Santos *et al. Journal of the Brazilian Computer Society* (2015) 21:5
DOI 10.1186/s13173-015-0024-1

Journal of the Brazilian Computer Society
a SpringerOpen Journal

RESEARCH Open Access

Reachability-based model reduction for Markov decision process

Felipe M Santos*, Leliane N Barros and Felipe W Trevizan

Abstract

This paper presents how to improve model reduction for Markov decision process (MDP), a technique that generates equivalent MDPs that can be smaller than the original MDP. In order to improve the current state-of-the-art, we take advantage of the information about the initial state of the environment. Given this initial state information, we perform a reachability analysis and then employ model reduction techniques to the reachable space of the original problem. Further, we also eliminate redundancies in the original MDP in order to speed up the model reduction phase. We also contribute by empirically comparing our technique against state-of-the-art model reduction techniques and MDP solvers that do not perform model reduction. The results show that our approach dominates the current model reduction algorithms and outperforms general MDP solvers in *dense* problems, i.e., problems in which actions have many probabilistic outcomes.

Keywords: Probabilistic planning; Markov decision process; Model reduction; Stochastic bisimulation

Background

One of the biggest challenges in the probabilistic planning is to solve large *Markov decision processes (MDPs)* [1]. This is because the number of states in an MDP grows exponentially with the number of state variables, a problem known as the *Bellman's curse of dimensionality* [1]. Many techniques have been proposed to avoid the complete enumeration of states, e.g., by exploiting the structure of factored models [2, 3] and by using the information of the initial state to find the states relevant for the optimal solution by focusing on them [4, 5].

Another approach is to apply *model reduction* to obtain a smaller MDP and then solve it using an off-the-shelf MDP solver [6]. In order to find the optimal solution for the original MDP, both the original and reduced MDP must be equivalent. Therefore, the problem of finding an equivalent and reduced MDP consists of computing a partition $\mathcal{P}$ of the original MDP state space S, such that each block represents a subset $B_i \subseteq S$ that groups equivalent states according to their reward and probabilistic transition functions.

*Correspondence: fmsantos@ime.usp.br
Institute of Mathematics and Statistics - University of São Paulo, Rua do Matão, 1010 São Paulo, Brazil

This paper extends the algorithms for computing stochastic bisimulation in two directions: *reachability analysis* and *partition elimination*. In the former, we use the MDP's initial state to find all the states relevant to the optimal solution and consider only this subspace of the original problem when applying model reduction. In the latter, we detect and delete intermediary partitions of the original MDP that are repeated, thus speeding up the convergence to a stochastic bisimulation. We also contribute with an empirical comparison among: the state-of-the-art model reduction algorithms and MDP solvers (that do not perform model reduction). The results show that our approach dominates the current model reduction algorithms in all problems, specially in *sparse* domain problems where reachability analysis prunes a significant part of the state space. The experiments also show that our technique outperforms general MDPs solvers (that do not perform model reduction) in *dense* problems, i.e., problems in which actions have many probabilistic outcomes. This is true specially when the domain suffer a significant reduction. This happens because stochastic bisimulations with *partition elimination* can summarize the dynamics of the domain and generate equivalent problems with half of the original MDP size.

The paper is organized as follows. The "Background" section presents our notation, background related to

MDPs, and algorithms to solve them. The "Aggregation algorithms" section presents the concepts of state aggregation algorithms and stochastic bisimulation, as well as the basic algorithm to compute them. The "Stochastic bisimulation over the reachable states" section contains our contributions to algorithms that compute stochastic bisimulations. The section "Results and discussion" empirically compares our technique combined with traditional planners against traditional model reduction algorithms and planners that do not employ model reduction. Finally, the "Conclusions" section presents our conclusions.

Markov decision processes

An infinite-horizon MDP [1] is a tuple $M = (S, A, P, R, \gamma)$, where:

- S is a finite set of states that can be observed in different moments in time;
- A is a finite set of actions and $A(s) \subseteq A$ is the set of applicable actions in the state s;
- $P : S \times A \times S$ is the transition function and is given by $P(s'|s,a)$ that defines the probability to reach $s' \in S$ after applying an action $a \in A$ in $s \in S$. $P(\cdot|\cdot,a)$ defines a probabilistic transition matrix where each row represents a state s, each column represents a state s' and an entry (s,s') has the probability $P(s'|s,a)$. We say that the transition matrix is *dense* if at least 50 % of the entries have probabilities greater than 0; otherwise, we called it a *sparse* transition matrix;
- $R : S \times A$ is the reward function that represents the reward received after applying an action $a \in A$ in the state $s \in S$; and
- $\gamma \in]0,1[$ is the discount factor used for weighting future rewards [1].

A solution of an MDP is a *policy* $\pi : S \mapsto A$ that maps each state $s \in S$ to an action $a \in A$.

The expected accumulated reward obtained when following a policy π from a state s is represented as $V^\pi(s)$ and is the fixed-point solution for the following system of equations:

$$V^\pi(s) = R(s,\pi(s)) + \gamma \sum_{s' \in S} P(s'|s,\pi(s)) V^\pi(s'), \qquad \forall s \in S. \tag{1}$$

A policy π^* is optimal if, for every policy π' and $s \in S$, $V^{\pi'}(s) \leq V^{\pi^*}(s)$. Notice that π^* might not be unique; however, the optimal expected accumulated reward from a state s, denoted by $V^*(s)$, is unique [7]. The value function V^* can be computed directly by finding the fixed-point solution of the *Bellman equations*, i.e., the following system of equations:

$$V^*(s) = \max_{a \in A} \left\{ R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^*(s') \right\}, \quad \forall s \in S. \tag{2}$$

An MDP $M = (S, A, P, R, \gamma)$ is enumerative if all elements of all sets that constitute the MDP are explicitly enumerated, e.g., state space S and probability tables P are represented directly by enumerating each element of them. Alternatively, an MDP can be represented in a factored form based on a set of state variables $X = \{X_1, \ldots, X_n\}$. Each state is represented as a vector $\vec{x} = (x_1, \ldots, x_n)$ of assignments where each x_i is either 0 or 1 to denote if the state variable X_i is active or not. Thus, the size of the set of states in a factored MDP is 2^n.

The transition function of a factored MDP is represented by a set of *dynamic Bayesian networks (DBNs)* [8], one for each action. A DBN for an action a is an acyclic directed graph that has the following two layers: (i) a layer representing the set X of state variables in the current state and (ii) a layer representing the set X' of state variables in the next state. Every arc in a DBN representing an action is from the layer X to X' and represents the dependencies between state variables under an action a. Given a variable X'_j, the *parents* of X'_j (denoted by parents(X'_j)) are all variable X_i such that there exists an arc from X_i to X'_j.

A DBN also contains the *conditional probability tables (CPTs)* that give us the probability of a state variable X'_j being true or false given the parents of X'_j. The advantage of using DBNs is that we do not need to enumerate all possible combinations of state variable values to represent the transition function. Instead, it is obtained as follows:

$$P(\vec{x}'|\vec{x},a) = \prod_{i=1}^{n} P(x'_i|\text{parents}(X'_j),a). \tag{3}$$

An efficient way to represent the CPTs (and factored reward functions) is through *algebraic decision diagram (ADD)* [9]. ADDs extend *binary decision diagrams (BDDs)* [10]. BDDs are decision trees represented in a more compact way in order to efficiently define functions with binary variables to a binary result, i.e., $f : \mathbb{B}^n \mapsto \mathbb{B}$. ADDs are used to represent functions that map binary variables to real values, i.e., $f : \mathbb{B}^n \mapsto \mathbb{R}$. Thus, to solve an MDP, we can also represent the value function as an ADD. There are ADD libraries to efficiently compute operations such as addition ($\oplus$), multiplication ($\otimes$), and marginalization $\left(\sum_{x_i \in X_i}\right)$.

Algorithms for solving MDPs

Value iteration and topological value iteration

The *value iteration (VI)* algorithm [1] uses dynamic programming in order to find V^*. Formally, VI solves the following recursive equations, where t is the number of stages-to-go:

$$V^t(s) = \begin{cases} \text{any value} & \text{if } t = 0 \\ \max_{a \in A} \left\{ R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^{t-1}(s') \right\} & \text{otherwise.} \end{cases} \tag{4}$$

The algorithm converges to V^* when the maximum error between the last two iterations is less than a small constant ϵ, for all state $s \in S$. This is expressed as:

$$\max_{s \in S} |V^t(s) - V^{t-1}(s)| \leq \epsilon. \tag{5}$$

VI can take a long time to converge because it needs to update the values of the complete set of states S in each iteration independently of the problem structure.

Topological value iteration (TVI) [11], an extension of VI, exploits the topological structure of the transition graph to speed up the convergence time by decreasing the number of updates performed.

Formally, TVI pre-processes the given MDP by performing a topological analysis of the graph representing the MDP, i.e., a graph in which the nodes are states and the arcs are actions. The result of this analysis is a set of the *strongly connected components (SCCs)* and TVI applies VI on each SCC in reversed topological order. This decomposition can speed up the convergence to V^* when the original MDP can be decomposed into several SCCs with similar size. In the worst case, when the MDP has only one SCC, TVI performs worst than VI due to the overhead of the topological analysis.

Labeled real-time dynamic programming

Stochastic shortest path problem (SSP) [7] is another model for probabilistic planning. The main differences between SSPs and infinite-horizon MDPs is that SSPs contain the information about the initial state of the environment as well as a set of goal states representing the stop criterion of the agent. Formally, an SSP is a tuple (S, s_0, G, A, P, C) where:

- as in the MDPs, S, A, and P are the set of states, set of actions, and transition function respectively;
- $s_0 \in S$ is the initial state;
- $G \subset S$ is the non-empty set of goal states; and
- $C : S \times A$ represents the cost of applying action $a \in A$ in the state $s \in S$.

SSPs are relevant for this work because any infinite-horizon MDP can be perfectly represented as an SSP [12, Section 7.3], thus we can also use algorithms developed for SSPs to solve the problems presented in this paper.

One example of algorithm for solving SSPs is *real-time dynamic programming* (RTDP) [4] that, given an SSP and a lower bound for V^*, computes an optimal policy for all the *relevant states*, i.e., all states reachable by the optimal policy starting from s_0. RTDP starts from s_0 and visits the set of states following a greedy policy until a goal state is reached. This procedure is known as a trial and, for the case of infinite-horizon MDPs converted to SSPs, a trial can be seen as an infinite process that randomly finishes with probability $1 - \gamma$ every time a state is visited. RTDP performs trials until it has converged to the optimal solution.

In each state s visited in a trial, RTDP computes the greedy policy in s (the action that maximizes Equation 4), updates $V^t(s)$, and samples a next state to be visited based on the probability distribution of the greedy action. Due to this sampling procedure, states with a low probability are updated less often than states with a higher probability, resulting in overall lower convergence.

Labeled RTDP (LRTDP) [5] is an algorithm that enhances RTDP providing a faster convergence of the optimal solution. This performance improvement is obtained by labeling the states that have already converged and finishes the trials when a converged state or goal is reached. LRTDP uses the procedure *CheckSolved* [5] that is responsible to decide if a state is converged or not. This decision is done based on the concept of *greedy graph of a state s*, i.e., the graph that contains all reachable states from s following the greedy policy. The optimal solution is obtained when all states in the greedy graph of s_0 are marked as converged.

Aggregation algorithms

Given an MDP M, an optimal policy for M can be found by using VI, TVI, or LRTDP. Another way to find an optimal policy is based on two steps: (1) getting an equivalent model of reduced size based on the original MDP and (2) solving the reduced model and applying the solution in the original MDP. The first step is known as *state aggregation for MDPs* [13].

Mathematically, state aggregation algorithms for MDPs receive as input an MDP $M = (S, A, P, R, \gamma)$ and computes a reduced MDP $M' = (S', A, P', R', \gamma)$ with a set of states S' that groups states into blocks of states, called *abstract states* in M'. It is desirable that: the set of states of M' be much smaller than the set of states of M, i.e., $|S'| \ll |S|$ and an optimal solution for M' to also be optimal for M.

There are many techniques for MDPs in the state aggregation category. For example:

- *Stochastic bisimulations (exact/approximate)* [6, 14], a technique that receives an MDP (factored or enumerative) and returns an enumerative MDP (or a *bounded-parameter MDP* [15], i.e., an MDP whose functions are given by intervals);
- *Homomorphisms (exact/approximate)* [16, 17], techniques that are similar to stochastic bisimulations, but can achieve greater reductions in some special scenarios;
- *Structured policy iteration (SPI)* [18], one of the first techniques to use decision trees to solve factored MDPs;
- *Stochastic planning using decision diagrams (SPUDD)* [2], a technique that is a factored version of value iteration. This was the first algorithm that used ADDs to solve MDPs more efficiently; and
- *Symbolic real-time dynamic programming - sRTDP* [3], a factored version of RTDP that also uses decision diagrams.

A complete overview about state aggregation algorithms is presented in [13]. Some of these algorithms are completely factored, that is, they never use the concept of enumerative states set. At the same time, others can combine factored and enumerative representations. We chose exact stochastic bisimulations because with them, we can easily compare time and reduction performance against exact enumerative MDP planners. Moreover, we do not need to define a criterion to compare different approximate solutions because the exact solutions are unique. Thus, from now on, we will consider that model reduction and model minimization are algorithms that compute exact stochastic bisimulations.

The state aggregation algorithms for MDPs usually work based on the concept of partitions. A *partition* of a set S is a set of disjoint subsets whose union is S itself. Each disjoint subset can also be called a *block*. Given an enumerative MDP M and a set of states S, a partition over S is given by $\mathcal{P} = \{B_1, \ldots, B_k\}$, where $\mathcal{P}$ is a partition of S and each $B_i \in \{1, \ldots, k\}$ is a block.

Two important concepts for model reduction are *refinement* and *coarsening* of a partition. A partition $\mathcal{P}'$ is a refinement of a partition $\mathcal{P}$ if and only if each block of $\mathcal{P}'$ is a subset of some block in $\mathcal{P}$, i.e., a refinement splits a block B_i in sub-blocks generating a finer partition. If $\mathcal{P}' = \mathcal{P}$, $\mathcal{P}'$ still is a refinement of $\mathcal{P}$. The concept of coarsening is opposite to the concept of refinement: if $\mathcal{P}'$ is a refinement of $\mathcal{P}$, $\mathcal{P}$ is a coarsening of $\mathcal{P}'$ [6].

Definition 1. Given that $\mathcal{P}_1$ and $\mathcal{P}_2$ are partitions of S described by enumerative states, we can generate a partition $\mathcal{P}_3$ that is a refinement of $\mathcal{P}_1$ and $\mathcal{P}_2$ with the intersection of them, i.e., $\mathcal{P}_3 = \mathcal{P}_1 \cap \mathcal{P}_2$ such that each $B_k \in \mathcal{P}_3$ is computed by the intersection of two blocks $B_i \in \mathcal{P}_1$ and $B_j \in \mathcal{P}_2$, with $B_i \cap B_j \neq \emptyset$. To compute every $B_k \in \mathcal{P}_3$, it is necessary to do all combinations of $B_i \cap B_j$.

If we use factored representation instead of enumerative representation, it is possible to get more efficient algorithms for performing model reduction [6]. Hence, let $X = \{X_1, \ldots, X_n\}$ be the set of state variables of a given factored MDP and $S \subseteq 2^n$ a set of valid states of this MDP. A block of states can be characterized by a *disjunctive normal form (DNF)* expression over the boolean variables in X, i.e., B_i can be represented as a boolean formula and we use a label v_i to identify the block. Thus, a *labeled partition* of S is a set $\mathcal{P} = \{(B_1, v_1), \ldots, (B_k, v_k)\}$ such that $(\bigcup_{i=1}^{k} B_i) = S$. Furthermore, as $\mathcal{P}$ is a an augmented partition, each pair v_i, v_j associated with blocks B_i, B_j must be different and unique in the partition. Each $(B_i, v_i) \in \mathcal{P}$ is a tuple with a partition block B_i and an unique label v_i that is common to all $s \in B_i$. For instance, $\mathcal{P} = \{(X_1, 2), (\neg X_1, 3)\}$ is a labeled partition with two blocks: a block of states, labeled with 2, where X_1 is satisfied and a block of states, labeled with 3, where $\neg X_1$ is satisfied.

Definition 2. Given that $\mathcal{P}_1$ and $\mathcal{P}_2$ are labeled partitions of S described by DNF expressions, we can generate a partition $\mathcal{P}_3$ that is a refinement of $\mathcal{P}_1$ and $\mathcal{P}_2$ with the intersection of them, i.e., $\mathcal{P}_3 = \mathcal{P}_1 \cap \mathcal{P}_2$ where $B_k \in \mathcal{P}_3$ is computed by the conjunction ($\wedge$) of two blocks $B_i \in \mathcal{P}_1$, $B_j \in \mathcal{P}_2$ such that B_i and B_j are not mutually exclusive DNF expressions of the kind $X_1 \wedge \neg X_1$. To compute every $B_k \in \mathcal{P}_3$, it is necessary to compute all combinations of conjunctions considering different possibilities of $B_i \cap B_j$. The label v_k must be different from the labels v_i and v_j. By definition, a partition of S with a single block is represented by the boolean expression *true*.

For example, given the partitions (Fig. 1) $\mathcal{P}_1 = \{(X_1, 2), (\neg X_1, 3)\}$ and $\mathcal{P}_2 = \{(X_2 \wedge X_3, 5), (X_2 \wedge \neg X_3, 7), (\neg X_2, 11)\}$, their intersection will result in the refined partition $\mathcal{P}_3 = \{(X_1 \wedge X_2 \wedge X_3, 10), (X_1 \wedge X_2 \wedge \neg X_3, 14), (X_1 \wedge \neg X_2, 22), (\neg X_1 \wedge X_2 \wedge X_3, 15), (\neg X_1 \wedge X_2 \wedge \neg X_3, 21), (\neg X_1 \wedge \neg X_2, 33)\}$. In general, given m partitions $\mathcal{P}_1, \ldots, \mathcal{P}_m$, it is possible to get a refinement by computing the intersection of them as $\mathcal{P}' = \bigcap_{i=1}^{m} \mathcal{P}_i$ [6].

Figure 2 shows a sequence of refinements. In the beginning of the first iteration, it is given a partition with a single block. After the first iteration of refinements, the partition is split into two blocks. In the next two iterations, we get partitions of two and three blocks, respectively. Finally, a new refinement is done and the partition does not change. Hence, it is not necessary to split more blocks. From this example, we could extract an MDP M' with $|S'| = 5$.

Given a factored MDP, it is possible to use its reward and transition functions to identify blocks of states with

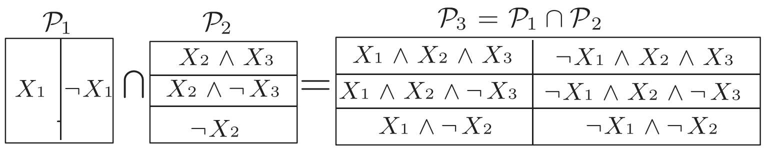
$\mathcal{P}_1$
$\mathcal{P}_2$
$\mathcal{P}_3 = \mathcal{P}_1 \cap \mathcal{P}_2$
X_1
$\neg X_1$
$X_2 \wedge X_3$
$X_2 \wedge \neg X_3$
$\neg X_2$
$X_1 \wedge X_2 \wedge X_3$
$\neg X_1 \wedge X_2 \wedge X_3$
$X_1 \wedge X_2 \wedge \neg X_3$
$\neg X_1 \wedge X_2 \wedge \neg X_3$
$X_1 \wedge \neg X_2$
$\neg X_1 \wedge \neg X_2$

Fig. 1 Refinement of partitions. Refinement of partitions $\mathcal{P}_1$ with two blocks and $\mathcal{P}_2$, with three blocks. The refinement result is given by $\mathcal{P}_1 \cap \mathcal{P}_2$, a partition with six blocks

the same reward or transition probability. For example, consider the domain *Sysadmin*, where we have n computers that must be running. Given a problem (instance) of this domain, we can apply the actions rebootCi or noop at each stage of the MDP. In an instance with two computers: $C1$ and $C2$, we have two state variables: running$C1$ and running$C2$ and three actions: noop, reboot$C1$ and reboot$C2$. The reward function for the action noop, given as an ADD, can be viewed in Fig. 3. For instance, if we execute the noop action in the state where running$C1$ is true and running$C2$ is false, the reward is 1. To refer to the partitions of an MDP, we use a special notation: for each action $a \in A$, we have a partition $\mathcal{P}_R^a$ with respect to the reward function; and for each pair of action $a \in A$ and state variable $X_i \in X$ that can be changed by a, we have a partition $\mathcal{P}_{X_i}^a$ with respect to the factored probabilistic transition function.

The partition obtained from the reward function in Fig. 4 is implicitly represented as $\mathcal{P}_R^{\text{noop}} = \{(B_1, v_1), (B_2, v_2), (B_3, v_3)\}$, where: $B_1 = \{\text{running}C1 \wedge \text{running}C2\}$, with reward 2; $B_2 = \{(\neg\text{running}C1 \wedge \text{running}C2) \vee (\text{running}C1 \wedge \neg\text{running}C2)\}$, with reward 1; and $B_3 = \{\neg\text{running}C1 \wedge \neg\text{running}C2\}$, with reward 0. We can do the same for the other actions generating the partitions: $P_R^{\text{reboot}C1}$ and $P_R^{\text{reboot}C2}$.

The partitions obtained from the factored probabilistic transition functions, for the *Sysadmin* example, for each pair of action and state variable are: $\mathcal{P}_{\text{running}C1}^{\text{noop}}$, $\mathcal{P}_{\text{running}C2}^{\text{noop}}$, $\mathcal{P}_{\text{running}C1}^{\text{reboot}C1}$, $\mathcal{P}_{\text{running}C2}^{\text{reboot}C1}$, $\mathcal{P}_{\text{running}C1}^{\text{reboot}C2}$ and $\mathcal{P}_{\text{running}C2}^{\text{reboot}C2}$. Thus, in general, the maximum number of different partitions we can have is $|A| + (|A| \times |X|)$, where $|A|$ comes from the reward function of each action and $|A| \times |X|$ comes from the factored probabilistic transition functions considering each pair of action and state variable.

A labeled partition can be represented using an ADD where each leaf represents an unique label v_i. The DNF expression that characterizes a block of states B_i is given by the disjunction of the conjunctions among state variables, obtained from different paths that go from the ADD root to the leaf labeled with v_i. For instance, Fig. 5 shows an ADD that represents the following labeled partition:

$$\mathcal{P} = \{(X_1 \wedge X_2, 2), (X_1 \wedge \neg X_2, 3) \vee (\neg X_1 \wedge \neg X_2, 3), \\ (\neg X_1 \wedge X_2, 5)\}$$

To compute the refinement of q labeled partitions represented as ADDs, we need to get the product of ADDs representing these partitions. Note that performing the product of ADDs representing partitions is the same of computing the intersection of the partitions [6]. In this case, we need to define unique labels for them that will also be unique after the product is computed. For this purpose, the blocks are labeled with prime numbers.

Figure 4 shows the ADD that represents the reward function for the action *noop* in the *Sysadmin* example that corresponds to the tabular representation in Fig. 3. We can make a partition based on this ADD by creating a copy of it and changing the leaf values to distinct prime numbers as in Fig. 6. Figure 7 shows an example of partition refinement showed in Fig. 1 using ADDs. To refer to the partitions represented with ADDs, we use the following notation: $\mathcal{P}_R^a$ is denoted by $\mathcal{P}_{DD}^{a,R}$ and $\mathcal{P}_{X_i}^a$ is denoted by $\mathcal{P}_{DD}^{a,X_i}$.

Stochastic bisimulation concepts

This section presents a technique that receives an MDP as input and computes an enumerative MDP of reduced size by searching for a *stochastic bisimulation* [6], i.e., a partition in which states in a same block have the same behavior under any action in the MDP.

Thus, these states can be considered equal because the partition can be seen as an equivalence relation [6].

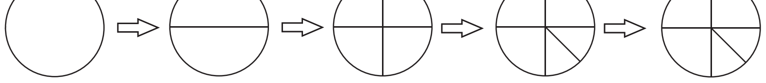

Fig. 2 A sequence of refinements. A partition with one block is refined until it reaches five blocks. Note that each refinement imply in a new partition that have at least the same number of blocks

runningC1	runningC2	reward
0	0	0
0	1	1
1	0	1
1	1	2

Fig. 3 Reward function as a table. Reward function for action noop in an instance of the domain SysAdmin with 2 computers

Definition 3. Let $\mathcal{P}$ be a partition of S. We say that $\mathcal{P}$ is a uniform partition with respect to the reward function if, for each $B_i \in \mathcal{P}$, $s, s' \in B_i$, and $a \in A$, we have that $R(s,a) = R(s',a)$ [6].

Definition 4. Given two blocks $B_j \in \mathcal{P}$ and $B_w \in \mathcal{P}$, if the following holds for all $s \in B_j$, $s' \in B_j$, and $a \in A$

$$\sum_{s'' \in B_w} P(s''|s,a) = \sum_{s'' \in B_w} P(s''|s',a),$$

for all $s \in B_j$, $s' \in B_j$, and $a \in A$. Then, we say that block B_j is stable with respect to B_w [6].

Definition 5. A block B_j is stable if it is stable with respect to all $B_w \in \mathcal{P}$ [6].

Definition 6. (Stochastic bisimulation) A partition $\mathcal{P}$ is homogeneous if $\mathcal{P}$ is uniform with respect to the reward function and if all blocks in $\mathcal{P}$ are stable. We say that a partition is a *stochastic bisimulation* if it is homogeneous [6].

For example, consider the MDP with five states in Fig. 8 (upper). Suppose that the following conditions hold for this MDP:

- For all $s \in S$, $A(s) = a$,
- $R(B_1, a) = -1$, and
- $R(B_2, a) = 0$.

Based on these conditions, we can say that $\mathcal{P} = \{B_1, B_2\}$ is a homogeneous partition that results in an MDP M' containing only two abstract states (Fig. 8 (lower)).

In order to find a stochastic bisimulation, we need to define an initial partition that can be a uniform partition with respect to the reward function. After that, we

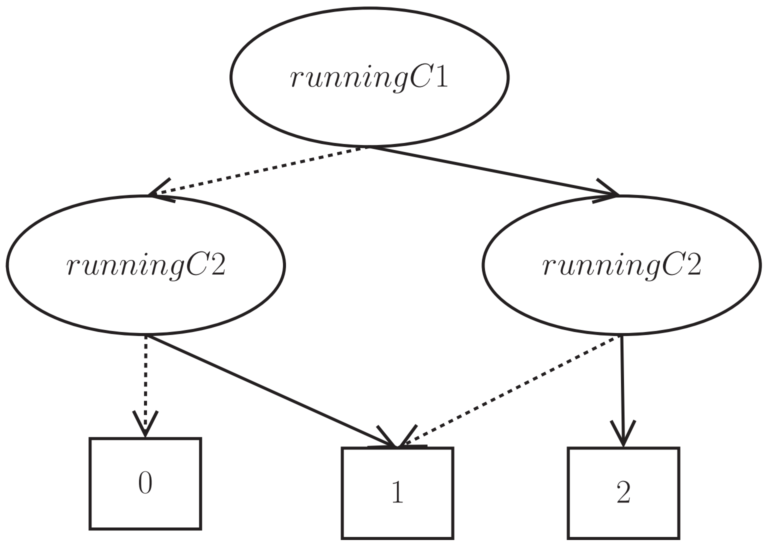

Fig. 4 Reward function represented as an ADD. The same reward function from Fig. 3 represented as an ADD

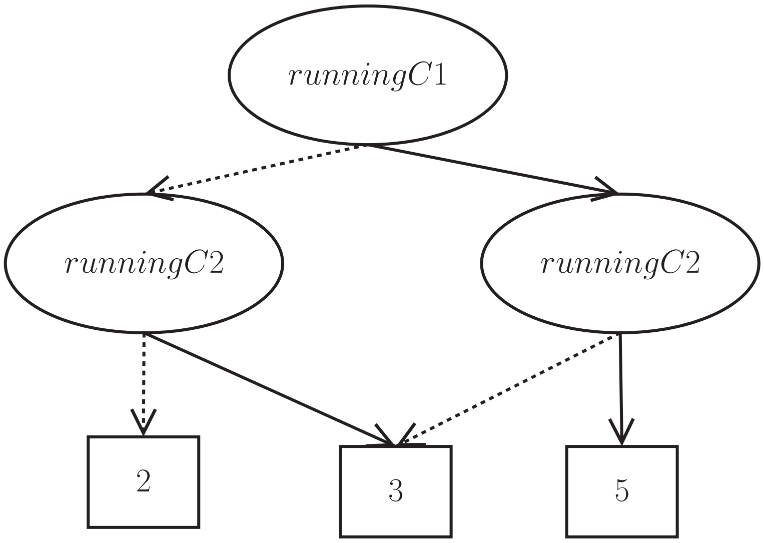

Fig. 5 A partition represented as an ADD and its DNF expression. A partition $\boldsymbol{\mathcal{P}}$ of a set of states S in which the blocks are represented by the set of state variables $X = \{X_1, X_2\}$. The partition is $\boldsymbol{\mathcal{P}} = \{(X_1 \wedge X_2, 2), ((X_1 \wedge \neg X_2) \vee (\neg X_1 \wedge \neg X_2), 3), (\neg X_1 \wedge X_2, 5)\}$

need to refine the partition blocks in order to make them stable, i.e., we refine them by splitting blocks that contain states that should not be together according to the transition function of these states. The process stops when all blocks are stable and the resulting partition is a stochastic bisimulation [6].

Definition 7. (Reduced enumerative MDP) Given a partition $\mathcal{P}$ that is a stochastic bisimulation, the reduced enumerative MDP [6] is defined as $M' = (S', A, P', R', \gamma)$, where S' is given by the blocks of $\mathcal{P}$, i.e., $B_i \in \mathcal{P}$ is an abstract state belonging to S'; $R'(B_i, a) = R(s, a)$ for any $s \in B_i$; and $P'(B_w|B_j, a) = \sum_{s' \in B_w} P(s'|s, a)$

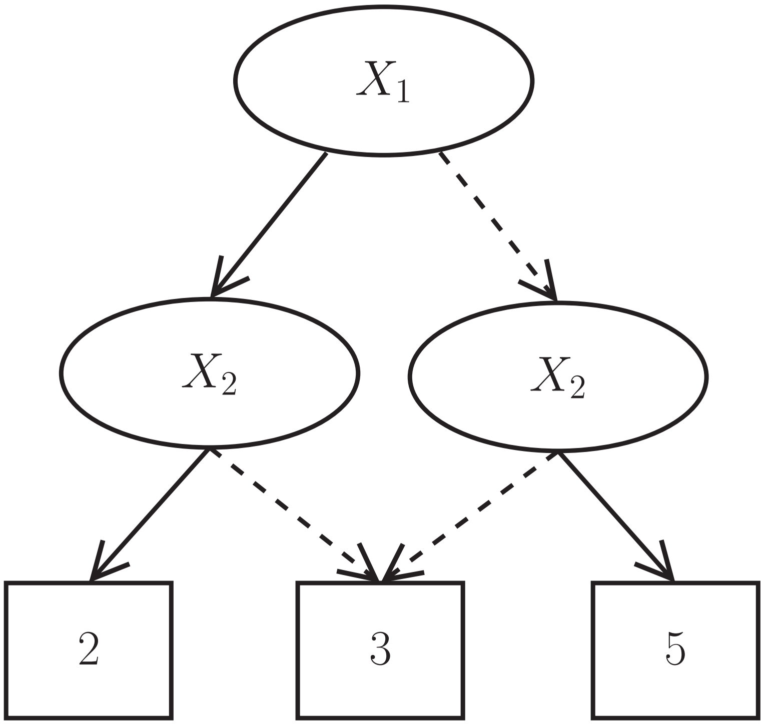

Fig. 6 A SysAdmin partition represented with an ADD. Partition obtained based on the reward function from Fig. 4

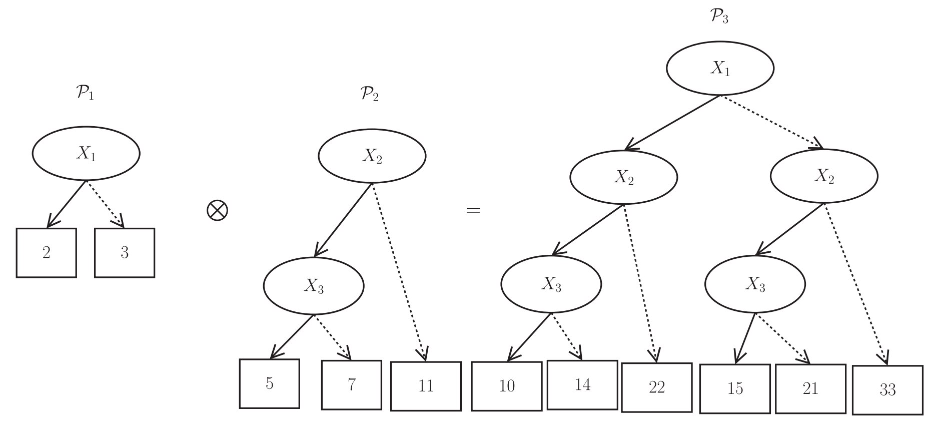

Fig. 7 Refinement of partitions with ADDs. Refinement of partitions $\boldsymbol{\mathcal{P}}_1$ and $\boldsymbol{\mathcal{P}}_2$ from Fig. 1, computed using the product $\boldsymbol{\mathcal{P}}_1 \otimes \boldsymbol{\mathcal{P}}_2$

for any $s \in B_j$ and for all $B_w \in P$, $B_j \in P$ and $a \in A$.

Theorem 1. *Given a stochastic bisimulation $\mathcal{P}$ for an MDP M and let M' be the reduced MDP obtained from $\mathcal{P}$, then an optimal policy for M' is also optimal for M* [6].

The advantage of solving M' is the possibility of doing less updates while looking for an optimal policy. For example, if $s_1, s_5 \in S$ and $B_i = \{s_1, s_5\} \in S'$, we can update only $V(B_i)$ instead of $V(s_1)$ and $V(s_5)$, with the guarantee (Theorem 1) that in the optimal policy, $\pi^*(s_1) = \pi^*(s_5) = \pi^*(B_1)$.

Factored model reduction

Model reduction with factored splits (MRFS) is an algorithm to compute a homogeneous partition for factored MDPs [6]. MRFS can find a stochastic bisimulation using concepts of MDPs partitions combined with an operation called *structure-based split (SS)* [6], that refines a single block $B_j \in \mathcal{P}$ with respect to a block $B_w \in \mathcal{P}$ in order to generate sub-blocks stable with respect to $B_w \in \mathcal{P}$. The SS operation receives a block $B_j \in \mathcal{P}$, a block $B_w \in \mathcal{P}$ and returns a partition $\mathcal{P}'$ that is a refinement of $\mathcal{P}$ in which B_j is replaced by sub-blocks $B'_j = \{B'_1, \ldots, B'_l\}$ such that any sub-block in B'_j is stable

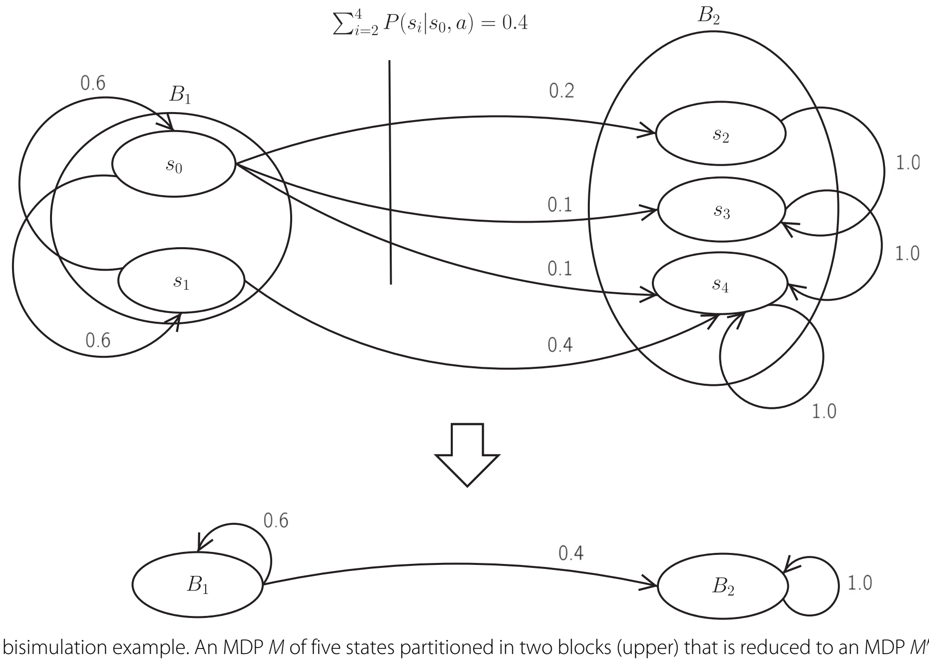

Fig. 8 A stochastic bisimulation example. An MDP *M* of five states partitioned in two blocks (upper) that is reduced to an MDP *M'* with two abstract states (lower)

with respect to B_w (Definition 4). This is computed as follows [6]:

$$\text{SS}\left(B_j, B_w, \mathcal{P}\right) = \left(\mathcal{P} - \{B_j\}\right) \cup \left(\bigcap_{a\in A} \text{BS}\left(B_j, B_w, a\right)\right), \tag{6}$$

where $\text{BS}\left(B_j, B_w, a\right) = B_j \cap \left(\bigcap_{X_i \in \text{vars}(B_w)} \mathcal{P}^a_{X_i}\right)$ represents a block split considering one action and vars(B_w) is the set of state variables used to represent block B_w [6]. When BS(B_j, B_w, a) is done, we have a partition of block B_j that is stable with respect to B_w considering only action a. To have B'_j stable with respect to B_w, SS calls BS for each action $a \in A$ and compute the intersection of the resultant partitions. The usage of vars(B_w) instead of all state variables in X is a way to efficiently compute the partitions, ignoring variables that will not affect the transition to block B_w.

With SS, model reduction does not enumerate all states explicitly while refining blocks, since the splits are done using only the state variables in vars(B_w) to refine a block B_j with respect to a block B_w [6]. MRFS works as follows. The process starts with a uniform partition with respect to the reward function. After that, while the current partition $\mathcal{P}$ contains a pair of blocks $B_j, B_w \in \mathcal{P}$ such that B_j is not stable with respect to B_w, the process calls SS using B_j and B_w as parameters. In the refined partition, the sub-blocks of B_j are stable with respect to B_w [6].

MRFS can also be computed with ADDs [19] by generating a partition for each MDP function and refine them. To do this efficiently, we can ignore some of these partitions considering that in the factored MDP structure, we can have state variables that are irrelevant to model reduction [20]. Thus, model reduction can consider only the *essential state variables*, named X_e. One way to compute X_e is to add the state variables in $\mathcal{P}^A_R$ to X_e and, after that, recursively add the parents of those variables looking for the factored MDP DBNs [20]. Using the set X_e, we can compute MRFS with ADDs as follows:

$$\mathcal{P}_{DD} = \mathcal{P}^{A,R}_{DD} \otimes \mathcal{P}^{A,X_e}_{DD}, \tag{7}$$

where $\mathcal{P}^{A,R}_{DD} = \bigotimes_{a\in A} \mathcal{P}^{a,R}_{DD}$ and $\mathcal{P}^{A,X_e}_{DD} = \bigotimes_{a\in A} \left(\bigotimes_{X_i \in X_e} \mathcal{P}^{a,X_i}_{DD}\right)$.

Methods

In the next sections we present the techniques that we use to improve model reduction performance.

Stochastic bisimulation over the reachable states

Suppose we have an MDP in which there is initial state information, that is, an MDP where we know a given initial state $s_0 \in S$. MRFS can be improved if we use the reachable states information given s_0, specially in problems that have sparse transition matrixes, i.e., where the set of reachable states can be much smaller than the complete set of states.

Let $S_{\text{reachable}|s_0}$ be the set of reachable states given s_0. If this set is computed before we look for a stochastic bisimulation, it is possible to get a *reachability partition* $\mathcal{P}^{S|s_0}_{DD} = \left\{\left(B_{\text{reachable}|s_0}, 1\right), \left(B_{\neg\text{reachable}|s_0}, 0\right)\right\}$, in which $B_{\text{reachable}|s_0}$ represents the block of reachable states given s_0 $\left(S_{\text{reachable}|s_0}\right)$ and $B_{\neg\text{reachable}|s_0}$ represents the block of unreachable states given s_0 $\left(S \backslash S_{\text{reachable}|s_0}\right)$.

Figure 9 summarizes three different kinds of partitions that could be done over a set of states S:

(a) a partition in which the states in S are divided into $|S|$ blocks, i.e., each block contains a single state;
(b) a partition over S in which each block can contain more than one state; and
(c) a partition over S that has refinements only in the subset $S_{\text{reachable}|s_0}$.

If we consider only partitions of item (c) to compute stochastic bisimulations, we can have the following advantages:

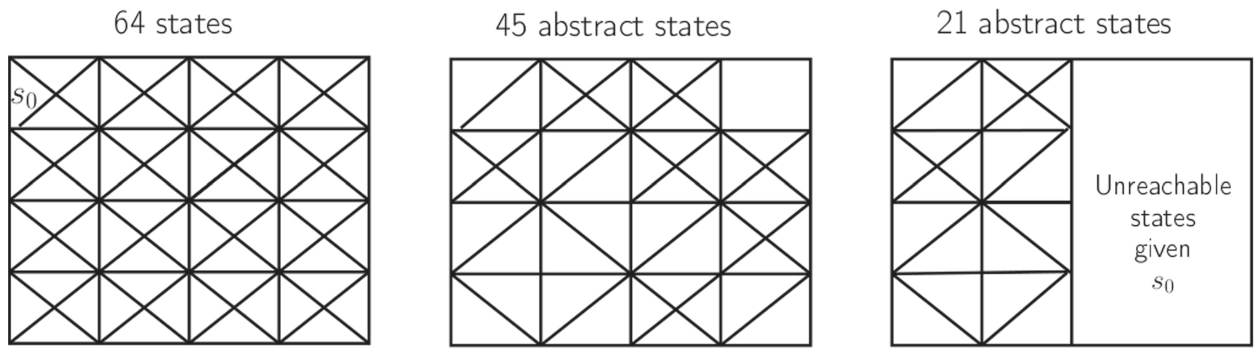

Fig. 9 Different partitions of S. The first box represents a set of states. The second box is a partition of the first box, and the third box is also a partition of the first box but only over the reachable states given s_0

- model reduction begins with smaller partitions, what makes computing more efficient; and
- the reduced model can be smaller than (b).

To find $\mathcal{P}_{DD}^{S|s_0}$, it is necessary to visit all the reachable states given s_0. This procedure is done layer by layer, simulating each action of the MDP in each reachable state (similarly to a breadth-first search [21]). Reachable states can be efficiently computed representing the sets of states in each layer as BDDs [22] and using BDD operations such as union, intersection, and marginalization. Algorithm 1 shows how we compute the reachable states given an MDP M and a state s_0.

Algorithm 1: GetReachabilityPartition (M, s_0)

```
Input: $M$: a factored MDP, $s_0$: an initial state
       represented in the factored way
Output: $\mathcal{P}_{DD}^{S|s_0}$: a reachability partition.
$\mathcal{P}_{DD}^{S|s_0} \leftarrow \{0\}$;
$\mathcal{P}_{DD_{prev}}^{S|s_0} \leftarrow \{0\}$;
$CurrentLayer_{DD} \leftarrow \{0\}$;
$CurrentLayer_{DD} \leftarrow CurrentLayer_{DD} \cup \{s_0 \mapsto 1\}$;
while true do
    $NextLayer_{DD} \leftarrow \emptyset$;
    $\mathcal{P}_{DD_{prev}}^{S|s_0} \leftarrow \mathcal{P}_{DD}^{S|s_0}$;
    foreach $a \in A$ do
        if $NextLayer_{DD} = \emptyset$ then
            // Consider that GetSuccessorsByAction do
            what it is proposed to do and ;
            // return a new layer of DD.;
            $NextLayer_{DD} \leftarrow$
            $GetSuccessorsByAction(CurrentLayer_{DD}, a)$;
        end
        else
            $NextLayerByAction_{DD} \leftarrow$
            $GetSuccessorsByAction(CurrentLayer_{DD}, a)$;

            $NextLayer_{DD} \leftarrow$
            $NextLayer_{DD} \cup NextLayerByAction_{DD}$;
        end
    end
    $\mathcal{P}_{DD}^{S|s_0} \leftarrow \mathcal{P}_{DD}^{S|s_0} \cup NextLayer_{DD}$;
    if $\mathcal{P}_{DD_{prev}}^{S|s_0} = \mathcal{P}_{DD}^{S|s_0}$ then
        break;
    end
    $CurrentLayer_{DD} \leftarrow NextLayer_{DD}$;
end
return $\mathcal{P}_{DD}^{S|s_0}$;
```

Reachability-based model reduction

Reachability-based MRFS (ReachMRFS) is an extended version of MRFS that computes stochastic bisimulations over $S_{\text{reachable}|s_0}$. Hence, it is required to compute first the reachability partition, $\mathcal{P}_{DD}^{S|s_0}$ (Algorithm 1), and after that, we compute MRFS considering only the states in $B_{\text{reachable}|s_0}$.

The algorithm ReachMRFS (Algorithm 2) works as follows. In the first external *foreach*, partitions $\mathcal{P}_{DD}^{a,R}$ are multiplied by the partition $\mathcal{P}_{DD}^{S|s_0}$, resulting in the partitions $\mathcal{Q}_{DD}^{a,R}$ (i.e., partitions based on the reward function for an action a over the reachability partition), where unreachable states are labeled with 0. As reachability partition has only two blocks, labeled with the values 0 and 1, when we compute the product of this partition with other MDP partitions, some leaves get the value 0 and others stay the same. In this way, many leaves can receive a value 0, which reduces the number of leaves and enables us to have a more compact representation. Furthermore, if we compute the refinements among partitions that were already simplified, the number of blocks in the refined partition is smaller than the number of partitions using all the possible states. The partitions $\mathcal{Q}_{DD}^{a,R}$ are used to compute a uniform partition with respect to the reward function (i.e., for all action $a \in A$) over $S_{\text{reachable}}$, that we call $\mathcal{P}_{DD}^{A,R}$.

Algorithm 2: ReachMRFS(M, $P_{DD}^{S|s_0}$, X_e)

```
Input: $M$: a factored MDP; $P_{DD}^{S|s_0}$: a reachability
       partition, $X_e$: the essential state variables.
Output: $\mathcal{P}$: a partition in which exists a stochastic
        bisimulation over the reachable states and 1
        block that contains all the unreachable states
        given $s_0$.
$\mathcal{P}_{DD}^{A,R} \leftarrow 1$;
foreach $a \in A$ do
    $\mathcal{Q}_{DD}^{a,R} \leftarrow \mathcal{P}_{DD}^{a,R} \otimes \mathcal{P}_{DD}^{S|s_0}$;
    $\mathcal{P}_{DD}^{A,R} \leftarrow \mathcal{P}_{DD}^{A,R} \otimes \mathcal{Q}_{DD}^{a,R}$;
end
$\mathcal{P}_{DD}^{A,X_e} \leftarrow 1$;
foreach $a \in A$ do
    $\mathcal{Q}_{DD}^{a,X_e} \leftarrow 1$;
    foreach $X_i \in X_e$ do
        $\mathcal{Q}_{DD}^{a,X_e} \leftarrow \mathcal{Q}_{DD}^{a,X_e} \otimes \mathcal{P}_{DD}^{a,X_i} \otimes \mathcal{P}_{DD}^{S|s_0}$;
    end
    $\mathcal{P}_{DD}^{A,X_e} \leftarrow \mathcal{P}_{DD}^{A,X_e} \otimes \mathcal{Q}_{DD}^{a,X_e}$;
end
$\mathcal{P}_{DD} \leftarrow \mathcal{P}_{DD}^{A,R} \otimes \mathcal{P}_{DD}^{A,X_e}$;
return $\mathcal{P}_{DD}$;
```

In the second external *foreach*, we compute a partition with stable blocks over the reachable states in the same way, by getting a simpler partition $\mathcal{P}_{DD}^{A,X}$. Finally, in the last two lines, we compute the stochastic bisimulation over reachable states and return it.

While MRFS computes the partition intersections as in Fig. 7, ReachMRFS use the reachability partition to simplify the computation. The advantages appear in two ways: (1) the program needs less space to store the ADDs in main memory and (2) the products among partitions can be done faster because many leaves are equal to 0, specially in problems with a sparse transition matrix. The benefits of ReachMRFS can be seen in Figs. 10 and 11 that shows the reduction in the size of the ADDs and hence, in the resulting partition when compared with the refinement in Fig. 7.

Reachability-based model reduction with partitions elimination

Another improvement to efficiently compute stochastic bisimulations is the elimination of repeated partitions computed by the algorithms MRFS and ReachMRFS. These algorithms use partitions based on reward and transition functions. In a general way, the maximum number of distinct MDP partitions used by these algorithms is at most $|A| + (|A| \times |X|)$. However, in practical situations, these partitions are not all distinct and if we compute the stochastic bisimulation using all these partitions, the process can be very slow.

Let $\mathcal{P}$ and $\mathcal{P}'$ be partitions of an MDP. We say that $\mathcal{P} = \mathcal{P}'$ if $|\mathcal{P}| = |\mathcal{P}'|$ and if for each $B_i \in \mathcal{P}$ there is a block $B_j \in \mathcal{P}'$ with the same DNF characterizing both of them. Based on this concept, a partition $\mathcal{P}'$ obtained from a function f (reward or probabilistic transition) is repeated if the reduction algorithm found a partition $\mathcal{P}$ (before $\mathcal{P}'$), obtained from a function g (reward or probabilistic transition), and we have $\mathcal{P} = \mathcal{P}'$. Given a list of partitions L, we say the partitions are distinct among themselves if for each partition $\mathcal{P}_i \in L$ obtained from an MDP function, L does not have $\mathcal{P}_j$ obtained from another MDP function such that $\mathcal{P}_i = \mathcal{P}_j$.

ReachMRFS-V2 (Algorithm 3) is a version of ReachMRFS that starts computing partition elimination and then computes the refinements among the partitions in L.

Algorithm 3: ReachMRFS-V2 (M, $P_{DD}^{S|s_0}$, X_e)

```
Input: $M$: a factored MDP, $P_{DD}^{S|s_0}$: a reachability
       partition, $X_e$: a set of essential state variables in
       the MDP.
Output: $\mathcal{P}$: a partition in which exists a stochastic
        bisimulation over the reachable states and 1
        block that contains all the unreachable states
        given $s_0$.
$\mathcal{P} \leftarrow 1$;
$\mathcal{P}_{distinct} \leftarrow \emptyset$ /* initializes a map that say if a partition
is repeated or not.*/;
foreach $a \in A$ do
    $\mathcal{P}_{distinct}.add(\mathcal{P}_R^a \mapsto true)$;
    foreach $X_i \in X_e$ do
        $\mathcal{P}_{distinct}.add(\mathcal{P}_{X_i}^a \mapsto true)$
    end
end
for $i = 1$ to $|\mathcal{P}_{distinct}|$ do
    if $\mathcal{P}_{distinct}[i] = true$ then
        for $j = i + 1$ to $|\mathcal{P}_{distinct}|$ do
            if $\mathcal{P}_{distinct}[i] = \mathcal{P}_{distinct}[j]$ then
                $\mathcal{P}_{distinct}[j] \leftarrow (\mathcal{P}_{distinct}[j] \mapsto false)$;
            end
        end
    end
end
$L \leftarrow \emptyset$;
for $i = 1$ to $|\mathcal{P}_{distinct}|$ do
    if $\mathcal{P}_{distinct}[i] = true$ then
        $L.add(\mathcal{P}_{distinct}[i])$;
    end
end
/* Computes the refinements using the partitions in L
and the reachable states in $P_{DD}^{S|s_0}$. */;
foreach $\mathcal{P}_i \in L$ do
    $\mathcal{P} \leftarrow \mathcal{P} \cap \mathcal{P}_i \cap P_{DD}^{S|s_0}$;
end
return $\mathcal{P}$;
```

Results and discussion

In [23], we have shown that ReachMRFS-V2 can be much more efficient than MRFS, specially in sparse domains. At the same time, in dense domains, the overhead of reachability analysis does not affect the general performance of model reduction because we use efficient ADD operations to find the reachable states. In this work, we complete those results by including VI and TVI to solve the reduced MDP. Furthermore, we also solve the original MDP M using these three algorithms to identify the trade-offs of applying model reduction.

For all algorithms, we use $\gamma = 0.99$ and $\epsilon = 10^{-3}$. The methodology employed in experiments is to execute each algorithm until convergence enforcing a time and memory cutoff of 1 h and 3GB, respectively. The results are presented as the average and 95 % confidence interval over 30 executions for each pair of planner and problem. The experiments were conducted on a 2.4-GHz machine with 16 cores running a 64-bit version of Linux and our implementation was developed

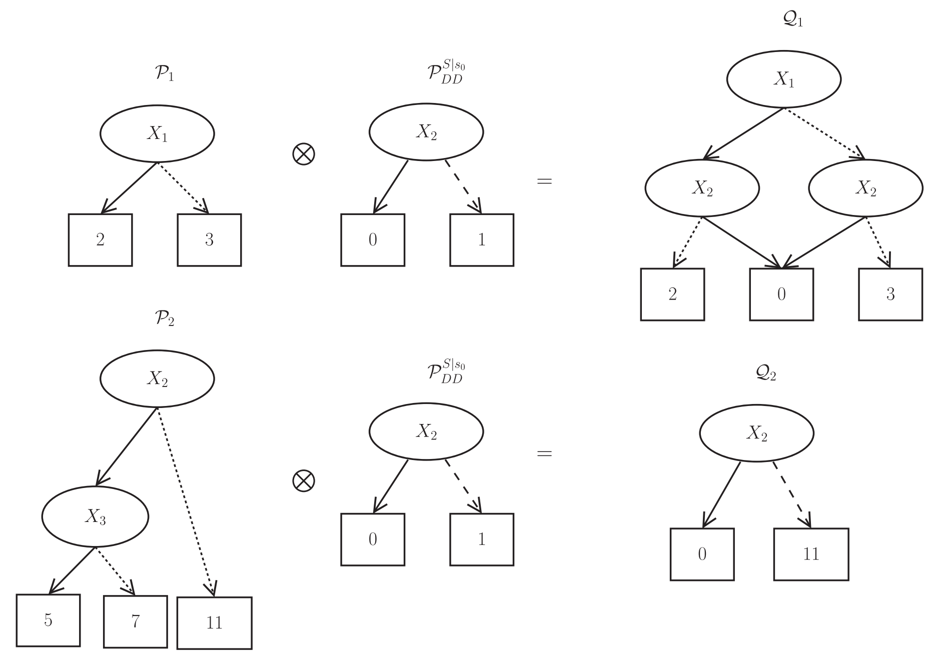

Fig. 10 Partitions over the set of reachable states given s_0. The partitions $\boldsymbol{\mathcal{P}}_1$ and $\boldsymbol{\mathcal{P}}_2$ of Fig. 7 are multiplied by the resulting partition $\boldsymbol{\mathcal{P}}_{DD}^{S|s_0}$, resulting in $\boldsymbol{\mathcal{Q}}_1$ and $\boldsymbol{\mathcal{Q}}_2$

in Java using the official *RDDL Simulator (RDDLSim)* [24]. Our implementation is available in the following repository: http://github.com/felipemartinsss/repository/tree/master/AIPlannersForRDDLSim.

The benchmark problems were taken from the *International Probabilistic Planning Competition (IPPC-2011)*, which were specified in *RDDL (Relational Dynamic Influence Diagram Language)* [24], a language to represent factored MDPs. In the IPPC-2011, there were eight domains with ten instances each and we selected the following domains for our experiments: *Crossing Traffic*, *Elevators*, *Game of Life*, *Navigation* and *Skill Teaching*.

Table 1 presents the comparison between ReachMRFS-V2 and MRFS using VI, TVI, and LRTDP as planners to solve the reduced MDP. As expected, the performance of the ReachMRFS-V2 dominates the performance of MRFS in all the problems, i.e., the slowest ReachMRFS-V2 planner (usually ReachMRFS-V2 + LRTDP) is faster than all planners that use MRFS planners. This is due the reachability analysis and partition elimination applied by ReachMRFS-V2.

Another interesting trend in Table 1 is that TVI (LRTDP) is the best (worst) planner for solving the reduced MDPs generated by both ReachMRFS-V2 and MRFS. The reason for this trend is the fact that all the states in the reduced MDPs are reachable from their initial states. Therefore, the sampling procedure of LRTDP is inefficient in comparison with the topological analysis

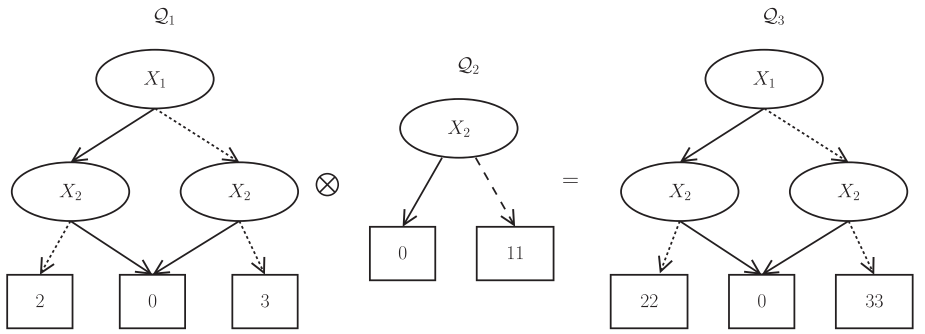

Fig. 11 Refinement of partitions considering reachable states. The same computation of Fig. 7, but considering that we have information of the initial state and the reachability partition (computed in Fig. 10)

Table 1 Average and 95 % confidence interval of the time, in seconds, to solve each problem using $\epsilon = 10^{-3}$

Problem		ReachMRFS-V2 + TVI	ReachMRFS-V2 + VI	ReachMRFS-V2 + LRTDP	MRFS + TVI	MRFS + VI	MRFS + LRTDP
Cross.	1	**3.64 ± 0.04**	4.10 ± 0.03	5.71 ± 0.08	19.41 ± 0.60	19.87 ± 0.65	20.86 ± 0.59
	2	**3.65 ± 0.03**	4.14 ± 0.05	6.53 ± 0.16	19.36 ± 0.60	19.80 ± 0.58	21.82 ± 0.67
	3	**15.29 ± 0.11**	22.74 ± 0.21	26.72 ± 0.40	-	-	-
	4	**15.17 ± 0.14**	22.49 ± 0.34	25.05 ± 0.34	-	-	-
Elevators	1	**4.45 ± 0.04**	4.90 ± 0.07	16.08 ± 0.33	100.10 ± 4.56	97.57 ± 5.49	101.99 ± 4.07
	2	161.60 ± 1.42	**155.99 ± 1.35**	1788.44 ± 14.60	-	-	-
	3	**146.87 ± 1.04**	149.76 ± 1.24	1468.64 ± 11.85	-	-	-
	4	16.86 ± 0.65	**15.35 ± 0.26**	146.66 ± 1.22	-	-	-
	7	**72.25 ± 0.83**	90.99 ± 0.86	3599.00 ± 0.11	-	-	-
Game	1	**137.10 ± 0.48**	192.98 ± 0.80	215.95 ± 1.31	161.62 ± 1.04	211.74 ± 1.31	234.31 ± 1.65
	2	**118.89 ± 0.35**	174.03 ± 0.97	239.72 ± 1.27	140.83 ± 0.97	191.77 ± 1.16	260.07 ± 2.40
	3	**114.57 ± 0.49**	168.88 ± 0.48	233.54 ± 1.51	137.38 ± 0.83	187.54 ± 1.21	253.25 ± 1.87
Navigation	1	**1.76 ± 0.01**	1.97 ± 0.01	2.41 ± 0.03	21.09 ± 0.76	20.36 ± 1.00	21.39 ± 0.82
	2	**2.29 ± 0.02**	2.59 ± 0.02	2.91 ± 0.03	189.90 ± 8.69	191.50 ± 6.66	194.26 ± 6.82
	3	**2.72 ± 0.07**	3.45 ± 0.03	3.26 ± 0.02	-	-	-
	4	**4.61 ± 0.04**	4.80 ± 0.04	4.83 ± 0.04	-	-	-
	5	**4.32 ± 0.05**	4.80 ± 0.05	5.18 ± 0.06	-	-	-
	6	**6.66 ± 0.07**	7.17 ± 0.07	8.03 ± 0.09	-	-	-
	7	**8.53 ± 0.06**	8.84 ± 0.07	9.64 ± 0.07	-	-	-
	8	**11.40 ± 0.09**	12.24 ± 0.11	13.39 ± 0.10	-	-	-
	9	**20.33 ± 0.21**	21.58 ± 0.21	22.94 ± 0.22	-	-	-
	10	**36.53 ± 0.51**	37.55 ± 0.47	38.55 ± 0.45	-	-	-
Skill	1	**2.76 ± 0.02**	2.86 ± 0.03	4.48 ± 0.07	19.24 ± 1.17	18.98 ± 1.00	18.07 ± 0.85
	2	**2.75 ± 0.04**	2.87 ± 0.03	4.51 ± 0.12	18.72 ± 0.95	18.20 ± 0.87	17.46 ± 0.60
	3	27.91 ± 0.23	**26.15 ± 0.23**	112.63 ± 0.80	175.94 ± 5.73	176.50 ± 7.12	328.26 ± 11.62
	4	27.55 ± 0.26	**25.92 ± 0.30**	100.38 ± 0.79	178.21 ± 7.62	172.60 ± 7.85	311.95 ± 12.43

If a solution is not found in the given time and memory thresholds, then '-' is shown. Best performance over all planners (columns) is shown in bold font. In the columns, Algorithm 1 + Algorithm 2 refers to the algorithm used in reduction phase (Algorithm 1) and the algorithm used to solve the reduced MDP (Algorithm 2)

Table 2 Average and 95 % confidence interval of the time, in seconds, to solve each problem using $\epsilon = 10^{-3}$

Problem		ReachMRFS-V2 + TVI	ReachMRFS-V2 + VI	ReachMRFS-V2 + LRTDP	TVI	VI	LRTDP
Cross.	1	3.64 ± 0.04	4.10 ± 0.03	5.71 ± 0.08	4.51 ± 0.03	2594.45 ± 58.66	**3.16 ± 0.05**
	2	3.65 ± 0.03	4.14 ± 0.05	6.53 ± 0.16	4.47 ± 0.04	2007.41 ± 37.64	**2.92 ± 0.04**
	3	15.29 ± 0.11	22.74 ± 0.21	26.72 ± 0.40	18.95 ± 0.16	-	**12.79 ± 0.18**
	4	**15.17 ± 0.14**	22.49 ± 0.34	25.05 ± 0.34	19.50 ± 0.31	-	17.43,0.18
Elevators	1	**4.45 ± 0.04**	4.90 ± 0.07	16.08 ± 0.33	5.83 ± 0.16	135.46 ± 3.61	13.50 ± 0.38
	2	161.60 ± 1.42	**155.99 ± 1.35**	1788.44 ± 14.60	240.45 ± 6.25	-	555.47 ± 7.06
	3	**146.87 ± 1.04**	149.76 ± 1.24	1468.64 ± 11.85	217.75 ± 5.41	-	980.74 ± 8.64
	4	16.86 ± 0.65	15.35 ± 0.26	146.66 ± 1.22	**14.61 ± 0.20**	1964.44 ± 32.10	82.39 ± 1.09
	7	**72.25 ± 0.83**	90.99 ± 0.86	3599.00 ± 0.11	78.90 ± 1.17	208.30 ± 3.48	1363.33 ± 11.72
Game	1	**137.10 ± 0.48**	192.98 ± 0.80	215.95 ± 1.31	557.46 ± 6.35	514.48 ± 4.60	917.27 ± 14.06
	2	**118.89 ± 0.35**	174.03 ± 0.97	239.72 ± 1.27	440.28 ± 5.33	399.34 ± 3.72	1263.55 ± 9.63
	3	**114.57 ± 0.49**	168.88 ± 0.48	233.54 ± 1.51	406.64 ± 4.95	369.03 ± 3.74	1399.49 ± 7.12
Navigation	1	1.76 ± 0.01	1.97 ± 0.01	2.41 ± 0.03	**1.40 ± 0.01**	43.49 ± 0.96	2.01 ± 0.02
	2	2.29 ± 0.02	2.59 ± 0.02	2.91 ± 0.03	**1.47 ± 0.01**	365.03 ± 10.71	2.16 ± 0.04
	3	2.72 ± 0.07	3.45 ± 0.03	3.26 ± 0.02	**1.64 ± 0.01**	-	2.46 ± 0.03
	4	4.61 ± 0.04	4.80 ± 0.04	4.83 ± 0.04	**3.30 ± 0.03**	-	3.85 ± 0.23
	5	4.32 ± 0.05	4.80 ± 0.05	5.18 ± 0.06	**2.56 ± 0.03**	-	3.50 ± 0.08
	6	6.66 ± 0.07	7.17 ± 0.07	8.03 ± 0.09	**3.60 ± 0.05**	-	6.62 ± 0.40
	7	8.53 ± 0.06	8.84 ± 0.07	9.64 ± 0.07	**5.10 ± 0.09**	-	5.66 ± 0.26
	8	11.40 ± 0.09	12.24 ± 0.11	13.39 ± 0.10	**4.61 ± 0.03**	-	5.69 ± 0.10
	9	20.33 ± 0.21	21.58 ± 0.21	22.94 ± 0.22	**5.59 ± 0.07**	-	6.66 ± 0.14
	10	36.53 ± 0.51	37.55 ± 0.47	38.55 ± 0.45	7.93 ± 0.09	-	**7.73 ± 0.12**
Skill	1	2.76 ± 0.02	2.86 ± 0.03	4.48 ± 0.07	4.09 ± 0.02	38.62 ± 0.98	**1.49 ± 0.02**
	2	2.75 ± 0.04	2.87 ± 0.03	4.51 ± 0.12	4.11 ± 0.03	41.03 ± 1.00	**1.56 ± 0.02**
	3	27.91 ± 0.23	26.15 ± 0.23	112.63 ± 0.80	38.25 ± 0.71	-	**12.44 ± 0.38**
	4	27.55 ± 0.26	25.92 ± 0.30	100.38 ± 0.79	38.84 ± 0.75	-	**11.95 ± 0.34**

If a solution is not found in the given time and memory thresholds, then '-' is shown. Best performance over all planners (columns) is shown in bold font. Results for all Reach MRFS planners are the same as in Table 1 applied by TVI since no state in the reduced MDPs can be ignored.

In Table 2, we compare ReachMRFS-V2 against no model reduction using VI, TVI, and LRTDP as planners.

For the *Crossing Traffic* domain, model reduction initially does not pay off and LRTDP has the best performance; however, as the size of the problem increases (problem #4), the reduced MDP is considerably smaller and ReachMRFS-V2 + TVI has the best overall performance.

In the *Elevators* domain, ReachMRFS-V2 + TVI is the overall best planner with ReachMRFS-V2 + VI being a closer runner-up. The reason for such performance is that the reachability analysis suffices to reduce the problem to a few thousand states. However, the reachable space contains only one strongly connected component, what is the worst case for TVI, therefore the overhead of the model reduction pays off when compared with TVI.

For the *Game of Life* domain, ReachMRFS-V2 + TVI dominates all the other planners and solve all the instances from three to ten times faster than the approaches without model reduction. This performance improvement is because the *Game of Life* problems are dense and all their states are reachable. Therefore, all the reduction obtained by ReachMRFS-V2 is due to the stochastic bisimulations.

For the *Navigation* domain, TVI has the best performance in almost all the instances because all the model reduction is due only to the reachability analysis; therefore, the stochastic bisimulations represent an expensive overhead.

For the *Skill Teaching* domain, LRTDP dominates all the other approaches because this domain is sparse and most of the model reduction is due to the reachability analysis. For instance, the problem number 4 has 2^{24} possible states, only 1053 of them are reachable from s_0, and the reduced model has 702 states. Therefore, the stochastic bisimulation contributes with a negligible reduction in the model.

Conclusions

The experiment results show that model reduction always pays off when we have information about the initial state. For MDPs with dense transition function, the results show that stochastic bisimulation computation can explore the domain structure and, combined with the TVI planner to solve the reduced model, can have the best performance. For instance, the *Game of Life* domain problems with nine variables were solved in 2–3 min with ReachMRFS-V2 + TVI (while without the reduction, LRTDP takes 15–23 min). For sparse domains like *Elevators* and *Crossing traffic*, the results show that there is an overhead of the model reduction. However, for 8 out of 26 problems, this overhead pays off and ReachMRFS-V2-based planners are the best overall considered planners. It is a future research topic to find a general rule for deciding whenever ReachMRFS-V2 should be applied or not.

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
This work is part of FMS Master thesis and LNB was his advisor. FMS carried out most of the investigation, implementation, experiments, and writing. LNB gave important ideas to the project and also contributed in the writing and revision of this manuscript. FWT gave important ideas about new techniques to solve MDPs and the implementation of the TVI algorithm and suggested a new series of experiments; he also gave an important contribution in writing and revising the paper. All authors read and approved the final manuscript.

Acknowledgements
The authors would like to thank:

- CAPES, for supporting FMS during the master degree research;
- FAPESP, for supporting FWT (process number 2013/11724-0);
- Mijail Gamarra Holguin, for helping with the implementation of the planners and the use of ADD library; and
- Karina Valdivia Delgado, for some ideas on factored representation and operations of MDPs.

Received: 20 May 2014 Accepted: 30 April 2015
Published online: 27 May 2015

References

1. Puterman ML (1994) Markov decision processes: discrete stochastic dynamic programming. John Wiley & Sons, New York, NY, USA
2. Hoey J, St-Aubin R, Hu A, Boutilier C (1999) SPUDD: Stochastic planning using decision diagrams. In: Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence. Morgan Kauffman, San Franciso, CA, USA. pp 279–288
3. Feng Z, Hansen EA, Zilberstein S (2003) Symbolic generalization for on-line planning. In: Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence. Morgan Kaufmann, San Francisco, CA, USA. pp 109–116
4. Barto AG, Bradtke SJ, Singh SP (1993) Learning to act using real-time dynamic programming. Artif Intell 72:81–138
5. Bonet B, Geffner H (2003) Labeled RTDP: improving the convergence of real-time dynamic programming. In: Proceedings of 13th International Conference on Automated Planning and Scheduling. AAAI Press, ICAPS, Trento, Italy. pp 12–21
6. Givan R, Greig M, Dean T (2003) Equivalence notions and model minimization in Markov decision processes. Artif Intell 147:163–223
7. Bertsekas D, Tsitsiklis JN (1996) Neuro-dynamic programming. Athena Scientific, Cambridge, MA, USA
8. Dean T, Kanazawa K (1990) A model for reasoning about persistence and causation. Comput Intell 5:142–150
9. Bahar RI, Frohm EA, Gaona CM, Hachtel GD, Macii E, Pardo A, Somenzi F (1993) Algebraic decision diagrams and their applications. In: Proceedings of the 1993 IEEE/ACM International Conference on Computer-aided Design. IEEE Computer Society Press, Los Alamitos, CA, USA. pp 188–191
10. Bryant RE (1986) Graph-based algorithms for Boolean function manipulation. IEEE Trans Comput 35:677–691
11. Dai P, Goldsmith J (2007) Topological value iteration algorithm for Markov decision processes. In: IJCAI'07 Proceedings of the 20th International Joint Conference on Artificial Intelligence. Morgan Kauffman, San Francisco, CA, USA. pp 1860–1865
12. Bertsekas DP (1995) Dynamic programming and optimal control. Vol. 1. Athena Scientific, Cambridge, MA, USA
13. Li L, Walsh TJ, Littman ML (2006) Towards a unified theory of state abstraction for MDPs. In: Proceedings of the 9th International Sysmposium on Artificial Intelligence and Mathematics, Fort Lauderdale, Florida, USA. pp 531–539
14. Dean T, Givan R, Leach S (1997) Model reduction techniques for computing approximately optimal solutions for Markov decision

processes. In: Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence. Morgan Kauffman, San Francisco, CA, USA. pp 124–131

15. Givan R, Leach S, Dean T (2000) Bounded-parameter Markov decision processes. Artif Intell 122:71–109
16. Ravindran B, Barto AG (2002) Model minimization in hierarchical reinforcement learning. Lecture Notes Comput Sci 2371/2002:196–211
17. Ravindran B, Barto AG (2004) Approximate homomorphisms: a framework for non-exact minimization in Markov decision processes. In: Proceedings of the 5th International Conference on Knowledge Based Computer Systems, Hyderabad, India
18. Boutilier C, Dearden R, Goldszmidt M (1995) Exploiting structure in policy construction. In: IJCAI-95. University of British Columbia Vancouver, BC, Canada, Canada. pp 1104–1111
19. Kim KE, Dean T (2002) Solving factored MDPs with large action space using algebraic decision diagrams. In: Proceedings of the 7th Pacific Rim International Conference on Artificial Intelligence: Trends in Artificial Intelligence. Springer-Verlag, London, UK. pp 80–89
20. Guo W, Leong TY (2010) An analytic characterization of model minimization in factored Markov decision processes. In: Proceedings of the 24th AAAI Conference on Artificial Intelligence. AAAI Press, Atlanta, Georgia. pp 1077–1082
21. Russel S, Norvig P (2003) Inteligência Artificial: Uma Abordagem Moderna. Segunda edn.. Campus/Elsevier, Rio de Janeiro
22. Pednault EPD (October, 1994) ADL and the state-transition model of action. Journal of Logic and Computation, Volume 4, Number 5:1077–1082
23. dos Santos FM, de Barros LN, Holguin MG (2013) Stochastic bisimulation for mdps using reachability analysis. In: 2013 Brazilian Conference on Intelligent Systems (BRACIS), Fortaleza, Ceará, Brazil. pp 213–218
24. Sanner S (2010) Relational dynamic influence diagram language (RDDL): language description. http://users.cecs.anu.edu.au/~ssanner/IPPC_2011/RDDL.pdf