2
(3,2)	72.6667	512	3	45	0.52	0.00	0.24	0.77	16.6	1.77	0.06	1.83	282.9
(3,3)	59.0000	4096	3	84	12.58	0.03	2.73	15.35	20.2	149.44	0.20	149.64	1205.5
(3,4)	52.0000	32,768	3	219	129.17	0.05	21.56	150.83	30.7	14,658.22	2.47	14,660.69	1610.9
(3,5)	48.3333	262,144	3	357	658.86	0.13	81.08	740.17	49.1				**MO**
(3,6)	48.3333	2,097,152	3	595	10,730.09	0.42	865.48	11,596.71	145.8				**MO**
(4,2)	96.8889	4096	3	132	31.61	0.03	12.06	43.72	26.5	173.40	0.22	173.62	1211.2
(4,3)	78.6667	65,536	3	464	1376.94	0.21	217.06	1594.48	82.2				**MO**

The column (c, d) gives the parameters of the problem and all other columns have the same meaning as in Table 1

7.2 Expected mean-payoff with $\mathsf{LTL}_{\mathsf{MP}}$ synthesis

We consider another application of the pseudo-antichain-based symblicit algorithm, but now for the EMP problem. This application is related to the problems of $\mathsf{LTL}_{\mathsf{MP}}$ realizability and synthesis [11,12]. Let us fix some notations and definitions. Let ϕ be an LTL formula defined over the set $P = I \uplus O$ of signals and let $\Sigma_P = 2^P$, $\Sigma_O = 2^O$ and $\Sigma_I = 2^I$. Let $\mathsf{Lit}(O) = \{o \mid o \in O\} \cup \{\neg o \mid o \in O\}$ be the set of literals over O. Let $w : \mathsf{Lit}(O) \mapsto \mathbb{Z}$ be a weight function where positive numbers represent rewards.[16] This function is extended to Σ_O as follows: $w(\sigma) = \Sigma_{o\in\sigma} w(o) + \Sigma_{o\in O\setminus\{\sigma\}} w(\neg o)$ for all $\sigma \in \Sigma_O$.

7.2.1 LTL_{MP} *realizability and synthesis*

The problem of $\mathsf{LTL}_{\mathsf{MP}}$ realizability is best seen as a game between two players, Player O and Player I. This game is infinite and such that at each turn k, Player O gives a subset $o_k \in \Sigma_O$ and Player I responds by giving a subset $i_k \in \Sigma_I$. The outcome of the game is the infinite word $(o_0 \cup i_0)(o_1 \cup i_1)\cdots \in \Sigma_P^\omega$. A strategy for Player O is a mapping $\lambda_O : (\Sigma_O \Sigma_I)^* \to \Sigma_O$, while a strategy for Player I is a mapping $\lambda_I : (\Sigma_O \Sigma_I)^* \Sigma_O \to \Sigma_I$. The outcome of the strategies λ_O and λ_I is the word $\mathsf{Out}(\lambda_O, \lambda_I) = (o_0 \cup i_0)(o_1 \cup i_1)\ldots$ such that $o_0 = \lambda_O(\epsilon), i_0 = \lambda_I(o_0)$ and for all $k \geq 1, o_k = \lambda_O(o_0 i_0 \ldots o_{k-1} i_{k-1})$ and $i_k = \lambda_I(o_0 i_0 \ldots o_{k-1} i_{k-1} o_k)$. A *value* $\mathsf{Val}(u)$ is associated with each outcome $u \in \Sigma_P^\omega$ such that

$$\mathsf{Val}(u) = \begin{cases} \liminf_{n\to\infty} \frac{1}{n} \sum_{k=0}^{n-1} w(o_k) & \text{if } u \models \phi \\ -\infty & \text{otherwise} \end{cases}$$

i.e. $\mathsf{Val}(u)$ is the mean-payoff value of u if u satisfies ϕ, otherwise, it is $-\infty$. Given an LTL formula ϕ over P, a weight function w and a threshold value $\nu \in \mathbb{Z}$, the $\mathsf{LTL}_{\mathsf{MP}}$ *realizability problem* asks to decide whether there exists a strategy λ_O for Player O such that $\mathsf{Val}(\mathsf{Out}(\lambda_O, \lambda_I)) \geq \nu$ for all strategies λ_I of Player I. If the answer is Yes, ϕ is said MP-realizable. The $\mathsf{LTL}_{\mathsf{MP}}$ *synthesis problem* is then to produce such a strategy λ_O for Player O.

To illustrate the problems of $\mathsf{LTL}_{\mathsf{MP}}$ realizability and synthesis, let us consider the following specification of a server that should grant exclusive access to a resource to two clients.

Example 7 A client requests access to the resource by setting to true its request signal (r_1 for client 1 and r_2 for client 2), and the server grants those requests by setting to true the respective grant signal g_1 or g_2. We want to synthesize a server that eventually grants any client request, and that only grants one request at a time. Additionally, we ask client 2's requests to take the priority over client 1's requests. Moreover, we would like to keep minimal the delay between requests and grants. This can be formalized by the LTL formula ϕ given below where the signals in $I = \{r_1, r_2\}$ are controlled by the two clients, and the signals in $O = \{g_1, w_1, g_2, w_2\}$ are controlled by the server. Moreover, we add the following weight function $w : \mathsf{Lit}(O) \to \mathbb{Z}$:

$$\begin{aligned} \phi_1 &= \Box(r_1 \to \mathsf{X}(w_1 \mathsf{U} g_1)) \\ \phi_2 &= \Box(r_2 \to \mathsf{X}(w_2 \mathsf{U} g_2)) \\ \phi_3 &= \Box(\neg g_1 \vee \neg g_2) \\ \phi &= \phi_1 \wedge \phi_2 \wedge \phi_3 \end{aligned} \qquad w(l) = \begin{cases} -1 \text{ if } l = w_1 \\ -2 \text{ if } l = w_2 \\ 0 \text{ otherwise.} \end{cases}$$

[16] Note that in [11,12], the weight function w is more general since it also associates values to $\mathsf{Lit}(I)$. However, for this application, we restrict w to $\mathsf{Lit}(O)$.

A possible strategy for the server is to behave as follows: it grants immediately any request of client 2 if the last ungranted request of client 1 has been emitted less than n steps in the past, otherwise it grants the request of client 1. The mean-payoff value of this solution in the worst-case (when the two clients always emit their respective request) is equal to $-(1 + \frac{1}{n})$.

7.2.2 Reduction to safety games

In [11,12], we propose an antichain-based algorithm for solving the $\mathsf{LTL}_{\mathsf{MP}}$ realizability and synthesis problems with a reduction to a two-player turn-based safety game. We here present this game G without explaining the underlying reasoning, see [11] for more details. The game $G = (S_O, S_I, E, \alpha)$ is a turn-based safety game such that S_O (resp. S_I) is the set of positions of Player O (resp. Player I), E is the set of edges labeled by $o \in \Sigma_O$ (resp. $i \in \Sigma_I$) when leaving a position in S_O (resp. S_I), and $\alpha \subseteq S_O \cup S_I$ is the set of bad positions (i.e. positions that Player O must avoid to reach). Let Win_O be the set of positions in G from which Player O can force Player I to stay in $(S_O \cup S_I) \backslash \alpha$, that is the set of winning positions for Player O. The safety game G restricted to positions Win_O is a representation of a subset of the set of all winning strategies λ_O for Player O that ensure a value $\mathsf{Val}(\mathsf{Out}(\lambda_O, \lambda_I))$ greater than or equal to the given threshold ν, for all strategies λ_I of Player I. Those strategies are called *worst-case winning strategies*.

Note that the reduction to safety games given in [11,12] allows to compute the set of all worst-case winning strategies (instead of a subset of them). Indeed the proposed algorithm is incremental on two parameters $K = 0, 1, \ldots$ and $C = 0, 1, \ldots$, and works as follows. For each value of K and C, a corresponding safety game is constructed, whose number of states depends on K and C. Those safety games have the following nice property. If player O has a worst-case winning strategy in the safety game for $K = k$ and $C = c$, then he has a worst-case winning strategy in all the safety games for $K \geq k$ and $C \geq c$. The algorithm thus stops as soon as a worst-case winning strategy is found. There exist theoretical bounds $\mathbb{K}$ and $\mathbb{C}$ such that the set of all worst-case winning strategies can be represented by the safety game with parameters $\mathbb{K}$ and $\mathbb{C}$. However, $\mathbb{K}$ and $\mathbb{C}$ being huge, constructing this game is unfeasible in practice.

7.2.3 From safety games to MDPs

We can go beyond $\mathsf{LTL}_{\mathsf{MP}}$ synthesis. Let G be a safety game as above, that represents a subset of worst-case winning strategies. For each state $s \in \mathsf{Win}_O \cap S_O$, we denote by $\Sigma_{O,s} \subseteq \Sigma_O$ the set of actions that are safe to play in s (i.e. actions that force Player I to stay in Win_O). For all $s \in \mathsf{Win}_O \cap S_O$, we know that $\Sigma_{O,s} \neq \emptyset$ by construction of Win_O. From this set of worst-case winning strategies, we want to compute the one that behaves *the best against a stochastic opponent*. Let $\pi_I : \Sigma_I \to]0, 1]$ be a probability distribution on the actions of Player I. Note that we require $\mathsf{Supp}(\pi_I) = \Sigma_I$ so that it makes sense with the worst-case. By replacing Player I by π_I in the safety game G restricted to Win_O, we derive an MDP $M_G = (S, \Sigma, T, \mathbf{E}, \mathbf{D})$ where:

- $S = \mathsf{Win}_O \cap S_O$,
- $\Sigma = \Sigma_O$, and for all $s \in S$, $\Sigma_s = \Sigma_{O,s}$,
- $T = \Sigma_I$,
- $\mathbf{E}$, $\mathbf{D}$ and $\mathbf{C}$ are defined for all $s \in S$ and $\sigma \in \Sigma_s$, such that:

- for all $\tau \in T$, $\mathbf{E}(s,\sigma)(\tau) = s'$ such that $(s,\sigma,s'') , (s'',\tau,s') \in E$,
- for all $\tau \in T$, $\mathbf{D}(s,\sigma)(\tau) = \pi_I(\tau)$, and
- $\mathbf{C}(s,\sigma) = w(\sigma)$.

Note that since $\Sigma_{O,s} \neq \emptyset$ for all $s \in S$, we have that M is Σ-non-blocking.

Computing the best strategy against a stochastic opponent among the worst-case winning strategies represented by G reduces to solving the EMP problem for the MDP M_G.[17]

Lemma 6 *The MDP M_G is monotonic, and functions $\mathbf{D}$, $\mathbf{C}$ are independent from S.*

Proof It is shown in [11,12] that the safety game G has properties of monotony. The set $S_O \cup S_I$ is equipped with a partial order $\preceq$ such that $(S_O \cup S_I, \preceq)$ is a complete lattice, and the sets S_O, S_I and Win_O are closed for $\preceq$. For the MDP M_G derived from G, we thus have that $(S, \preceq)$ is a (semi)lattice, and S is closed for $\preceq$. Moreover, by construction of G (see details in [11, Sec. 5.1]) and M_G, we have that $\preceq$ is compatible with $\mathbf{E}$. By construction, $\mathbf{D}$, $\mathbf{C}$ are independent from S. □

7.2.4 Symblicit algorithm

In order to apply the pseudo-antichain-based symblicit algorithm of Sect. 6, Assumptions 1 and 2 must hold for M_G. This is the case for Assumption 2 since $\mathbf{E}(s,\sigma)(\tau)$ can be computed for all $s \in S$, $\sigma \in \Sigma_s$ and $\tau \in T$ (see [11, Sec. 5.1]), and $\mathbf{D}$ is given by π_I. Moreover, from [11, Prop. 24] and $\mathsf{Supp}(\pi_I) = \Sigma_I$, we have an algorithm for computing $\lceil \mathsf{Pre}_{\sigma,\tau}(\downarrow\{x\}) \rceil$, for all $x \in S$. So, Assumption 1 holds too. Notice also that for the MDP M_G derived from the safety game G, the symbolic representations described in Sect. 6.2 are compact, since $\mathbf{D}$ and $\mathbf{C}$ are independent from S (see Lemma 6).

Therefore for this second class of MDPs, we have again an efficient pseudo-antichain-based algorithm to solve the EMP problem, as indicated by the next experiments.

7.2.5 Experiments

We have implemented the pseudo-antichain-based symblicit algorithm for the EMP problem and integrated it into Acacia+ (v2.2) [10]. Acacia+ is a tool written in Python and C that provides an antichain-based version of the algorithm described above for solving the $\mathsf{LTL}_{\mathsf{MP}}$ realizability and synthesis problems. The last version of Acacia+ is available at http://lit2.ulb.ac.be/acaciaplus/, together with all the examples considered in this section. It can also be used directly online via a web interface. We compared our implementation with an MTBDD based symblicit algorithm implemented in PRISM [44]. To the best of our knowledge, only strategy iteration algorithms are implemented for the EMP problem. In the sequel, for the sake of simplicity, we refer to the MTBDD based implementation as PRISM and to the pseudo-antichain-based one as Acacia+. Notice that for Acacia+, the given execution times and memory consumptions only correspond to the part of the execution concerning the symblicit algorithm (and not the construction of the safety game G and the subset of worst-case winning strategies that it represents).

We compare the two implementations on a benchmark of [12] obtained from the $\mathsf{LTL}_{\mathsf{MP}}$ specification of Example 7 extended with stochastic aspects (Stochastic shared resource arbiter). For the stochastic opponent, we set a probability distribution such that requests of

[17] More precisely, it reduces to the EMP problem where the objective is to maximize the expected mean-payoff (see footnotes 1 and 3).

client 1 are more likely to happen than requests of client 2: at each turn, client 1 has probability $\frac{3}{5}$ to make a request, while client 2 has probability $\frac{1}{5}$. The probability distribution $\pi_I : \Sigma_I \to]0, 1]$ is then defined as $\pi_I(\{\neg r_1, \neg r_2\}) = \frac{8}{25}$, $\pi_I(\{r_1, \neg r_2\}) = \frac{12}{25}$, $\pi_I(\{\neg r_1, r_2\}) = \frac{2}{25}$ and $\pi_I(\{r_1, r_2\}) = \frac{3}{25}$. We use the backward algorithm of Acacia+ for solving the related safety games. The benchmark is parameterized in the threshold value ν. Results are given in Table 3. Note that the number of states in the MDPs depends on the implementation. Indeed, for PRISM, it is the number of reachable states of the MDP, denoted $|M_G^R|$, that is, the states that are really taken into account by the algorithm, while for Acacia+, it is the total number of states since unlike PRISM, our implementation does not prune unreachable states. For this application scenario, we observe that the ratio (number of reachable states)/(total number of states) is in general quite small.[18]

On this benchmark, PRISM is faster that Acacia+ on large models, but Acacia+ is more efficient regarding the memory consumption and this in spite of considering the whole state space. For instance, the last MDP of Table 3 contains more than 450 millions of states and is solved by Acacia+ in around 6.5 h with $<$100 Mo of memory, while for this example, PRISM runs out of memory. Note that the surprisingly large amount of memory consumption of both implementations on small instances is due to Python libraries loaded in memory for Acacia+, and to the JVM and the CUDD package for PRISM [32].

To fairly compare the two implementations, let us consider Fig. 6 (resp. Fig. 7) that gives a graphical representation of the execution times (resp. the memory consumption) of Acacia+ and PRISM as a function of the number of states taken into account, that is, the total number of states for Acacia+ and the number of reachable states for PRISM. For that experiment, we consider the benchmark of examples of Table 3 with four different probability distributions on Σ_I. Moreover, for each instance, we consider the two MDPs obtained with the backward and the forward algorithms of Acacia+ for solving safety games. The forward algorithm always leads to smaller MDPs. On the whole benchmark, Acacia+ times out on three instances, while PRISM runs out of memory on four of them. Note that all scales in Figs. 6 and 7 are logarithmic.

On Fig. 6, we observe that for most of the executions, Acacia+ works faster that PRISM. We also observe that Acacia+ does not behave well for a few particular executions, and that these executions all correspond to MDPs obtained from the forward algorithm of Acacia+.

Figure 7 shows that regarding the memory consumption, Acacia+ is more efficient than PRISM and it can thus solve larger MDPs (the largest MDP solved by PRISM contains half a million states while Acacia+ solves MDPs of more than 450 million states). This points out that monotonic MDPs are better handled by pseudo-antichains, which exploit the partial order on the state space, than by BDDs.

Finally, in the majority of experiments we performed for both the EMP and the SSP problems, we observe that most of the execution time of the pseudo-antichain-based symblicit algorithms is spent for lumping. It is also the case for the MTBDD based symblicit algorithm [46].

7.3 Qualitative verification of probabilistic lossy channel systems

In this section, we depart from the symblicit algorithms for the EMP and SSP problems, and move to a third application handling infinite rather than finite MPDs, and showing the variety of applications of pseudo-antichains. We show that pseudo antichains can apply to the qualitative verification for probabilistic lossy channel systems, based on results of [2].

[18] For all the MDPs considered in Tables 1 and 2, this ratio is 1.

Table 3 Expected mean-payoff on the Stochastic shared resource arbiter benchmark with two clients and increasing threshold values

ν	Acacia+								PRISM				
	$\lvert M_G \rvert$	#it	$\lvert S_{\sim_L} \rvert$	lump	LS	impr	total	mem	$\lvert M_G^R \rvert$	constr	strat	total	mem
−1.1	5259	2	22	0.12	0.01	0.02	0.15	17.4	691	0.43	0.07	0.50	168.1
−1.05	72,159	2	42	0.86	0.01	0.09	0.97	17.7	6440	1.58	0.18	1.76	249.9
−1.04	35,750	2	52	1.63	0.02	0.13	1.79	18.1	3325	1.78	0.28	2.06	264.1
−1.03	501,211	2	70	4.41	0.04	0.26	4.71	18.8	15829	4.83	0.46	5.29	277.0
−1.02	530,299	2	102	16.62	0.11	0.64	17.39	20.2	11641	6.74	0.59	7.33	343.4
−1.01	4,120,599	2	202	237.78	0.50	3.94	242.30	26.2	43891	29.91	1.61	31.52	642.5
−1.005	64,801,599	2	402	3078.88	3.07	28.19	3110.50	48.0	563,585	179.23	4.72	183.95	1629.2
−1.004	63,251,499	2	502	7357.72	5.68	52.81	7416.77	60.5	264,391	270.30	7.71	278.01	2544
−1.003	450,012,211	2	670	23,455.44	12.72	120.25	23,589.49	93.6					**MO**

The column ν gives the threshold, $\lvert M_G^R \rvert$ the number of reachable states in the MDP, and all other columns have the same meaning as in Table 1. The expected mean-payoff $\mathbb{E}_\lambda^{\mathrm{MP}}$ of the optimal strategy λ for all the examples is -0.130435

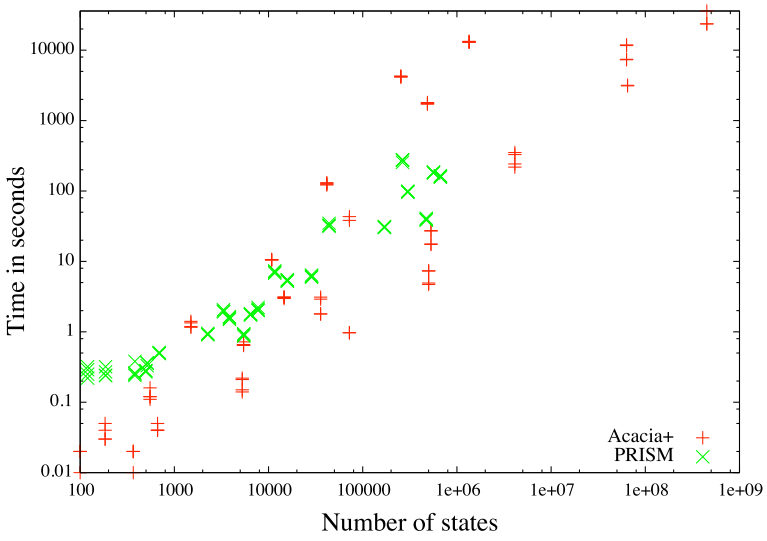

Fig. 6 Execution time

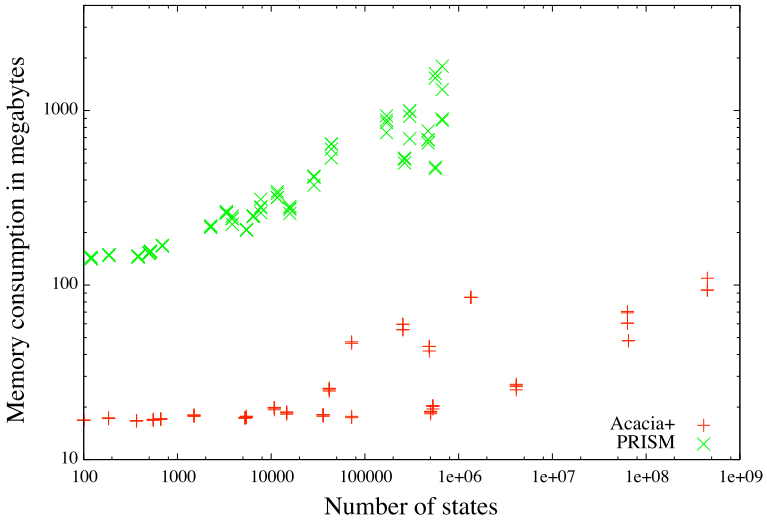

Fig. 7 Memory consumption

This paper proposes an ad hoc data structure and symbolic algorithms, and reports on a prototype applied to the verification of a simple protocol handling two-way transfers in an unreliable setting (see Pachl's protocol in Fig. 8). It appears that the data structure proposed in [2] is closely related to our pseudo-antichains. We detail in this section the differences, and explain how a simple modification of our basic algorithms on pseudo-antichains allowed us to easily reimplement the symbolic algorithms of [2]. Our implementation allowed us to verify a parameterized version of the example of Fig. 8, and we report the running time and the memory consumption on that case study.

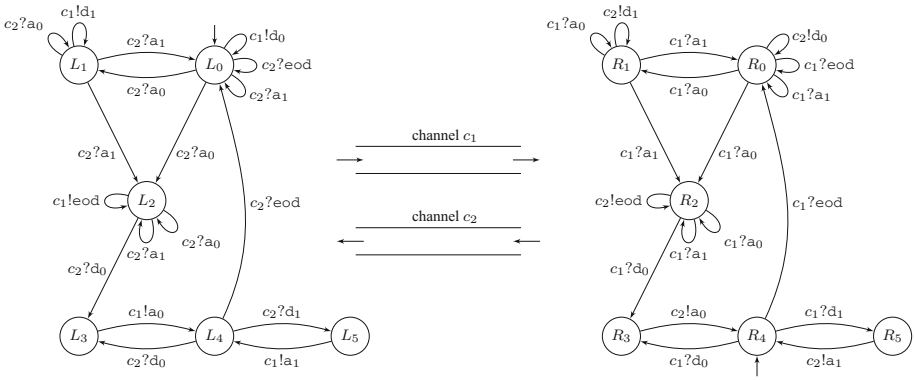

Fig. 8 Pachl's communication protocol

7.3.1 Lossy channel systems

Lossy channel systems are a classical model for asynchronous communication protocols over unreliable channels. A *lossy channel system* (*LCS*) [1,25] is a tuple (Q, C, M, Δ) where Q is a finite set of control *locations*, C is a finite set of *FIFO channels*, M is a finite *message alphabet*, and Δ is a finite set of *transition rules*. Rules can either be of the form $q \xrightarrow{c!m} p$ to send message $m \in M$ along channel $c \in C$, or $q \xrightarrow{c?m} p$ to receive message m from channel c. The semantics of an LCS is a transition system $(\mathsf{Conf}, \Delta, \rightarrow)$, where the (infinite) set of *configurations* Conf gathers all $s = (q, \mu_1, \ldots, \mu_n)$ with $q \in Q$ a location, and $(\mu_i)_{i \in C} \in M^*$ describes the channels contents. Transitions between configurations s and s' consist in the application of some rule $\delta \in \Delta$ from the LCS followed by arbitrary many message losses. The loss of messages is formalized using the *subword ordering*: $\mu \sqsubseteq \mu'$ if μ can be obtained by removing messages from μ', and $\sqsubseteq$ extends naturally to configurations. Thanks to Higman's Lemma [30], $\sqsubseteq$ is a well-quasi-order, and the transition system $(\mathsf{Conf}, \Delta, \rightarrow)$ equipped with $\sqsubseteq$ is a well-structured transition system [26].

Example 8 Pachl's protocol [38] handles two-way communications over lossy channels. It consists of two identical processes that exchange data over lossy channels using an acknowledgment mechanism based on the alternating bit protocol, and is represented in Fig. 8. The actual contents of the data messages is abstracted away, so that we use $\mathtt{d}_0, \mathtt{d}_1$ to record the alternating control bit, and $\mathtt{a}_0, \mathtt{a}_1$ are the corresponding acknowledgment messages. The protocol starts in configuration (L_0, R_4) where the left component is the sender and the right one is the receiver. At any time (provided its last data message has been acknowledged) the sender may signal the end of its data sequence by sending message $\mathtt{eod}$ and then the two processes swap their roles. Pachl's protocol is naturally modeled as an LCS: building the asynchronous product of the two processes yields an LCS with 36 locations, 2 channels and a 5-message alphabet $M = \{\mathtt{a}_0, \mathtt{a}_1, \mathtt{d}_0, \mathtt{d}_1, \mathtt{eod}\}$.

7.3.2 Probabilistic lossy channel systems

Modeling message losses by nondeterminism may not be realistic, so that variants of LCS with probabilistic losses have been proposed. A *probabilistic lossy channel system* (*PLCS*) [3] is an LCS equipped with a *fault rate* $\tau \in (0, 1)$ that specifies the probability that a given message

stored in one of the channels is lost during a step. The operational semantics of an PLCS is an infinite-state MDP $(\mathsf{Conf}, \Delta, \mathbf{P})$ on the set of configurations Conf. The value $\mathbf{P}(s, \delta, s')$ represents the probability (depending on τ) that the system moves from configuration s to configuration s' when firing transition rule δ. The formal definition of $\mathbf{P}$ is technically heavy but natural, so we omit it here. From the infinite-state MDP $(\mathsf{Conf}, \Delta, \mathbf{P})$, a strategy λ induces an infinite-state MC $(\mathsf{Conf}, \mathbf{P}_\lambda)$ and we write $\mathbb{P}^s_\lambda$ for its measure when the initial configuration is s. Here, we consider finite-memory, and even general (i.e. history dependent) strategies, contrary to the previous sections. For various types of properties φ over paths in the MDP $(\mathsf{Conf}, \Delta, \mathbf{P})$, one can consider the following qualitative verification questions: does there exist a strategy λ such that $\mathbb{P}^s_\lambda(\varphi)$ is $> 0, = 1, < 1$ or $= 0$? Fixpoint algorithms are provided in [2] to compute the set of initial configurations s from which such a strategy exists. These algorithms manipulate symbolic representations of sets of configurations described by a variant of pseudo-antichains.

7.3.3 Symbolic set of configurations

Let us briefly introduce the symbolic representations from [2] and compare them with pseudo-antichains. The variant of pseudo-elements to represent sets of channel contents has the following distinctive features. First, for $L \subseteq M^*$, one considers the upward-closure $\uparrow L$ with respect to the subword relation $\sqsubseteq$, rather than the downward-closure $\downarrow L$. Next, since $(M^*, \sqsubseteq)$ is a well-quasi-order, all antichains are finite [30], yet $(M^*, \sqsubseteq)$ is not a semilattice: the least upper bound of two configurations may be an antichain rather than a unique element. For instance, for $M = \{a, b\}$, $ab \sqcup ba = \{aba, bab\}$. This difference requires the adaptation of Proposition 2, especially (see third item) the computation of $\alpha_1 \sqcup \alpha_2 = \{a_1 \sqcup a_2 \mid a_1 \in \alpha_1, a_2 \in \alpha_2\}$ for two antichains $\alpha_1, \alpha_2 \subseteq M^*$, and can easily be done. A third difference lies in the notion of *prefixed closure* $\sigma \uparrow x$ for $x \in M^*$ and $\sigma \in M \cup \{\epsilon\}$ a message or the empty word, which allows to speak about the first message in a channel. We symbolically represent the prefixed closure $\sigma \uparrow x$ by $\sigma^\wedge x$. The variant of x (symbolically representing $\uparrow x$) is thus $\sigma^\wedge x$ (symbolically representing $\sigma \uparrow x$), and propagates to variants of antichains, and pseudo-elements, as summarized in the table below. As an example, for $M = \{a, b\}$, the variant of pseudo-element $(a^\wedge ab, \{\epsilon^\wedge b, b^\wedge aa\})$ represents the set of channel contents $a\uparrow ab \backslash (\uparrow b \cup b \uparrow aa)$.

	Classical	Variant
Element	x	$\sigma^\wedge x$
Antichain	$\alpha = \{x_i \mid i \in I\}$	$\{\sigma_i^\wedge x_i \mid i \in I\}$
Pseudo-element	(x, α)	$\theta = (\sigma^\wedge x, \{\sigma_i^\wedge x_i \mid i \in I\})$

Assuming an LCS with n channels, the symbolic representations of subsets of configurations as proposed in [2] is as follows. A *simple symbolic set* is of the form $(q, \theta_1, \ldots, \theta_n)$ where $q \in Q$ is a location, and each θ_i is a variant of pseudo-element as explained before. A *symbolic set* is a finite union of such simple symbolic elements. Any set of configurations represented by a symbolic set is called a *region*. For instance, for $M = \{a, b\}$ and $n = 1$, the simple symbolic set $(q, (a^\wedge ab, \{\epsilon^\wedge b, b^\wedge aa\}))$ represents the region $\{(q, \mu) \mid \mu \in a\uparrow ab \backslash (\uparrow b \cup b \uparrow aa)\}$. Adapting the algorithms we gave for pseudo-antichains, one can show that regions are closed under Boolean operations, as detailed in [2].

7.3.4 Qualitative verification of PLCSs

The qualitative verification of PLCSs relies on fixpoints algorithms on sets of configurations, using intersection, union, negation, and two predecessor operators. For $L \subseteq \mathsf{Conf}$ a set of configurations, $Pre(L)$ denotes the set of one-step predecessors of L, i.e. $Pre(L) = \{s \mid \exists s \xrightarrow{\delta} s' \text{ with } s' \in L\}$, and $Pre^*(L)$ is the set of predecessors of L, easily defined as a least fixpoint using Pre. The one-step constrained predecessors of L, for some $T \subseteq \mathsf{Conf}$, is defined as $cPre_T(L) = \{s \mid \exists \delta : s \xrightarrow{\delta} s' \text{ with } s' \in L, \text{and } \forall s \xrightarrow{\delta} s', s' \in T\}$. In words, s belongs to $cPre_T(L)$ if some rule δ may take from s to L while ensuring not to leave T. As for Pre, $cPre^*_T(L)$ is the least fixpoint of all constrained predecessors. Regions are closed under these predecessors operators, and from a symbolic representation for a region L, one can compute symbolic representations for $Pre(L)$, $Pre^*(L)$, $cPre_T(L)$, and $cPre^*_T(L)$ [2]. The termination of the iterative computation of least fixpoints is guaranteed because $(\mathsf{Conf}, \sqsubseteq)$ is a well-quasi-order, and applies as well to other well-structured transition systems [6].

Symbolic algorithms are proposed in [2] to compute the set of configurations s from which there exists a strategy λ such that $\mathbb{P}^s_\lambda(\varphi)$ is $> 0, = 1, < 1$ or $= 0$, for properties φ such as reachability ($\Diamond L$), invariant ($\Box L$), repeated reachability ($\Box\Diamond L$), persistence ($\Diamond\Box L$), and Streett formulas ($\bigwedge_{1 \le i \le n}(\Box\Diamond L_i \rightarrow \Box\Diamond L'_i)$). All these algorithms reduce to the symbolic computation of fixpoints using the predecessor operators $Pre(L)$ and $cPre_T(L)$, as well as Boolean connectors. For most properties, only the existence of a finite-memory strategy can be decided since the problem is undecidable when ranging over the full class of strategies. More details can be found in [2, Section 4].

7.3.5 Experiments

As explained before, although very different, the verification of PLCSs relies on fixpoints algorithms on (variants of) pseudo-antichains, and therefore shares similarities with the resolution of the stochastic shortest path problem on STRIPSs (Sect. 7.1) and the expected mean-payoff with $\mathsf{LTL}_{\mathsf{MP}}$ synthesis (Sect. 7.2). The algorithms for antichains and pseudo-antichains were easily adapted to regions in Acacia+, together with the basic algorithms for the predecessor operators $Pre(L)$ and $cPre_T(L)$. All fixpoint terms given in [2, Section 4] for verifying various types of qualitative safety and liveness properties on PLCSs have also been implemented. The source code is publicly available at http://lit2.ulb.ac.be/PLCS/, together with the benchmark presented in this section.

In order to demonstrate the scalability of our approach, the benchmark is a parameterized version of Example 8, where the number of states varies with the parameter n. The parameterized version of the left process is depicted in Fig. 9.

As a first experiment, we have conducted a *safety analysis* on the parameterized version of Pachl's protocol: it has been checked that deadlock configurations cannot be reached from the initial configuration $((L'_0, R_4), \epsilon, \epsilon)$. The approach is the following one. The set of deadlock configurations is defined as $\mathsf{Dead} = \mathsf{Conf} \backslash Pre(\mathsf{Conf})$, the set of unsafe configurations is equal to $\mathsf{Unsafe} = Pre^*(\mathsf{Dead})$, and the set of initial configurations with unsafe channel contents is the intersection $\mathsf{Unsafe} \cap ((L'_0, R_4), \uparrow\epsilon, \uparrow\epsilon)$. The computation of all these sets can be performed symbolically at the level of pseudo-antichains. This allowed us to check that the initial configuration $((L'_0, R_4), \epsilon, \epsilon)$ does not belong to Unsafe, ans thus cannot lead to a deadlock configuration. In Table 4 (column Safety), we report the running time and the memory consumption for increasing values of the parameter n.

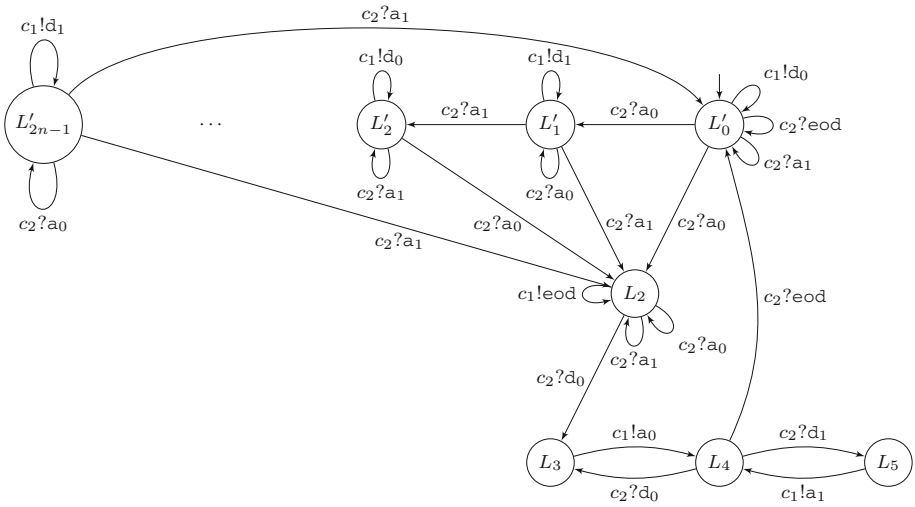

Fig. 9 Left process in the parameterized version of Pachl's protocol

Table 4 Safety and liveness analysis of the parameterized version of Pachl's protocol

n	size	Safety		Liveness	
		time	mem	time	mem
10	196	0.36	6.5	1.09	6.08
20	576	1.93	9.75	4.81	8.02
50	2916	28.88	31.48	150.06	20.03
100	10,816	447.61	107.29	2695.32	66.27
150	23,716	3288.77	236.39	17,453.49	133.62
170	30,276	5000.06	303.61	28,871.63	182.08
200	41,616	8039.76	435.25	**TO**	

The column n gives the parameter of the instance and size is the number of control states of the resulting product LCS. For both properties, *time* is the execution time in seconds, and *mem* is the memory consumption in megabytes

The second experiment is a *liveness analysis* of the parameterized protocol: it has been checked that there always exists a strategy ensuring to visit almost-surely infinitely often both (L'_0, R_4) and (L_4, R'_0). This liveness property expresses that the two parties of Pachl's protocol alternatively play the role of sender and receiver. More precisely we have symbolically computed the set of configurations s from which there exists a strategy λ such that $\mathbb{P}^s_\lambda(\Box\Diamond X \wedge \Box\Diamond X') = 1$, with $X = ((L'_0, R_4), \uparrow\epsilon, \uparrow\epsilon)$ and $X' = ((L_4, R'_0), \uparrow\epsilon, \uparrow\epsilon)$. Then, we have checked that the initial configuration $((L'_0, R_4), \epsilon, \epsilon)$ belongs to the set of these configurations s. In Table 4 (column Liveness), we report the running time and the memory consumption for increasing values of the parameter n.

This benchmark shows that our implementation could perform the safety and liveness analysis of infinite systems generated by PLCSs with thousands of control states. The size of the models is much smaller than the ones handled in the two previous case studies (stochastic shortest path and expected mean-payoff), but the properties have to be checked on the generated infinite-state MDP (whereas this MDP is finite for the SSP et EMP problems), and the fixpoint computation for PLCSs runs in non-primitive recursive time in the size of the model [3].

8 Conclusion

In this paper, we have presented the interesting class of monotonic MDPs, and the new data structure of pseudo-antichains. We have shown how monotonic MDPs can be exploited by symblicit algorithms using pseudo-antichains (instead of MTBDDs) for two quantitative settings: the expected mean-payoff and the stochastic shortest path. Those algorithms have been implemented, and we have reported promising experimental results for two applications coming from automated planning and $\mathsf{LTL}_{\mathsf{MP}}$ synthesis. We have also proposed a third application dealing with infinite MDPs in the context of PLCSs. We are convinced that pseudo-antichains can be used in the design of efficient algorithms in other contexts like for instance model-checking or synthesis of non-stochastic models, as soon as a natural partial order can be exploited. It could be interesting to have additional applications of pseudo-antichains in a way to better understand their performance with respect to the time and memory consumption, and how the compactness of the representations by pseudo-antichains influences the efficiency of the algorithms.

Acknowledgements We would like to thank Mickael Randour for his fruitful discussions, Marta Kwiatkowska, David Parker and Christian Von Essen for their help regarding the tool PRISM, and Holger Hermanns and Ernst Moritz Hahn for sharing with us their prototypical implementation. This work has been partly supported by ERC Starting Grant (279499: inVEST), ARC project (Number AUWB-2010-10/15-UMONS-3), European project Cassting (FP7-ICT-601148), and an F.R.S.-FNRS grant "Mission Scientifique".

References

1. Abdulla, P.A., Jonsson, B.: Verifying programs with unreliable channels. Inf. Comput. **127**(2), 91–101 (1996)
2. Baier, C., Bertrand, N., Schnoebelen, P.: Symbolic verification of communicating systems with probabilistic message losses: liveness and fairness. In: Najm, E., Pradat-Peyre, J., Donzeau-Gouge, V. (eds.) FORTE, volume 4229 of Lecture Notes in Computer Science, pp. 212–227. Springer (2006)
3. Baier, C., Bertrand, N., Schnoebelen, P.: Verifying nondeterministic probabilistic channel systems against ω-regular linear-time properties. ACM Trans. Comput. Log. **9**(1), Article No. 5 (2007)
4. Baier, C.: Principles of model checking. MIT Press, Cambridge (2008)
5. Baier, C., Katoen, J.-P., Hermanns, H., Wolf, V.: Comparative branching-time semantics for Markov chains. Inf. Comput. **200**(2), 149–214 (2005)
6. Bertrand, N., Schnoebelen, P.: Computable fixpoints in well-structured symbolic model checking. Form. Methods Syst. Des. **43**(2), 233–267 (2013)
7. Bertsekas, D.P., Tsitsiklis, J.: Neuro-dynamic programming. Athena Scientific, Anthropological Field Studies, Belmont (1996)
8. Bertsekas, D.P., Tsitsiklis, J.N.: An analysis of stochastic shortest path problems. Math. Oper. Res. **16**(3), 580–595 (1991)
9. Blum, A.L., Langford, J.C.: Probabilistic planning in the graphplan framework. In: Biundo, S., Fox, M. (eds.) Recent Advances in AI Planning, pp. 319–332. Springer (2000)
10. Bohy, A., Bruyère, V., Filiot, E., Jin, N., Raskin, J.-F.: Acacia+, a tool for LTL synthesis. In: Madhusudan, P., Seshia, S.A. (eds.) CAV, volume 7358 of Lecture Notes in Computer Science, pp. 652–657. Springer (2012)
11. Bohy, A., Bruyère, V., Filiot, E., Raskin, J.-F.: Synthesis from LTL specifications with mean-payoff objectives. *CoRR*, abs/1210.3539 (2012)
12. Bohy, A., Bruyère, V., Filiot, E., Raskin, J.-F.: Synthesis from LTL specifications with mean-payoff objectives. In: Piterman, N., Smolka, S.A. (eds.) TACAS, volume 7795 of Lecture Notes in Computer Science, pp. 169–184. Springer (2013)
13. Bohy, A., Bruyère, V., Raskin, J.: Symblicit algorithms for optimal strategy synthesis in monotonic markov decision processes. In: Chatterjee, K., Ehlers, R., Jha, S. (eds.) SYNT, volume 157 of EPTCS, pp. 51–67 (2014)
14. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Trans. Comput. **35**(8), 677–691 (1986)

15. Buchholz, P.: Exact and ordinary lumpability in finite Markov chains. J. Appl. Probab. **31**(1), 59–75 (1994)
16. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, L.J.: Symbolic model checking: 10^{20} states and beyond. Inf. Comput. **98**(2), 142–170 (1992)
17. Chatterjee, K., Henzinger, T.A., Jobstmann, B., Singh, R.: In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS, volume 6605 of Lecture Notes in Computer Science. Lecture Notes in Computer Science, pp. 267–271. Springer (2011)
18. Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching-time temporal logic. In: Kozen, D. (ed.) Logic of Programs, volume 131 of Lecture Notes in Computer Science, pp. 52–71. Springer (1981)
19. de Alfaro, L.: Computing minimum and maximum reachability times in probabilistic systems. In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR, volume 1664 of Lecture Notes in Computer Science, pp. 66–81. Springer (1999)
20. Derisavi, S., Hermanns, H., Sanders, W.H.: Optimal state-space lumping in Markov chains. Inf. Process. Lett. **87**(6), 309–315 (2003)
21. Doyen, L., Raskin, J.-F.: Improved algorithms for the automata-based approach to model-checking. In: Grumberg, O., Huth, M. (eds.) TACAS, volume 4424 of Lecture Notes in Computer Science, pp. 451–465. Springer (2007)
22. Fikes, R.E., Nilsson, N.J.: STRIPS:a new approach to the application of theorem proving to problem solving. Artif. Intell. **2**(3), 189–208 (1972)
23. Filar, J., Vrieze, K.: Competitive Markov Decision Processes. Springer, Berlin (1997)
24. Filiot, E., Jin, N., Raskin, J.-F.: Antichains and compositional algorithms for LTL synthesis. Form. Methods Syst. Des. **39**(3), 261–296 (2011)
25. Finkel, A.: Decidability of the termination problem for completely specified protocols. Distrib. Comput. **7**(3), 129–135 (1994)
26. Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere!. Theor. Comput. Sci. **256**(1–2), 63–92 (2001)
27. Fujita, M., McGeer, P.C., Yang, J.C.-Y.: Multi-terminal binary decision diagrams: an efficient data structure for matrix representation. Form. Methods Syst. Des. **10**(2/3), 149–169 (1997)
28. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Form. Asp. Comput. **6**(5), 512–535 (1994)
29. Hartmanns, A.: Modest: a unified language for quantitative models. In: FDL, IEEE, pp. 44–51 (2012)
30. Higman, G.: Ordering by divisibility in abstract algebras. Proc. Lond. Math. Soc. **3**(2), 326–336 (1952)
31. Howard, R.A.: Dynamic Programming and Markov Processes. Wiley, New Jersey (1960)
32. Jansen, D.N., Katoen, J.-P., Oldenkamp, M., Stoelinga, M., Zapreev, I.S.: How fast and fat is your probabilistic model checker? an experimental performance comparison. In: Yorav, K. (ed.) Haifa Verification Conference, volume 4899 of Lecture Notes in Computer Science, pp. 69–85. Springer (2007)
33. Katoen, J.-P., Zapreev, I.S., Hahn, E.M., Hermanns, H., Jansen, D.N.: The ins and outs of the probabilistic model checker MRMC. Perform. Eval. **68**(2), 90–104 (2011)
34. Kemeny, J.G., Snell, J.L.: Finite Markov Chains. Van Nostrand Company, Inc, New York (1960)
35. Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV, volume 6806 of Lecture Notes in Computer Science, pp. 585–591. Springer (2011)
36. Larsen, K.G., Skou, A.: Bisimulation through probabilistic testing. Inf. Comput. **94**(1), 1–28 (1991)
37. Majercik, S.M., Littman, M.L.: Maxplan: a new approach to probabilistic planning. In: Simmons, R.G., Veloso, M.M., Smith, S.F. (eds.) AIPS, pp. 86–93. AAAI, Palo Alto (1998)
38. Pachl, J.K.: Protocol description and analysis based on a state transition model with channel expressions. In: Rudin, H., West, C.H. (eds.) PSTV, Proceedings of the IFIP WG6.1, pp. 207–219. North-Holland (1987)
39. Paige, R., Tarjan, R.E.: Three partition refinement algorithms. SIAM J. Comput. **16**(6), 973–989 (1987)
40. Parker, D.: Personal communication, 2013-11-20
41. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley, New Jersey (1994)
42. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice Hall, Englewood Cliffs (1995)
43. Veinott, A.F.: On finding optimal policies in discrete dynamic programming with no discounting. Ann. Math. Stat. **37**(5), 1284–1294 (1966)
44. Von Essen, C.: Personal communication, 2013-11-20
45. Von Essen, C., Jobstmann, B.: Synthesizing efficient controllers. In: Kuncak, V., Rybalchenko, A. (eds.) VMCAI, volume 7148 of Lecture Notes in Computer Science, pp. 428–444. Springer (2012)

46. Wimmer, R., Braitling, B., Becker, B., Hahn, E.M., Crouzen, P., Hermanns, H., Dhama, A., Theel, O.E.: Symblicit calculation of long-run averages for concurrent probabilistic systems. In: QEST, IEEE Computer Society, pp. 27–36 (2010)
47. Wulf, M.D., Doyen, L., Henzinger, T.A., Raskin, J.-F.: Antichains: a new algorithm for checking universality of finite automata. In: Ball, T., Jones, R.B. (eds.) CAV, volume 4144 of Lecture Notes in Computer Science, pp. 17–30. Springer (2006)