Journal of Artificial Intelligence Research 65 (2019) 271-306 Submitted 11/2018; published 06/2019

Probabilistic Planning with Reduced Models

Luis Pineda LPINEDA@CICS.UMASS.EDU
Facebook AI Research, Montreal, QC, Canada

Shlomo Zilberstein SHLOMO@CICS.UMASS.EDU
University of Massachusetts, Amherst, MA, USA

Abstract

Reduced models are simplified versions of a given domain, designed to accelerate the planning process. Interest in reduced models has grown since the surprising success of determinization in the first international probabilistic planning competition, leading to the development of several enhanced determinization techniques. To address the drawbacks of previous determinization methods, we introduce a family of reduced models in which probabilistic outcomes are classified as one of two types: *primary* and *exceptional*. In each model that belongs to this family of reductions, primary outcomes can occur an unbounded number of times per trajectory, while exceptions can occur at most a finite number of times, specified by a parameter. Distinct reduced models are characterized by two parameters: the maximum number of primary outcomes per action, and the maximum number of occurrences of exceptions per trajectory. This family of reductions generalizes the well-known most-likely-outcome determinization approach, which includes one primary outcome per action and zero exceptional outcomes per plan. We present a framework to determine the benefits of planning with reduced models, and develop a continual planning approach that handles situations where the number of exceptions exceeds the specified bound during plan execution. Using this framework, we compare the performance of various reduced models and consider the challenge of generating good ones automatically. We show that each one of the dimensions—allowing more than one primary outcome or planning for some limited number of exceptions—could improve performance relative to standard determinization. The results place previous work on determinization in a broader context and lay the foundation for a systematic exploration of the space of model reductions.

1. Introduction

Automated planning—the ability to reason about the course of actions necessary to achieve one's goals—is the hallmark of intelligence and a core area of AI research. Planning is crucial for the development of autonomous systems in a wide range of domains, from household products such as the iRobot's Roomba vacuum cleaners to space exploration vehicles such as the Curiosity and Opportunity Mars rovers. A new generation of sophisticated systems such as autonomous wheelchairs, autonomous robotic tractors, and autonomous cars are currently under development. As the range of the tasks performed by such systems grows, so does the complexity of the associated planning problems.

When actions have probabilistic outcomes, they can be modeled using the Markov decision process (MDP) (Puterman, 1994)—a rich model that has been extensively used by the AI community for planning (Boutilier & Poole, 1996; Dean, Kaelbling, Kirman, & Nicholson, 1995; Kaelbling, Littman, & Cassandra, 1998) and learning (Barto, Bradtke, & Singh, 1995; Sutton & Barto, 1998). Work on MDPs has a long and fruitful history, starting with

the introduction of the model in the late 1950s and the pioneering work of Bellman (1957) and Howard (1960). We focus in this paper on planning under uncertainty in autonomous systems such as mobile robots (Hsu, Latombe, & Kurniawati, 2006; Koenig, Goodwin, & Simmons, 1996; Thrun, Burgard, & Fox, 2005). However, the results are equally relevant to other applications of MDPs such as network management (Singh, Alpcan, Agrawal, & Sharma, 2010), optimizing software on mobile phones (Cheung, Okamoto, Maker, Liu, & Akella, 2009), and managing water levels of river reservoirs (Petrik & Zilberstein, 2011).

In this work we are concerned with resource-bounded settings, where both planning time and plan quality must be optimized (Dean & Boddy, 1988; Horvitz, 1987; Russell & Wefald, 1991; Svegliato, Wray, & Zilberstein, 2018; Zilberstein, 1996, 2011). One paradigm for planning under time and other resource constraints is to perform a limited amount of planning online, then execute an action, and then perform more planning during execution time as needed (e.g., if the current plan fails or new information about the domain is obtained). Actions in that case are selected for execution based on an approximate plan that may be incomplete. In this context, we are particularly interested in *planning with reduced models*—models that are easier to solve and allow the planner to find partial plans very quickly. Interest in planning with reduced models has a long history, using such principles as aggregating equivalent states (Dean & Givan, 1997; Givan, Dean, & Greig, 2003; Hostetler, Fern, & Dietterich, 2014; Petrik & Subramanian, 2014; Ravindran & Barto, 2002), approximate partitioning of the state space (Dean, Givan, & Leach, 1997), or limiting plan reachability to a certain envelope (Dean et al., 1995; Trevizan & Veloso, 2014). While reduced models are designed to deliberately ignore some aspects of the domain to accelerate planning, other works have examined ways to handle missing information and known incompleteness of the domain model (Garland & Lesh, 2002; Nguyen & Kambhampati, 2014; Nguyen, Kambhampati, & Do, 2013; Weber & Bryce, 2011).

Renewed interest in planning with reduced models was prompted by the surprising success of FF-REPLAN (Yoon, Fern, & Givan, 2007) in the 2004 International Probabilistic Planning Competition (IPPC-04) (Younes, Littman, Weissman, & Asmuth, 2005). FF-REPLAN employs a simple approach: it creates a deterministic version of the underlying MDP and solves it using the FF classical planner (Hoffmann & Nebel, 2001), resulting in a *partial plan* for the original problem. This partial plan is then executed until an unexpected state is reached, upon which the planning process repeats using the current state as the initial state. The main advantage of such *determinization-based algorithms* is their ability to leverage the efficiency of classical planners and quickly generate partial plans in domains where probabilistic planners are too slow. It was surprising, however, that such a simple approach outperformed an array of competing state-of-the-art MDP solvers, which sparked substantial research to understand the benefits and alleviate the drawbacks of this approach.

In particular, some have argued that determinization-based algorithms perform poorly in settings that are *probabilistically interesting* in some sense (Little & Thiébaux, 2007). This has led to the introduction of improved determinization-based approaches that produce more robust partial plans. FF-REPLAN (Yoon et al., 2007) is the most basic of these approaches that simply solves a deterministic relaxation of the original MDP in an online fashion, without attempting to anticipate future deviations from the plan. FF-HINDSIGHT (Yoon, Fern, Givan, & Kambhampati, 2008; Yoon, Ruml, Benton, & Do, 2010) uses a *hindsight optimization* approach to approximate the value function of the original MDP by sampling multiple

deterministic futures that are solved using the FF planner. RFF (Teichteil-Königsbuch, Kuter, & Infantes, 2010) generates a plan for an envelope of states such that the probability of reaching a state outside the envelope is below some predefined threshold. To increase the envelope, RFF chooses states outside of the current plan and computes actions for these states using an approach similar to FF-REPLAN. HMDPP (Keyder & Geffner, 2008) introduces the so-called *self-loop determinization* in order to force the deterministic planner to generate plans with a low probability of deviations, using a pattern database heuristic to avoid dead ends.

Despite their partial success, determinization-based algorithms have some *inherent drawbacks*, mostly because they typically consider the outcomes of an action in isolation and are not sensitive to decision-theoretic measures such as expected utility. Consequently, these techniques can be overly optimistic (Kolobov, Mausam, & Weld, 2010) and can produce plans that are arbitrarily worse (in terms of solution cost) than the optimal plan. Furthermore, even in cases where optimal plans could actually be obtained using isolated outcomes, it is not always clear, nor intuitive, which outcomes should be included in the determinization. Our goal in this work is to exploit the insights offered by successful methods for planning with reduced models, while addressing their inherent drawbacks and limitations.

To this end, we introduce a more general paradigm in which an algorithm such as the single-outcome variant of FF-REPLAN (FF_s) (Yoon et al., 2007) is just one extreme point on a *spectrum of MDP reductions* that differ from each other along two dimensions: (1) the number of outcomes per state-action pair that are fully accounted for in the reduced model, and (2) the number of occurrences of the remaining outcomes that are planned for in advance. For example, FF_s considers one outcome per state-action pair and plans for zero occurrences of the remaining outcomes in advance (i.e., it completely ignores them). In contrast, we propose algorithms that fully account for a subset of the original outcomes—possibly a subset of cardinality greater than one—and consider the remaining outcomes at most k times in any trajectory found during plan execution.

Our work bears some resemblance to the use of the *all-outcomes determinization* (Yoon et al., 2007; Keller & Eyerich, 2011) for probabilistic planning. In this type of determinization, a deterministic reduction of the original MDP is created, with the property that each probabilistic outcome in the original problem is mapped into a deterministic action in the reduced problem. This type of determinization has the advantage that it is guaranteed to find a plan with a non-zero probability of reaching a goal, if such a plan exists. However, the all-outcomes determinization ignores information about the probability associated with different outcomes, and can therefore choose actions with high probability of leading to catastrophic outcomes. Instead, our reduced model paradigm allows the planner to use the full transition model a bounded number of times in each possible trajectory, and to exploit the differences in probabilities of the remaining outcomes once the bound is reached.

Beyond determinization, other planning approaches have been developed for reduced or incomplete models. SSIPP (Trevizan & Veloso, 2014), for example, relies on pruning all states beyond a certain reachable envelope. Other works have examined explicitly the question of improving plan robustness to known incompleteness of the domain model (Nguyen et al., 2013). Our work not only explores an entire spectrum of relaxations, but also uses an evaluation metric that minimizes the *comprehensive cost* of reaching the goal, including re-planning cost.

Hence, we introduce a new approach for *planning with reduced MDP models* that generalizes determinization-based algorithms. We begin with the definition of a family of MDP reduced models, and show how to efficiently solve reduced models using heuristic search algorithm, such as LAO* (Hansen & Zilberstein, 2001) and LRTDP (Bonet & Geffner, 2003). We then introduce a *continual planning* approach that handles situations where the number of exceptions exceeds the bound during plan execution, and study the theoretical properties of the resulting plans. In particular, we show how to evaluate the benefits of a particular reduced model analytically. We then show how to use this evaluation technique to choose a good determinization, and more generally, how to find good reduced models. In the experimental results section, we show that each of the two model reduction dimensions contribute significantly to outperforming existing approaches.

2. Problem Formulation

In this section we present the mathematical definition of the probabilistic planning problem, and formally introduce our proposed model reduction approach.

2.1 Markov Decision Processes

We focus on *stochastic shortest-path* MDPs (also known as stochastic shortest-path problems, or SSPs), in which the goal is to minimize the expected cost of reaching a set of goal states[1]. An MDP can be defined as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{C}, s_0, \mathcal{G} \rangle$, where $\mathcal{S}$ is a finite set of states; $\mathcal{A}$ is a finite set of actions; $\mathcal{T}(s'|s,a)$ is a stationary transition function specifying the probability of going to state s' when action a is executed in state s; $\mathcal{C}(s,a)$ is a stationary cost function that gives the non-negative cost of executing action a in state s; $s_0 \in \mathcal{S}$ is a given start state; and $\mathcal{G} \subset \mathcal{S}$ is a set of absorbing goal states. Additionally, we assume that $\mathcal{C}(s,a) > 0$ for any state $s \notin \mathcal{G}$.

A solution to an MDP is a *policy*, a mapping $\pi : \mathcal{S} \to \mathcal{A}$, indicating that action $\pi(s)$ should be taken at state s. A policy π induces a value function $V^\pi : \mathcal{S} \to \mathbb{R}$ that represents the expected cumulative cost of reaching $s_g \in \mathcal{G}$ by following policy π from state s. An optimal policy, π^*, is one that minimizes this expected cumulative cost. We also use the notation V^* to refer to the optimal value function.

For an MDP to be well-defined, a policy must exist such that the goal is reachable from any state (reachable from s_0) with probability 1 (i.e., there must be a *proper* policy). Under this assumption, an MDP is guaranteed to have an optimal solution, and the optimal value function is unique. This optimal value function can then be found as the fixed point of the Bellman update operator:

$$V^*(s) = \min_a [\mathcal{C}(s,a) + \sum_{s' \in \mathcal{S}} \mathcal{T}(s'|s,a) V^*(s')]. \quad (1)$$

An optimal policy can be extracted from $V^*(s)$ by choosing greedily with respect to this value function.

Solving MDPs optimally is often intractable (Littman, 1997), which has led to the development of many approximate algorithms, often based on value function approximation

1. While we focus on SSPs for our presentation, the ideas described in this work can also be directly applied to non goal-based discrete MDPs. Hence, we will use the term MDP throughout this work.

methods. As indicated above, we focus here on extending recent paradigms to handle the complexity of MDPs based on determinization—an online planning strategy in which deterministic versions of the MDP are repeatedly constructed and solved using a classical planner. Next, we define a space of MDP reductions that generalizes the idea of determinization.

2.2 A Broad Spectrum of MDP Model Reductions

We propose a new family of MDP reduced models that are characterized by two key parameters: the number of outcomes per action that are fully accounted for, and the maximum number of occurrences of the remaining outcomes that are planned for in advance. Building on our earlier work (Pineda & Zilberstein, 2014), we refer to the first set of outcomes as *primary outcomes* (those that will be fully accounted for) and to the remaining outcomes as *exceptional outcomes*.

We consider factored representations of MDPs—such as PPDDL (Younes et al., 2005)—in which actions are represented as probabilistic operators of the form:

$$a = \langle prec, cost, [p_1^a : e_1^a, ..., p_m^a : e_m^a] \rangle,$$

where *prec* is a set of conditions necessary for the action to be executed, *cost* is the cost of the action (assumed to be the same in all states), and for each $i \in \{1, ..., m\}$, p_i^a is the probability of outcome e_i^a occurring when the action is executed. The transition function can be recovered from this representation by means of a function τ that maps outcomes to successor states, so that $s' = \tau(s, e_i^a)$ and $\mathcal{T}(s, a, s') = p_i^a$. Note that typical MDP representations, like PPDDL, model actions as parameterized action *schemata*, each of which declares a function from objects to a *grounded* action. We formalize our framework at the level of grounded actions, although we expect that, in practice, reducing the problem at the schema level will be more practical (and it is the approach we use in our experiments).

For any action a, let $\mathcal{P}_a \subseteq \{e_1^a, ..., e_m^a\}$ be the set of its primary outcomes. Given sets $\mathcal{P}_a$ for each action $a \in \mathcal{A}$, we define a reduced version of an MDP that accounts for a bounded number of occurrences of exceptional outcomes, which we refer to as *exceptions*. Note that an exception is any effect that belongs to $\{e_1^a, ..., e_m^a\} \setminus \mathcal{P}_a$.

Formally, a **reduced model** of an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{C}, s_0, \mathcal{G} \rangle$ is another MDP, $M = \langle \mathcal{S}', \mathcal{A}, \mathcal{T}', \mathcal{C}', s_0', \mathcal{G}' \rangle$, where

- The set of states is defined as $\mathcal{S}' \triangleq \mathcal{S} \times \{0, 1, ..., k\}$, where k is a positive integer;
- The set of actions is the original set, $\mathcal{A}$;
- The transition function is defined by Eqs. (2), (3) and (4) below;
- The cost function is defined as $\mathcal{C}'(\langle s, j \rangle, a) \triangleq \mathcal{C}(s, a)$, for all $\langle s, j \rangle \in \mathcal{S}' \wedge a \in \mathcal{A}$;
- The initial state is $s_0' \triangleq \langle s_0, k \rangle$;
- The set of goals is defined as $\mathcal{G}' \triangleq \{\langle s, j \rangle \in \mathcal{S}' | s \in \mathcal{G}\}$.

The transition function $\mathcal{T}'$ of the augmented MDP is defined as follows. Given a state $\langle s, j \rangle$, the counter j represents the maximum number of exceptions, per trajectory,

that will be accounted for by the planner when computing a plan for $\langle s, j\rangle$. When $j=0$, the reduced model assumes that no more exceptions can occur, so the new transition function is:

$$\forall s, a, s' \quad \mathcal{T}'(\langle s, j\rangle, a, \langle s', j'\rangle) \triangleq \begin{cases} p'_i & e_i^a \in \mathcal{P}_a \ \wedge \ j' = j = 0 \\ 0 & e_i^a \notin \mathcal{P}_a \ \wedge \ j' = j = 0 \end{cases}, \tag{2}$$

where we use the shorthand $s' = \tau(s, e_i^a)$ and the set $\{p'_1, ..., p'_m\}$ is any set of real numbers that satisfy

$$\forall i : \left(e_i^a \in \mathcal{P}_a \ \Rightarrow \ p'_i > 0\right) \ \wedge \ \sum_{i:e_i^a \in \mathcal{P}_a} p'_i = 1. \tag{3}$$

For states $\langle s, j\rangle$ with $j>0$, the full transition model is used, and the exception counter is updated appropriately if an exception occurs. Thus, the transition function in this case becomes:

$$\forall s, a, s', j, j' \quad \mathcal{T}'(\langle s, j\rangle, a, \langle s', j'\rangle) \triangleq \begin{cases} p_i^a & e_i^a \in \mathcal{P}_a \ \wedge \ j' = j \\ p_i^a & e_i^a \notin \mathcal{P}_a \ \wedge \ j' = j - 1 \\ 0 & \text{otherwise} \end{cases}. \tag{4}$$

Note that while the complete state space of a reduced MDP is actually larger than that of the original problem, the benefit of the reduction is that, for well-chosen values of k and sets $\mathcal{P}_a$, the set of *reachable states* can become much smaller. This is desirable because the runtime of heuristic search algorithms for solving MDPs such as LAO* (Hansen & Zilberstein, 1998, 2001) and LRTDP (Bonet & Geffner, 2003) depends heavily on the size of the reachable state space. Furthermore, by changing k and the maximum size of the sets $\mathcal{P}_a$, we can adjust the amount of uncertainty we are willing to ignore in order to have a smaller reduced problem. Figure 1 illustrates the pruning effect that can be achieved with a reduced model.

Building on the formulation presented above, the following definition formalizes the concept of $\mathcal{M}_l^k$-reductions.

Definition 1 ($\mathcal{M}_l^k$-reduction of an MDP). *An $\mathcal{M}_l^k$-reduction of an MDP is an augmented MDP with the transition function defined by Eqs. (2), (3), and (4), where $j \in \{0, 1, ..., k\}$ and $\forall a \ |\mathcal{P}_a| \leq l$.*

For example, the single-outcome determinization used in the original FF-REPLAN (Yoon et al., 2007) is an instance of an $\mathcal{M}_1^0$-reduction where each set $\mathcal{P}_a$ contains the single most likely outcome of the corresponding action a.

Note that for any given values of k and l there might be more than one possible $\mathcal{M}_l^k$-reduction. We introduce the notation $M \in \mathcal{M}_l^k$ to indicate that M is some instance of an $\mathcal{M}_l^k$-reduction; different instances are characterized by two choices. One is the set of specific outcomes that will be labeled primary. The other is how to distribute the probability of the exceptional outcomes among the primary ones when $j = 0$—i.e., the choice of p'_i in Eq. (2). In this work we simply normalize the probabilities of the primary outcomes so that they sum up to one. However, more complex ways to redistribute the probabilities of exceptional outcomes are possible.

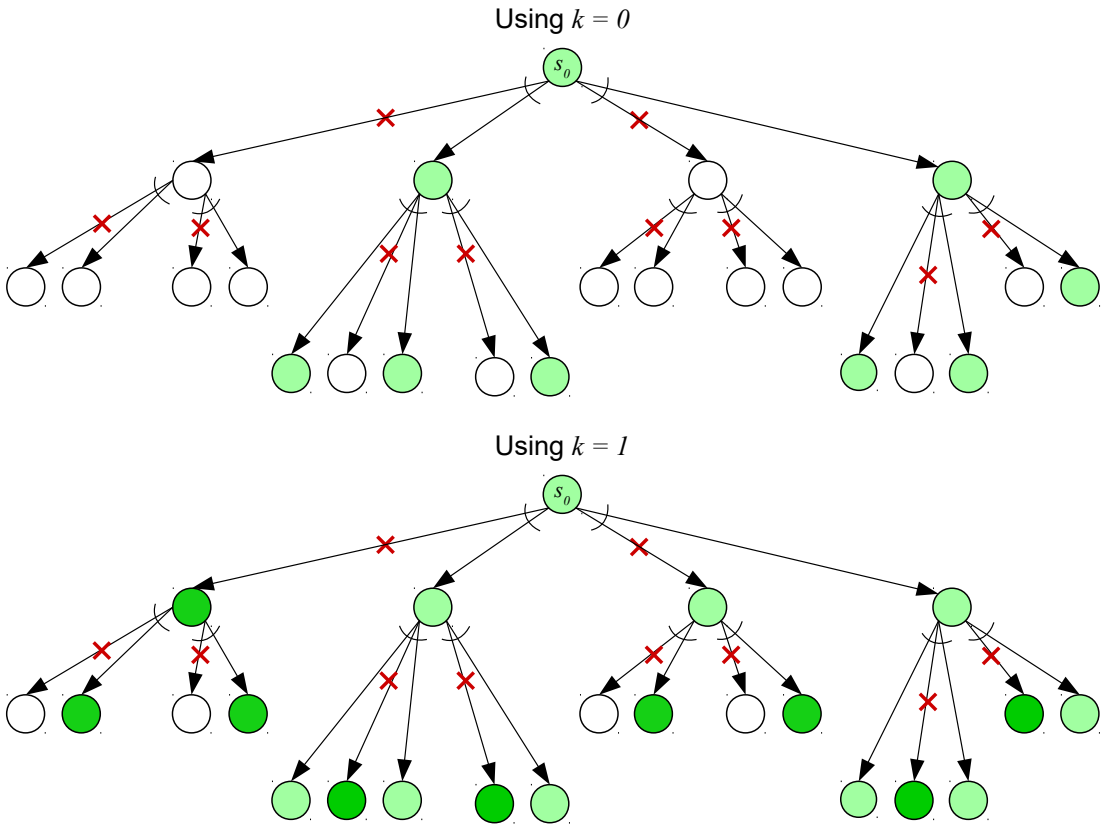

Figure 1: Illustration of the pruning effect of an $\mathcal{M}_l^k$-reduction, using two different values of k. Exceptional outcomes are marked with a red cross and reachable states are highlighted in green (darker color for those reachable with $k = 1$ but not $k = 0$). The value of k can be used to regulate the trade-off between computational efficiency and plan robustness.

The concept of $\mathcal{M}_l^k$-reductions raises a number of critical questions about its potential benefits in planning:

1. How should we assess the comprehensive value of an $\mathcal{M}_l^k$-reduction? Can this be done analytically?

2. Considering the space of $\mathcal{M}_l^k$-reductions, is determinization (i.e., $\mathcal{M}_1^0$-reduction) always preferable?

3. In the space of possible determinizations, can the best ones be identified using a simple heuristic (e.g., choosing the most likely outcome)? Or do we need more sophisticated value-based methods for that purpose?

4. How can we explore efficiently the space of $\mathcal{M}_l^k$-reductions? How can we find good ones or the best one?

In later sections we answer these questions, showing evidence that an $\mathcal{M}_1^0$-reduction (i.e., a single-outcome determinization) is not always desirable. Furthermore, even when determinization can provide good (or even optimal) performance, a value-based approach is needed to choose the most appropriate primary outcome per action. However, before we can attempt to answer these questions, we need a way to evaluate the benefits of a particular $\mathcal{M}_l^k$-reduction. In the next section we show how, given k and sets $\mathcal{P}_a$ for all actions in the original MDP, we can evaluate analytically the expected cost of solving the original problem using plans derived by solving the reduced problem.

3. Planning for More than k Exceptions

A plan generated using a reduction $M \in \mathcal{M}_l^k$ is likely to be incomplete because more than k exceptions could occur during plan execution, leading to a state that isn't included in the plan. Hence, we propose a *continual planning* approach that takes advantage of the added robustness of reduced model plans, such that it can handle a limited number of exceptions and thereby facilitate uninterrupted plan execution.

The terms *continuous planning* and *continual planning* generally refer to system architectures in which plan generation and plan execution are integrated and performed concurrently, in contrast to the more traditional plan-then-execute paradigm (Brenner & Nebel, 2009; Chien, Knight, Stechert, Sherwood, & Rabideau, 2000; desJardins, Durfee, Ortiz, & Wolverton, 1999; Myers, 1999; Pineda, Takahashi, Jung, Zilberstein, & Grupen, 2015). Building on these early efforts, our goal is to introduce a continual planning approach for solving MDPs that is amenable to an analytical evaluation and could provide performance guarantees. In contrast, early work on continual planning often resulted in complex planning and execution architectures that are hard to analyze from a theoretical perspective.

To this end, we propose a continual planning strategy specifically designed for $\mathcal{M}_l^k$-reductions. A high-level version of this approach, named $\mathcal{M}_l^k$-REPLAN, is shown in Algorithm 1. We use the notation $\mathcal{P}$ to represent the choice of primary outcomes for a reduction; that is, a mapping[2] $\mathcal{P} : \mathcal{A} \rightarrow 2^{\{e_1^a,\ldots,e_m^a\}}$, relating each action to a set of primary outcomes. $\mathcal{M}_l^k$-REPLAN relies on function CREATE-REDUCED-MDP, which takes as input an MDP, $\mathcal{M}$, an initial state, s, the chosen primary outcomes, $\mathcal{P}$, and the exception counter, k; its output is the corresponding reduced MDP, with initial state s.

$\mathcal{M}_l^k$-REPLAN begins by creating a reduced model (line 1) and solving it optimally (see COMPUTE-OPTIMAL-PLAN in line 2). This plan is then executed (line 4), and whenever the exception counter reaches the lower bound, $j = 0$, the algorithm generates a new reduced model in line 6 (for reasons explained below) and an optimal plan for this reduced model (line 7). At this point in execution there will still be an action ready in the current plan, so we can compute the new plan while simultaneously executing an action from the existing plan. As long as the new plan is ready when the action finishes executing, plan execution will resume without delay. Action execution relies on function EXECUTE-ACTION, which receives the current state and an action, applies this action to the system, and returns the state reached after the action is executed, updating the exception counter appropriately.

2. This is a slight abuse of notation, since the set of possible outcomes $\{e_1^a, ..., e_m^a\}$ is indexed by action a. We do this in the interest of readability and make sure that the intended meaning is clear from the context in the rest of the paper.

Algorithm 1: $\mathcal{M}_l^k$-REPLAN: A continual planning approach for handling more than k exceptions

input: $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{C}, s_0, \mathcal{G} \rangle$, k, $\mathcal{P}$

1 $M \leftarrow$ CREATE-REDUCED-MDP$(\mathcal{M}, s_0, \mathcal{P}, k)$

2 $\pi \leftarrow$ COMPUTE-OPTIMAL-PLAN$(M, \langle s_0, 0 \rangle)$

3 $\langle s, j \rangle \leftarrow \langle s_0, 0 \rangle$

while $s \notin \mathcal{G}$ **do**

  if $j \neq 0$ **then**

4 $\langle s, j \rangle \leftarrow$ EXECUTE-ACTION$(\langle s, j \rangle, \pi(\langle s, j \rangle))$

  else

5 Create new state $\hat{s}$ with one zero-cost action $\hat{a}$ s.t. $\forall s' \in \mathcal{S}\colon Pr(\langle s', k \rangle | \hat{s}, \hat{a}) = \mathcal{T}(s' | s, \pi(\langle s, j \rangle))$

6 $M \leftarrow$ CREATE-REDUCED-MDP$(\mathcal{M}, \hat{s}, \mathcal{P}, k)$

  do in parallel

7 $\pi' \leftarrow$ COMPUTE-OPTIMAL-PLAN$(M, \hat{s})$

8 $\langle s, j \rangle \leftarrow$ EXECUTE-ACTION$(\langle s, j \rangle, \pi(\langle s, j \rangle))$;

9 $\pi \leftarrow \pi'$

10 $\langle s, j \rangle \leftarrow \langle s, k \rangle$

Note that, after re-planning, the algorithm sets the exception counter of the current state to $j = k$, since the new plan can handle up to k additional exceptions.

There is one complication in this continual planning process. Since the new plan will be activated from a start state that is not yet known (when the planning process starts), all the possible start states need to be taken into account, including those reached as a result of another exception. Therefore, we create a new dummy start state (line 5) that leads via a single zero-cost action to all the start states we may encounter when the execution of the current action terminates; we then create a new reduced model using the dummy state as initial state (line 6).

For the sake of clarity of the algorithm and its analysis, we described a straightforward implementation where the execution time of one action is sufficient to generate a plan for the reduced model. When planning requires more time, it may delay the execution of the new plan. In Section 3.2, we introduce a variant of Algorithm 1 that is designed to address this specific issue.

3.1 Evaluating the Performance of the Continual Planning Approach

Unlike existing continual planning methods (Chanel, Lesire, & Teichteil-Königsbuch, 2014), the proposed approach facilitates a precise analytical evaluation of reduced models. Let π_k be a *universal plan* (Schoppers, 1987) for a reduced model $M \in \mathcal{M}_l^k$—one that covers every possible state of the reduced model M. While universal planning is considered impractical in large domains (Ginsberg, 1989), we are using it here to propose an offline technique to

evaluate the performance of the continual planning method in hindsight; finding π_k is not needed when solving a given problem instance using Algorithm 1.

While the continual planning and execution algorithm does not generate a universal plan, we observe that it always executes actions that agree with π_k as it conforms to the following rule: whenever it reaches a state $\langle s, 0\rangle$ (in which no more exceptions will be considered), it executes $\pi_k(\langle s, 0\rangle)$ and, if the outcome state is s' (as a result of either a primary or exceptional outcome), it moves to state $\langle s', k\rangle$ (of the newly generated plan) and executes $\pi_k(\langle s', k\rangle)$. This is essentially what the continual planning process does, by producing online a new partial plan for any outcome of the last action according to the previous plan.

More formally, this planning and execution approach generates a trajectory of the following Markov chain defined over states of the form $\langle s, j\rangle$, with initial state $\langle s_0, k\rangle$ and the following transition function, for any $s \in S, 0 \leq j \leq k$:

$$\forall s, j, s', j' \quad Pr(\langle s', j'\rangle | \langle s, j\rangle) = \begin{cases} \mathcal{T}'(\langle s, j\rangle, \pi_k(\langle s, j\rangle), \langle s', j'\rangle) & j > 0 \\ \mathcal{T}(s, \pi_k(\langle s, j\rangle), s') & j = 0 \ \wedge \ j' = k \\ 0 & \text{otherwise} \end{cases}.$$

The middle case represents the transition from $\langle s, 0\rangle$ to (s', k), which also indicates the transition to a new plan. Let V_{cp}^M denote the value function defined over this *continual planning Markov chain* with respect to a given reduction M. Then we have:

Proposition 1. *$V_{cp}^M(\langle s_0, k\rangle)$ provides the expected value of the continual planning and execution approach for a given reduced model M, when the plan is executed in the original (not reduced) problem domain.*

Existing continual planning methods often involve heuristic decisions about the interleaving of planning and execution, making it necessary to evaluate them empirically. The ability to derive an exact expected value for the proposed planning and execution approach makes it easier to compare different reduced models, knowing that the expected value is not biased by the sampling method.

3.2 An Anytime Version of $\mathcal{M}_l^k$-REPLAN

The structure of $\mathcal{M}_l^k$-reductions can also be leveraged to produce an *anytime* planner; that is, a planner that can be interrupted at any point and still have an action ready, producing better plans with larger allowances of planning time. We refer to this anytime variant of $\mathcal{M}_l^k$-REPLAN as $\mathcal{M}_l^k$-ANYTIME, which is outlined in Algorithm 2. As in $\mathcal{M}_l^k$-REPLAN, $\mathcal{M}_l^k$-ANYTIME considers planning in parallel to action execution. However, this new anytime version does not assume that the execution time of an action is sufficient to generate a new plan. Instead, the planner can be preempted whenever an action is requested (line 7), and the agent always chooses an action greedily based on the most recent value estimates (line 4). Note that, as in the case of $\mathcal{M}_l^k$-REPLAN, the algorithm tries to create a plan for the states that can be encountered *after* the current action is executed (line 5). But, unlike $\mathcal{M}_l^k$-REPLAN, the anytime version plans during the execution of *every* action (as opposed to only when $j = 0$), and therefore it always uses the greedy action corresponding to $\langle s, k\rangle$.

Algorithm 2: $\mathcal{M}_l^k$-ANYTIME

input: $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{C}, s_0, \mathcal{G} \rangle$, k, $\mathcal{P}$, τ

1 $M \leftarrow$ CREATE-REDUCED-MDP$(\mathcal{M}, s_0, \mathcal{P}, k)$

2 PLAN-FOR-LIMITED-TIME$(M, \langle s_0, k \rangle, \tau)$

3 $s \leftarrow s_0$

while $s \notin \mathcal{G}$ **do**

4 $\quad a \leftarrow$ GREEDY-ACTION$(M, \langle s, k \rangle)$

5 $\quad$ Create new state $\hat{s}$ with one zero-cost action $\hat{a}$ s.t. $\forall s' \in \mathcal{S}$: $Pr(\langle s', k \rangle | \hat{s}, \hat{a}) = \mathcal{T}(s'|s, a)$

6 $\quad M \leftarrow$ CREATE-REDUCED-MDP$(\mathcal{M}, \hat{s}, \mathcal{P}, k)$

$\quad$ **do in parallel**

7 $\quad\quad$ PLAN-UNTIL-PREEMPTED$(M, \langle s, k \rangle)$

8 $\quad\quad s \leftarrow$ EXECUTE-ACTION(s, a)

4. The Choice of Reduced Model Matters

In this section we show evidence that a careful choice of reduced model can result in policies that have significantly better cost than policies generated by popular determinization-based approaches. First, in Section 4.1, we show how this can be accomplished by planning with more than one primary outcome (i.e., going beyond single-outcome determinization), but without accounting for the full original model. Second, in Section 4.2, we show that, in some problems, the choice of primary outcome has a large impact in the quality of resulting plans, even when using only a single-outcome determinization for planning.

4.1 The Value of Going Beyond Single-Outcome Determinization

The defining property of most determinization-based algorithms is the use of fully deterministic models in planning, entirely ignoring what we call exceptional outcomes. In fact, even the widely used all-outcomes determinization treats each probabilistic outcome as a fully deterministic one, completely ignoring the relationship between outcomes of the same action. Hence, we argue that an $\mathcal{M}_1^0$-reduction is not always desirable, and that an $\mathcal{M}_l^0$-reduction with $l > 1$ could be significantly better for some domains.

To illustrate this point—that determinization could sometimes lead to poor performance relative to other reduced models—we use a modified version of the racetrack domain (Barto et al., 1995), a well-known reinforcement learning benchmark. The problem involves a simulation of a race car on a discrete track of some length and shape, where a starting line has been drawn on one end and a finish line on the opposite end of the track. The state of the car is determined by its location and its two-dimensional velocity. The car can change its speed in each dimension by at most 1 unit, giving a total of nine possible actions. After applying an action there is a probability p_{slip} that the resulting acceleration is zero, simulating failed attempts to accelerate/decelerate because of unpredictably slipping on the track. Additionally, we include a probability p_{er} that the resulting acceleration is off by one dimension w.r.t. the intended acceleration. The goal is to go from the start line to the finish line in as few moves as possible.

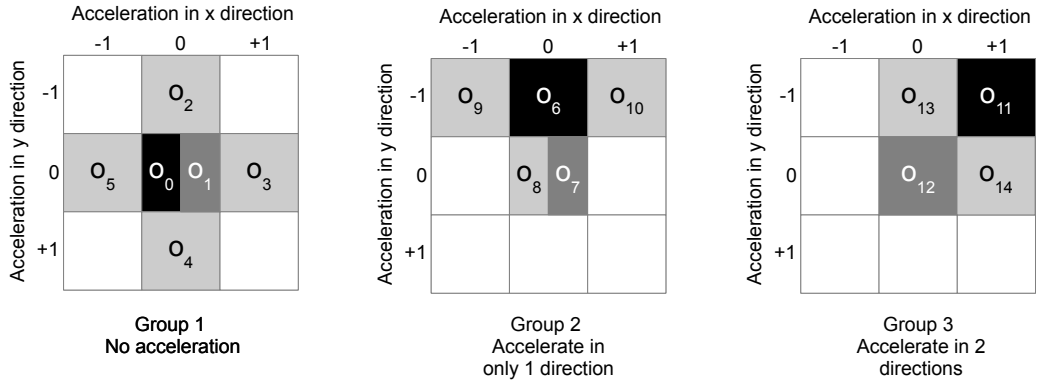

Figure 2: Action groups in the racetrack domain: dark squares represent the intended action, gray squares represent the acceleration outcome associated with slipping, and the light gray squares represent the remaining outcomes.

	$\lvert\mathcal{S}\rvert$	$V_{cp}^{M_1}\langle s_0, 0\rangle$	$V_{cp}^{M_2}\langle s_0, 0\rangle$	$\hat{V}^{ao}(s_0)$
small	239	9.22%	5.40%	126.8%
medium	2219	28.18%	7.42%	118.6%
large	24587	48.91%	11.53%	102.8%

Table 1: Comparison of the best determinization (M_1) and the best $\mathcal{M}_2^0$-reduction (M_2) for three racetrack problems.

To decrease the number of reductions to consider, instead of treating the outcomes of all nine actions separately, we can group symmetrical actions and apply the same reduction to all actions in the same group. The racetrack domain has three groups of symmetric actions: four actions that accelerate/decelerate in both directions, four actions that accelerate/decelerate in only one direction, and one action that keeps the current speed. Figure 2 illustrates these groups of actions and their possible outcomes; for each group, a decomposition is specified by the set of outcomes, relative to the intended outcome (shown in darker color), that are labeled as primary. In our experiments we used three racetrack problems of different sizes (see Figure 3).

We compared the following two reductions, M_1 and M_2:

$$M_1 = \min_{M \in \mathcal{M}_1^0} V_{cp}^{M}(\langle s_0, k\rangle) \quad \text{and} \quad M_2 = \min_{M \in \mathcal{M}_2^0} V_{cp}^{M}(\langle s_0, k\rangle);$$

that is, we compared the best possible $\mathcal{M}_1^0$-reduction (determinization) of this problem, with its best possible $\mathcal{M}_2^0$-reduction. For reference, we also report the expected cost (estimated using 1000 simulations) of a policy obtained with an all-outcomes determinization of the problem; we denote this cost as $\hat{V}^{ao}(s_0)$).

Table 1 shows the increase in cost of these reductions with respect to the optimal expected cost obtained by solving using the full model. In all of the three tracks considered,

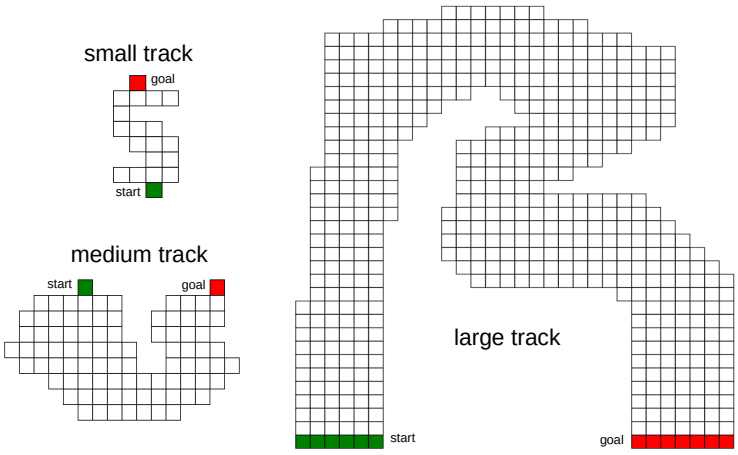

Figure 3: Three instances of the racetrack domain.

Primary outcome	**P01**	**P02**	**P03**	**P04**	**P05**	**P06**	**P07**	**P08**	**P09**	**P10**
(not (not-flattire))	50	50	50	50	50	50	50	50	50	28
(not-flattire)	30	10	4	0	0	0	0	0	0	0

Table 2: Number of successful trials, out of a maximum of 50, using two different $\mathcal{M}_1^0$-reductions on ten TRIANGLE-TIREWORLD problems.

the use of single-outcome determinization resulted in a 9% or higher increase in cost, while the maximum cost increase for the best $\mathcal{M}_2^0$-reduction was less than 12% in all cases. Additionally, note that using an all-outcome determinization, which cannot be represented as an $\mathcal{M}_l^k$-reduction, results in particularly poor performance in this domain. To see why, consider that under the error model considered in this example, the no-acceleration action includes a unique low probability outcome for each possible direction the agent can move to. Thus, for instance, a planner based on the all-outcomes determinization can potentially choose a plan that always decides not to accelerate, since this plan has a non-zero probability of reaching a goal. Admitedly, there are techniques that can alleviate this issue (e.g., increasing the cost of actions associated to low probabity outcomes), but the goal of the previous analysis is to highlight the importance of the choice of reduced model, *ceteris paribus*.

4.2 Choosing the Right Outcomes

In some problems determinization works well. That is, the cost of using continual planning with the *best* $\mathcal{M}_1^0$-reduction may be close to the optimal cost V^*. However, the choice of primary outcomes by simply inspecting the domain description may still present a non-trivial challenge. For example, the commonly used most-likely-outcome heuristic may not work well.

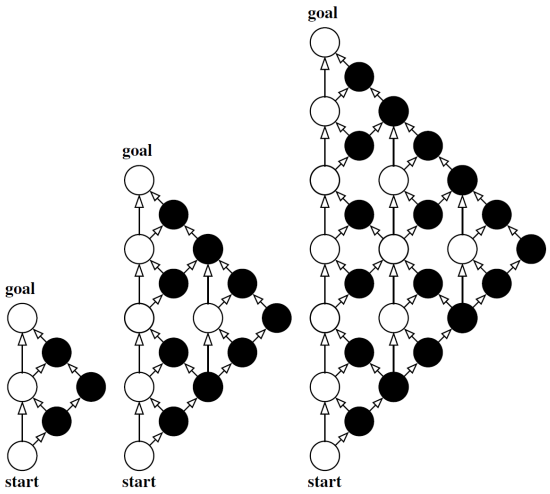

Figure 4: Three instances of the TRIANGLE-TIREWORLD domain. Locations with spare tires are marked in black (Little & Thiébaux, 2007).

To illustrate this issue we experimented with different determinizations of the TRIANGLE-TIREWORLD domain (Little & Thiébaux, 2007). This problem involves a car traveling between locations on a graph shaped like a triangle (see Figure 4). Every time it moves there is a certain probability of getting a flat tire when the car reaches the next location (60% in the experiments in this section), but only some locations include a spare tire that can be used to repair the car. Note that, since the car cannot change its location when it has a flat tire, this domain has dead-ends. We address this issue using a well-known technique for planning in this type of problem. In particular, we use a cap on state costs, $\mathcal{D}$, and modify the Bellman backup operator as follows

$$V(s) = \min\Big\{\mathcal{D}, \min_{a\in\mathcal{A}}\Big\{\mathcal{C}(s,a) + \sum_{s'\in\mathcal{S}} \mathcal{T}(s,a,s')V(s')\Big\}\Big\},$$

which guarantees the convergence of heuristic search algorithms (Kolobov, Mausam, & Weld, 2012).

This domain has two possible determinizations, depending on whether getting a flat tire is considered an exception or a primary outcome. Table 2 shows the results (number of trials reaching the goal) of evaluating the two determinizations on 10 instances of this domain, using LAO* as the underlying optimal planner (Hansen & Zilberstein, 2001). The best determinization is undoubtedly the one in which getting a flat tire is considered the primary outcome. The resulting plan enabled the car to reach the goal in most of the large majority of simulated rounds (from a maximum of 50 rounds to be solved within a 20 minutes time limit), while the other determinization resulted in complete failure to reach the goal for (P04 ... P10). Note that the failure of the best planner in problem P10 was solely due to it not having enough time to find an optimal plan.

As it turns out, the right determinization for this problem is not very intuitive, as one typically expects for primary outcomes to correspond to the most likely outcome of an

action or to its intended outcome when it succeeds (the most likely outcome is not having a flat tire with probability 60%.) This counterintuitive result might lead one to consider the use of conservative heuristics, labeling the worst-case outcome as primary. Although this would indeed work very well in the TRIANGLE-TIREWORLD domain, it would perform poorly in other domains such as the racetrack problem. Additionally, note that an all-outcomes determinization does not work well in this problem either, as our experimental results with FF-REPLAN (Section 7.2) show.

To sum up, some determinizations can indeed result in optimal performance, but there seems to be no all-purpose "rule of thumb" to choose the best one. This suggests that a more principled value-based approach is needed in order to find a good determinization or a good reduction in general. We explore two different strategies for choosing reduced models in Section 6.

5. Algorithms that Directly Leverage $\mathcal{M}_l^k$-Reductions

So far, we have assumed that $\mathcal{M}_l^k$-reductions will be solved optimally using a regular MDP solver, such as LAO*. However, in some cases it is possible to leverage the structure of a reduction even further in order to devise more efficient planning algorithms. In this section we present one such algorithm, specifically tailored to $\mathcal{M}_1^k$-reductions, which employs a classical planner to accelerate computation for states in which no more exceptions will be considered, i.e., in which the exception counter $j = 0$. The main advantage of using determinization is that it makes it possible to leverage highly efficient classical planners for the solution of probabilistic problems. As it turns out, we can also incorporate this approach into our reduced models framework, particularly when using $\mathcal{M}_1^k$-reductions. Note that a $\mathcal{M}_1^k$-reduction becomes a deterministic problem for any state with exception counter $j = 0$. Thus, a classical planner can be used for solving the deterministic parts of the augmented state space.

To illustrate this idea, we describe a modified version of LAO* that leverages the FF classical planner (Hoffmann & Nebel, 2001). This solver, FF-LAO* (Algorithms 3-6), receives as input an $\mathcal{M}_1^k$-reduction, $M = \langle \mathcal{S}', \mathcal{A}, \mathcal{T}', \mathcal{C}', \langle s_0, k\rangle, \mathcal{G}'\rangle$—i.e., one where $\forall a \in \mathcal{A}, |\mathcal{P}_a| = 1$; an exception bound, k; and an error tolerance, ϵ. We use $\mathbb{M}$ to denote the original MDP from which M is derived.

FF-LAO* works almost exactly as LAO*[3], except that FF is used to compute values and actions for states that have reached the exception bound—i.e., states of the form $\langle s, 0\rangle$. This occurs in lines 4 and 8 of Algorithm 3, where the state expansion and test convergence procedures are replaced with versions that use FF (Algorithms 4 and 5, respectively). Readers familiar with LAO* may notice differences with respect to the usual expansion and convergence test procedures. In particular, note the inclusion of *if* statements in line 7 (both procedures), where the successors of the expanded state are only added to the stack if $j > 0$. The reason is that states $\langle s, 0\rangle$ will be solved by calling FF, so there is no need to expand their successors. It is possible, of course, to remove these *if* statements and let FF-LAO* continue the search; in that case, FF will be used as an inadmissible heuristic.

3. We use the so-called *improved* LAO* algorithm, where the greedy solution graph is searched in depth-first fashion, and Bellman backups are performed in post-order traversal, both for the state expansion step and the convergence test step.

However, this does not improve the theoretical properties of the algorithm (neither version is optimal, due to the use of FF), and results in higher computation times, so we prefer the version shown in the pseudocode.

The actual call to FF is done in Algorithm 6 (FF-BELLMAN-UPDATE). This procedure performs a Bellman update, as in Eq. (1), for any state $\langle s, j\rangle$ with $j > 0$, and stores the updated cost estimate and best action in global variables $V[\langle s, j\rangle]$ and $\pi[\langle s, j\rangle]$, respectively (lines 6-7). We assume, as is common for heuristic search algorithms, that the values $V[\langle s, j\rangle]$ are initialized using an admissible heuristic for M.

For states $\langle s, 0\rangle$, the FF-BELLMAN-UPDATE procedure creates a PDDL file[4], denoted as D, representing the deterministic problem induced by M when $j = 0$, with initial state s (CREATE-PDDL in line 3). The procedure then calls FF with input D (line 4) and memoizes costs and actions for all the states visited in the plan computed by FF (lines 5-7). More concretely, for each state s_i visited by this plan, we set $V[\langle s_i, 0\rangle]$ to be the cost, according to C', of the plan computed by FF for that state (line 6), and set $\pi[\langle s_i, 0\rangle]$ to be the corresponding action (line 7). Additionally, note that the estimates $V[\langle s, 0\rangle]$ are not admissible, even with respect to the input $\mathcal{M}_1^k$-reduction, since FF is not an optimal planner for deterministic problems. Finally, in the case that FF returns failure, we set $V[\langle s, 0\rangle] = \infty$ and $\pi[\langle s, 0\rangle] = \text{NOP}$.

FF-BELLMAN-UPDATE also returns the residual, defined as the absolute difference between the previous cost estimate, and the estimate after applying the Bellman equation. This residual is used by FF-TEST-CONVERGENCE to check the stopping criterion of the algorithm.

Handling plan deviations during execution For the experiments with FF-LAO* we use a slightly different continual planning approach than the one described in Section 3; this new approach is illustrated in Algorithm 7. The idea is simple: during execution, check if the current state has an action already computed with $j = k$. If that is the case, this action is executed (line 7). Otherwise, FF-LAO* is called to solve the reduced model with initial state $\langle s, k\rangle$ (lines 5-6). FF-LAO*-REPLAN receives the choice of determinization as input ($\mathcal{P}$), and creates an $\mathcal{M}_1^k$-reduction accordingly (line 1).

Theoretical considerations We now show conditions under which FF-LAO* is guaranteed to succeed. The following definition will be useful: a *proper policy rooted at s* is one that reaches a goal state with probability 1 from every state it can reach from s.

Proposition 2. *Given an admissible heuristic for the reduced model M, if M has at least one proper policy rooted at $\langle s_0, k\rangle$, then FF-LAO* is guaranteed to find one in finite time.*

Proof. Whenever FF-LAO* expands a state $\langle s, 0\rangle$ and calls FF on this state, if the call succeeds, the states s_i, for $i \in [1, ..., L]$, that are part of the plan computed by FF essentially become terminal states of the problem, with final costs set as in line 6 of Algorithm 6; thus, this induces a new MDP in which the states $s_i, s_{i+1}, ..., s_L$ are additional goals. Since FF is a sub-optimal planner, we have that $\sum_{i \leq x \leq L} C'(\langle s_x, 0\rangle, a_i) \geq V[\langle s_i, 0\rangle]$, and thus the values of all other states $\langle s, j\rangle$, with $j > 0$, are guaranteed to be admissible with respect to the new updated value of the added terminal states. In other words, the current value function is

4. In practice, we create the PDDL file representing M before calling FF-LAO* and store its name in memory. CREATE-PDDL is shown for simplicity of presentation.

Algorithm 3: FF-LAO*

input: $M = \langle S', A, T', C', \langle s_0, k\rangle, G'\rangle$, k, ϵ

```
while true do
    // Node expansion step
    while true do
        visited ← ∅
        cnt ← FF-EXPAND(M, ⟨s_0, k⟩, k, visited)
        if cnt = 0 then
            // No tip nodes were expanded, so current policy is closed
            break
    // Convergence test step
    while true do
        visited ← ∅
        error ← FF-TEST-CONVERGENCE(M, ⟨s_0, k⟩, k, visited)
        if error < ε then
            return // solution found
        if error = ∞ then
            break // change in partial policy, go back to expansion step
```

Algorithm 4: FF-EXPAND

input: $M = \langle S', A, T', C', \langle s_0, k\rangle, G'\rangle$, $\langle s, j\rangle$, k, *visited*

```
if ⟨s, j⟩ ∈ visited then
    return 0
visited ← visited ∪ {⟨s, j⟩}
cnt ← 0
if π[⟨s, j⟩] = ∅ then
    // Expand this state for the first time
    FF-BELLMAN-UPDATE(M, ⟨s, j⟩, k)
    return 1
else if j > 0 then
    forall ⟨s', j'⟩ s.t. T'(⟨s', j'⟩|⟨s, j⟩, π[⟨s, j⟩]) > 0 do
        cnt ← cnt + FF-EXPAND(M, ⟨s, j⟩, k, visited)
FF-BELLMAN-UPDATE(M, ⟨s', j'⟩, k)
return cnt
```

admissible with respect to new MDP induced by the solution found by FF. Therefore, after every successful call to FF, the resulting set of values and terminal states form a well-defined SSP, which LAO* is able to solve.

Moreover, in the case that a call to FF fails for some state $\hat{s}$, this state will be assigned an infinite cost, and thus the improved version of LAO* will avoid $\hat{s}$ as long as there is some other path to the goal. Because FF is complete, any state belonging to a proper policy will

Algorithm 5: FF-TEST-CONVERGENCE

input: $M = \langle S', A, T', C', \langle s_0, k\rangle, G'\rangle$, $\langle s, j\rangle$, k, *visited*
if $\langle s, j\rangle \in$ *visited* **then**
  return 0
visited $\leftarrow$ *visited* $\cup \{\langle s, j\rangle\}$
error $\leftarrow 0$
$a \leftarrow \pi[\langle s, j\rangle]$
if $a = \emptyset$ **then**
  // The test reached a state that hasn't been expanded yet
  return ∞
else if $j > 0$ **then**
  forall $\langle s', j'\rangle$ *s.t.* $T'(\langle s', j'\rangle | \langle s, j\rangle, \pi[\langle s, j\rangle]) > 0$ **do**
    error $\leftarrow \max\big($*error*, FF-TEST-CONVERGENCE$\big(M, \langle s, j\rangle, k,$ *visited*$\big)\big)$
error $\leftarrow \max\big($*error*, FF-BELLMAN-UPDATE$\big(M, \langle s, j\rangle, k\big)\big)$
if $\pi[\langle s, j\rangle] \neq a$ **then**
  return ∞ *// the policy changed*
return *error*

Algorithm 6: FF-BELLMAN-UPDATE

input: $M = \langle S', A, T', C', \langle s_0, k\rangle, G'\rangle$, $\langle s, j\rangle$, k
output: *error*
$V' \leftarrow V[\langle s, j\rangle]$
if $j = 0$ **then**
  $D \leftarrow$ CREATE-PDDL(M, s)
  $\{s_1, a_1, s_2, a_2, ..., s_L, a_L\} \leftarrow$ CALL-FF(D)
  for $i \in \{1, ..., L\}$ **do**
    $V[\langle s_i, 0\rangle] \leftarrow \sum_{i \leq x \leq L} C'(\langle s_x, 0\rangle, a_i)$
    $\pi[\langle s_i, 0\rangle] \leftarrow a_i$
else
  $V[\langle s, j\rangle] \leftarrow \min_a C'(\langle s, j\rangle, a) + \sum_{\langle s', j'\rangle} T'(\langle s', j'\rangle | \langle s, j\rangle, a) V[\langle s', j'\rangle]$
  $\pi[\langle s, j\rangle] \leftarrow \arg\min_a C'(\langle s, j\rangle, a) + \sum_{\langle s', j'\rangle} T'(\langle s', j'\rangle | \langle s, j\rangle, a) V[\langle s', j'\rangle]$
return $|V[\langle s, j\rangle] - V'|$

be assigned a positive cost, so $\hat{s}$ could not have been part of a proper policy for M. Thus, under the conditions of the theorem, every call to FF transforms the problem into an MDP with avoidable dead-ends (Kolobov et al., 2012), which LAO* is able to solve. □

Unfortunately, as is the case for virtually all re-planning algorithms, not much can be guaranteed about the quality of plans found by FF-LAO*-REPLAN for $\mathbb{M}$. However, as we show in our experiments, by carefully choosing the input determinization, $\mathcal{P}$, and the

Algorithm 7: FF-LAO*-REPLAN

input: $\mathbb{M}=\langle S, A, T, C, s_0, G\rangle, \mathcal{P}, k, \epsilon$
$M \leftarrow$ CREATE-REDUCED-MDP$(\mathbb{M}, s_0, \mathcal{P}, k)$
$s \leftarrow s_0$
while $s \notin G$ **do**
  if $\langle s, k\rangle \notin \pi$ **then**
    $M \leftarrow$ CREATE-REDUCED-MDP$(\mathbb{M}, s, \mathcal{P}, k)$
    FF-LAO*(M, k, ϵ)
  $s \leftarrow$ EXECUTE-ACTION$(s, \pi[\langle s, k\rangle])$

bound k, FF-LAO*-REPLAN can find successful policies extremely quickly, even in domains well-known for their computational hardness and the presence of dead-end states.

6. Learning Good Reduced Models

In this section, we explore the problem of finding good reduced models to use for planning, with the ultimate objective of producing plans that perform close to optimal when evaluated on the original problem. We present two approaches, both of which learn a reduction on a small instance of a target domain, which can then be applied to larger instances of the domain.

6.1 A Greedy Approach for Learning Reduced Models

The first approach is a greedy algorithm for finding a model $M \in \mathcal{M}_l^k$ with a low cost $V_{cp}^M(\langle s_0, k\rangle)$ for some given k and l. The main premise of the approach is that problems in a given domain share some common structure, and that the relative performance of different $\mathcal{M}_l^k$-reductions generalizes across different problem instances. Although this is a strong assumption, experiments we report in Section 7 confirm that it works in practice.

Given k and l, every reduction $M \in \mathcal{M}_l^k$ is uniquely determined by the mapping $\mathcal{P}^M : A \to 2^{\{e_1^a, \ldots, e_m^a\}}$, which associates every action with the set of its primary outcomes. Since outcomes are indexed by the action they are associated to, this mapping can also be uniquely represented as a set $\mathcal{E}^M \equiv \bigcup_{a\in\mathcal{A}} \mathcal{P}^M(a)$, so that $e^a \in \mathcal{E}^M \implies e^a \in \mathcal{P}^M(a)$. Using this notation, finding a good $\mathcal{M}_l^k$-reduction amounts to solving the following combinatorial optimization problem:

$$\begin{array}{ll} \max\limits_{\mathcal{E}^M\subseteq\mathcal{E}} & -V_{cp}^M(\langle s_0, k\rangle), \quad \mathcal{E} \equiv \bigcup\limits_{a\in\mathcal{A}} \mathit{outcomes}(a) \\ \text{s.t.} & \forall a \in \mathcal{A},\ 1 \leq |\{e : e \in \mathcal{E}^M \cap \mathit{outcomes}(a)\}| \leq l. \end{array} \tag{5}$$

This optimization problem is particularly hard to solve due to two complications. First, it is possible that some reductions M introduce dead-ends even if the original MDP had none. This can happen, for example, if all the outcomes that can make progress towards the goal are outside the set of primary outcomes, and the only path towards the goal requires the occurrence of more than k of these outcomes. Second, as we show below, the maximized objective function is not *submodular* (Nemhauser, Wolsey, & Fisher, 1978), making it harder

to develop a bounded approximation scheme. A function $f : 2^W \to \mathbb{R}$, where W is a finite set, is submodular if for every $A \subseteq B \subseteq W$ and $e \in W \setminus B$, the following diminishing returns property holds (Krause & Golovin, 2014),

$$f(A \cup \{e\}) - f(A) \geq f(B \cup \{e\}) - f(B).$$

Submodular functions are attractive because they lead to good approximate greedy maximization algorithms. Unfortunately, as mentioned above, the objective function described by Eq. (5) is not submoduar, as the following proposition shows.

Proposition 3. *The function* $f(\mathcal{E}^M) \triangleq -V_{cp}^M(\langle s_0, k\rangle)$ *is not submodular.*

Proof. We provide an example contradicting submodularity, using the MDP shown in Figure 5. Consider two $\mathcal{M}_2^0$-reductions M_1 and M_2, where $\mathcal{E}^{M_1} = \{e_1^A, e_1^B, e_2^B\}$ and $\mathcal{E}^{M_2} = \{e_1^A, e_2^B\}$. It is not hard to see that $f(\mathcal{E}^{M_1}) = f(\mathcal{E}^{M_2}) = -51$, since both reductions result in action B being chosen, with a resulting expected cost of 51. Now consider adding outcome e_2^A to both $\mathcal{E}^{M_1}$ and $\mathcal{E}^{M_2}$. Let $\rho_e(S) = f(S \cup \{e\}) - f(S)$. Then we have $\rho_{e_2^A}(\mathcal{E}^{M_1}) = -19 + 51 = 31$, since action A is chosen under $\mathcal{E}^{M_1} \cup \{e_2^A\}$—i.e., the full model—with expected cost of 19, while $\rho_{e_2^A}(\mathcal{E}^{M_2}) = 0$, since action B is still chosen under $\mathcal{E}^{M_2} \cup \{e_2^A\}$. But this implies that $\rho_{e_2^A}(\mathcal{E}^{M_2}) < \rho_{e_2^A}(\mathcal{E}^{M_1})$ and $\mathcal{E}^{M_2} \subset \mathcal{E}^{M_1}$, which contradicts submodularity. □

Similar counterexamples can be constructed for larger values of k. Intuitively, lack of submodularity results because the benefit of adding a particular outcome to the reduction might not become evident unless some other outcomes had been previously added. Nevertheless, we have found empirical evidence that a simple greedy approach works well in practice, despite the difficulty in obtaining a bound with respect to optimal solution of the combinatorial optimization problem described in Eq. (5).

Our method, described in Algorithm 8, starts with M equal to the full probabilistic model, and iteratively removes from $\mathcal{E}^M$ the outcome e that minimizes $V_{cp}^{\hat{M}}(\langle s_0, k\rangle)$ (the expected cost of the induced continual planning approach); $\hat{M}$ represents the reduced model resulting after removing e from M. In the pseudo-code, the best outcome to remove is denoted as e_{best}^{α}, where α is the action this outcome is associated to. This process is continued as long as: (1) the maximum number of primary outcomes is larger than the desired l (lines 19 and 21), and (2) the relative increase in expected cost with respect to the value of the full model is lower than some threshold (line 21).

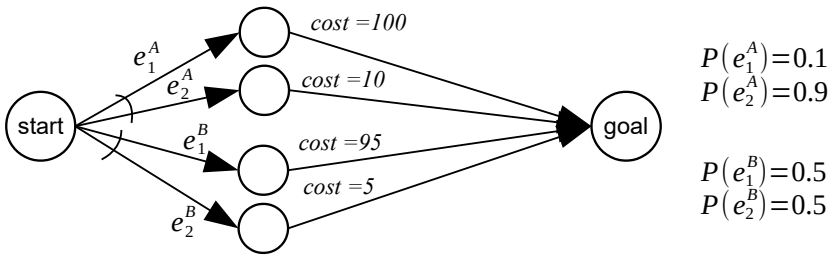

Figure 5: Example showing that $-V_{cp}^M(\langle s_0, k\rangle)$ is not submodular. Actions A and B have cost 1.

We use VALUE-ITERATION to compute $V_{cp}^{\hat{M}}(\langle s_0, k\rangle)$, as doing so requires a universal plan (see Section 3.1). Therefore, during this greedy process we also discard any reduction that makes the problem unsolvable (line 12), thereby ensuring that the value $V_{cp}^{M}(\langle s_0, k\rangle)$ remains well-defined. A reduction becomes unsolvable if *any* state of the model is a dead-end (i.e., a state from which no policy can reach the goal) under that reduction, and it is solvable otherwise.

To check if a reduction $\hat{M}$ is solvable, we perform a strongly connected component analysis on a modified version of $\hat{M}$. Specifically, we add an artificial initial state, $\bar{s}_0$, and an action, $\bar{a}$, and modify the transition function so that $T'(s'|\bar{s}_0, \bar{a}) = \frac{1}{|S'|}$ for all $s' \in S'$. In other words, $\bar{s}_0$ leads to all states in the reduced model with equal probability. Furthermore, we modify the transition function so that $T(\bar{s}_0|s_g, a) = 1$ for all states $s_g \in G'$, that is, goals transition back to the artificial initial state.

As it turns out, it is easy to detect dead-ends in $\hat{M}$ by performing a strongly connected component analysis on the all-outcomes determinization of this new problem (e.g., using Tarjan's algorithm (Tarjan, 1972)). If $\hat{M}$ has no dead-ends, then this modified problem will have a single component. Conversely, if there is more than one component in the problem, then there must be a dead-end state. To see why no dead-ends implies a single component, note that having goals connected back to $\bar{s}_0$ implies that any state with a path to the goal is strongly connected to $\bar{s}_0$; the converse is easily proven from the same observation. Note that we added $\bar{s}_0$, rather than connect the goal with the initial state, because computing $V_{cp}^{\hat{M}}(\langle s_0, k\rangle)$ requires a universal plan for the reduction, rather than one that covers only those states reachable from $\langle s_0, k\rangle$.

Obviously, this greedy approach could be costly in terms of computation time, since every evaluation of the objective function involves computing a universal plan for the reduced model, and for $k > 0$ this is in fact more costly than solving the original problem using value iteration. In order to overcome this difficulty, the greedy approach is meant to be applied to relatively small problem instances that can be solved quickly, allowing the planner to learn a good reduced model for the domain. The underlying assumption is that if a small problem instance captures the relevant structure of the domain, then a good reduction for this instance generalizes to larger problems.

6.2 Learning a Good Determinization

The second approach is specifically targeted towards learning a good single-outcome determinization, although its main idea can be directly applied in learning $\mathcal{M}_l^k$-reductions. The approach is motivated by the observation that many stochastic domains have inherent structures that make some of their determinizations significantly more effective than others. As illustrated in Section 4.2, one such domain is the TRIANGLE-TIREWORLD problem (Little & Thiébaux, 2007), where the optimal policy can be obtained by planning as if a flat tire will always occur. The interesting part is that this is true for *all* instances of this problem, regardless of size.

TRIANGLE-TIREWORLD is a great example of a domain where all problem instances share a probabilistic structure that can be captured by a single-outcome determinization. In practical terms, this means that it is possible to learn a determinization on the smaller problems, and then use it for solving larger ones. Moreover, one advantage of learning

Algorithm 8: GREEDY-LEARN: A greedy method for finding good reduced models

input: MDP problem $\mathbb{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{C}, s_0, \mathcal{G} \rangle, k, l, \tau$
output: Reduced model M

$\mathcal{E}^M \leftarrow \bigcup_{a \in \mathcal{A}} outcomes(a)$
$M \leftarrow \mathcal{M}_l^k$-reduction of $\mathbb{M}$ with primary outcomes $\mathcal{E}^M$
$V_{opt} \leftarrow V_{cp}^M(\langle s_0, k \rangle)$
while ***true*** **do**
  $V_{best} \leftarrow \infty$
  $\alpha \leftarrow \emptyset$
  $e_{best}^{\alpha} \leftarrow \emptyset$
  for $a \in \mathcal{A}$ **do**
    for $e^a \in (outcomes(a) \cap \mathcal{E}^M)$ **do**
      $\hat{\mathcal{E}} \leftarrow \mathcal{E}^M \setminus \{e\}$
      $\hat{M} \leftarrow$ CREATE-REDUCED-MDP$(\mathbb{M}, s_0, \hat{\mathcal{E}}, k)$
      if $SOLVABLE(\hat{M}) \wedge V_{cp}^{\hat{M}}(\langle s_0, k \rangle) < V_{best}$ **then**
        $V_{best} \leftarrow V_{cp}^{\hat{M}}(\langle s_0, k \rangle)$
        $\alpha \leftarrow a$
        $e_{best}^{\alpha} \leftarrow e^a$
  if $V_{best} = \infty$ **then**
    break // *Removing any outcome makes problem unsolvable*
  $\mathcal{E}^M \leftarrow \mathcal{E}^M \setminus \{e_{best}^{\alpha}\}$
  $l_{max} \leftarrow \max_a |outcomes(a) \cap \mathcal{E}^M|$
  $M \leftarrow$ CREATE-REDUCED-MDP$(\mathbb{M}, s_0, \mathcal{E}^M, k)$
  if $\left(\frac{V_{best} - V_{opt}}{V_{opt}}\right) > \tau \wedge l_{max} \leq l$ **then**
    break

determinizations over more complex $\mathcal{M}_l^k$-reductions is that it is easier to enumerate all the possible determinizations of a domain, and that each of these can be solved much faster (e.g., by using FF-LAO*-REPLAN).

Building on these observations, Algorithm 9 illustrates LEARNING-DET, a brute-force approach to learn a determinization $\mathcal{P}$ for problem $\mathcal{D}$. Given an input $\mathbb{M}_l$, representing the problem used for learning, this procedure does a comprehensive search over the space of all of the domain's determinizations, at the level of parameterized action schemata. For each, we estimate the probability of success (P_i) and the expected execution cost ($\mathbb{C}_i$) of executing a continual planning approach (e.g., $\mathcal{M}_l^k$-REPLAN or FF-LAO*-REPLAN) on $\mathbb{M}_l$; the costs and probabilities are estimated using Monte-Carlo simulations. Finally, we pick the determinization with the lowest expected cost, among the ones with the highest probability of success.

There are some subtleties involved in this process. Note that both of the continual planning approaches described here assume that there is a proper policy for the given problem. This will most likely not be the case for many of the determinizations explored

Algorithm 9: LEARNING-DET

input: $\mathcal{D}, \mathbb{M}_l = \langle S, A, T, C, s_0, G \rangle, k$
output: $\mathcal{P}$
$\{\mathcal{P}_1, ..., \mathcal{P}_\mu\} \leftarrow$ Create all possible determinizations of $\mathcal{D}$
forall $i \in \{1, ..., \mu\}$ **do**
  $M \leftarrow$ CREATE-REDUCED-MDP$(\mathbb{M}_l, s_0, \mathcal{P}_i, k)$
  Estimate probability of successs and expected cost of a continual planning approach with input M, k
$P^* \leftarrow \max_i P_i$
$\mathcal{P} \leftarrow \mathcal{P}_{\min_i \mathbb{C}_i \ s.t. \ P_i = P^*}$

by LEARNING-DET; in fact, under some determinizations the goals might be completely unreachable from any state. To circumvent this, we use the same technique we described in Section 4.2, where a cap is put on the maximum cost that can be assigned to a state. While this introduces a new parameter impacting the planner's decisions, and hides the true impact of dead-end states, note that LEARNING-DET still attempts to maximize the multi-objective evaluation criterion typically used when unavoidable dead-ends exist (Kolobov et al., 2012; Steinmetz, Hoffmann, & Buffet, 2016).

7. Experimental Results

In this section we present experiments for evaluating the performance of $\mathcal{M}_l^k$-REPLAN and $\mathcal{M}_l^k$-ANYTIME (Section 7.1), and FF-LAO*-REPLAN (Section 7.2). We also evaluate the accuracy of our strategies for learning reduced models: the greedy approach described in Section 6.1 (evaluated in Section 7.1), and LEARNING-DET, described in Section 6.2 (evaluated in Section 7.2)[5].

7.1 Evaluating $\mathcal{M}_l^k$-REPLAN

We evaluate the use of our continual planning approach, $\mathcal{M}_l^k$-REPLAN with several reductions of the racetrack domain, and compare their performance with LAO* using the full transition model; we also use LAO* to solve the reduced models.[6] For the racetrack problem, we used $p_{slip} = 0.1$ and $p_{er} = 0.05$ (see description in Section 4.1). We evaluate $\mathcal{M}_l^k$-REPLAN using two possible sets of primary outcomes, one with $l = 1$ and one with $l = 2$, and values of $k \in \{0, 1, 2, 3\}$ for each of them; in our discussion we use the notation $\mathcal{M}_l^k$ to refer to the planner using the $\mathcal{M}_l^k$-reduction. In all cases, we used the optimal solution of the all-outcomes determinization as the initial heuristic, which is admissible for every possible reduction of the domain.

5. Source code for reproducing these experiments is available at *https://github.com/luisenp/mdp-lib*.
6. While FF-LAO* is in principle applicable to the racetrack domain, our implementation of the domain is not represented in PPDDL, thus we do not include results for FF-LAO* in this section. Note that representing the racetrack domain in PPDDL is somewhat cumbersome, particularly due to the need for collision detection with walls. Since we later experiment with much larger PPDDL domains, we conjecture that not much insight would be gained from such an effort.

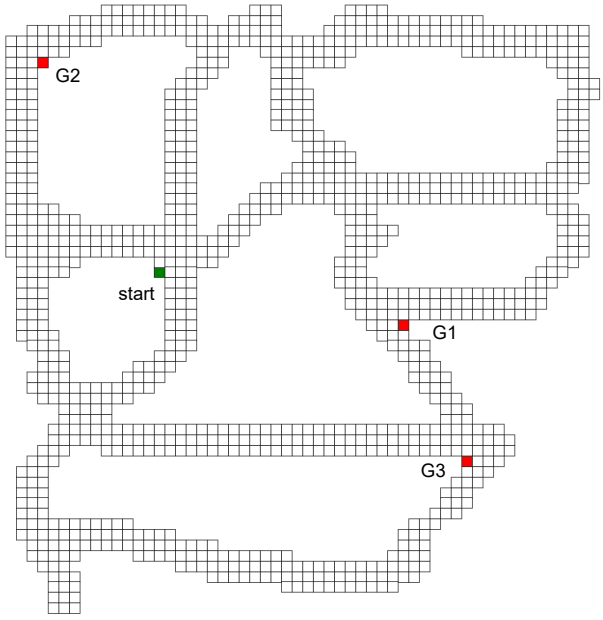

Figure 6: An instance of the racetrack domain.

We learned the two sets of primary outcomes using GREEDY-LEARN (Algorithm 8), on a smaller track with 1,367 states, using $k = 0$ and $\tau = 1.05$. In the case of $l = 2$ (i.e., at most two primary outcomes), GREEDY-LEARN found that using determinization was within the desired tolerance (τ); therefore we used the last $\mathcal{M}_2^k$-reduction found such that at least one action had two primary outcomes. In particular, the $\mathcal{M}_1^k$-reduction used was the most-likely-outcome determinization (outcomes o0, o6, and o11 in Figure 2), and the $\mathcal{M}_2^k$-reduction added to that the possibility of slipping when accelerating in one direction (outcomes o0, o6, o11, and o7 in Figure 2). The racetrack used in our experiments is shown in Figure 6, which has 34,897 states. We evaluated the planner over the three different goal configurations shown in the figure.

Table 3 (bottom) shows the total CPU time spent on planning, which includes the time used to compute an initial plan, as well as the time needed for re-planning. The time reported is the average taken over 500 simulations (the observed standard error was negligible, so it's not reported). In the large majority of cases, the planning time is significantly shorter than the time necessary to plan with the full model; in fact, for values of $k = 0$ and $k = 1$ this time is shorter by multiple orders of magnitude. The only case considered in which planning with the reduced model is slower corresponds to problem G3, where using $k = 3$ was slower than using the full model.

The expected costs of the resulting policies are shown in Table 3 (top), which are computed exactly using the Markov Chain discussed in Section 3.1. Note that planning with the most-likely-outcome determinization ($\mathcal{M}_1^0$), while being extremely fast, always results in more than 19% increase in cost with respect to the optimal cost (and in one case 34.6%).

	Expected Cost								
	LAO*	$\mathcal{M}_1^0$	$\mathcal{M}_1^1$	$\mathcal{M}_1^2$	$\mathcal{M}_1^3$	$\mathcal{M}_2^0$	$\mathcal{M}_2^1$	$\mathcal{M}_2^2$	$\mathcal{M}_2^3$
G1	16.03	19.16	16.42	16.19	16.10	18.35	16.35	16.26	16.08
G2	14.96	20.14	15.51	15.12	15.03	19.28	15.40	15.15	15.01
G3	20.00	23.83	20.69	20.30	20.15	24.09	20.51	20.31	20.21
	CPU Time								
	LAO*	$\mathcal{M}_1^0$	$\mathcal{M}_1^1$	$\mathcal{M}_1^2$	$\mathcal{M}_1^3$	$\mathcal{M}_2^0$	$\mathcal{M}_2^1$	$\mathcal{M}_2^2$	$\mathcal{M}_2^3$
G1	7,610	1	26	495	3,640	5	120	1,978	7,285
G2	8,244	1	136	1,309	4,871	2	151	1,138	4,400
G3	6,813	1	126	1,723	8,093	8	489	5,306	18,501

Table 3: Expected cost and average planning time obtained with several reduced models of the racetrack domain.

Adding a single outcome, without increasing k ($\mathcal{M}_2^0$), decreases the expected cost by at least 5% in two of the problems considered, with marginal increase in total planning time. Notice, however, that $\mathcal{M}_2^0$ resulted in a slight increase in expected cost for problem G3, which indicates that simply adding an outcome is not guaranteed to result in a better plan. On the other hand, increasing k generally results in better policies in these experiments, a trend that is observed in all the problems and with all values of k. With $k = 1$, the policies are always within 4% of optimal. With $k = 3$, the difference with respect to the optimal cost reduces to less than 1%, even when the underlying model is deterministic ($\mathcal{M}_1^3$). Interestingly, for G3, the use of $k = 3$ results in longer planning times compared with optimal planning.

The results discussed above offer evidence that $\mathcal{M}_l^k$-reductions can be used to compute near-optimal plans, and do so orders of magnitude faster than optimal planning under the full model. However, note that in these experiments the planners were allowed as much time as needed for planning, which is not necessarily the most practical approach. Indeed, a standard setting encountered in the planning literature is to study the performance of a planner when there is only a limited window of time available for planning before each action.

To this end, we also evaluated $\mathcal{M}_l^k$-ANYTIME (Algorithm 2) using the same set of racetrack problems illustrated in Figure 6, assuming a fixed time per action ranging from 0.05 seconds to 6.4 seconds (increased in multiples of 2). We used the LRTDP algorithm (Bonet & Geffner, 2003) as the underlying optimal planner, since it has better anytime properties than LAO*. The results for the best performing reductions are shown in Figure 7, along with the results obtained using the full model. The plots show the expected costs obtained with all the reductions, averaged over 10,000 simulations, along with error bars representing 95% confidence intervals.

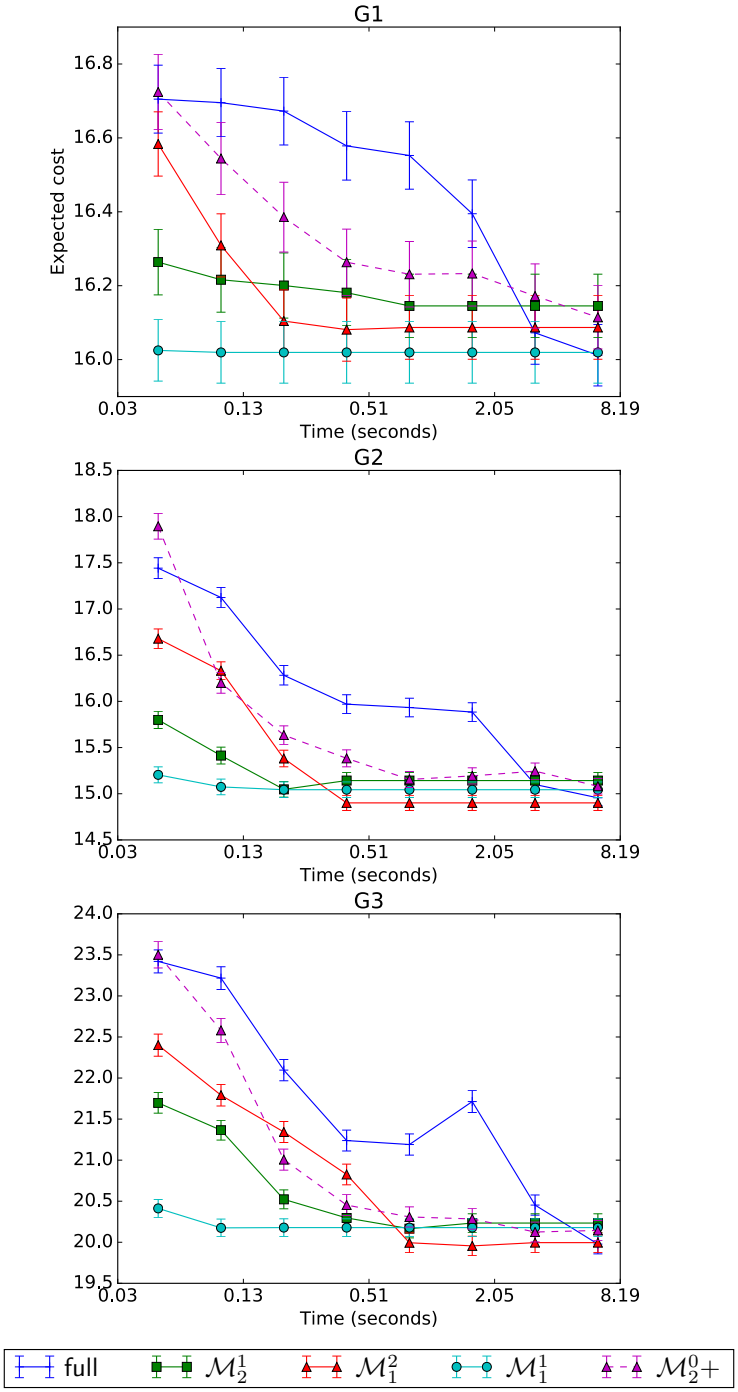

Figure 7: Anytime performance of several $\mathcal{M}_l^k$-reductions using $\mathcal{M}_l^k$-ANYTIME to solve three instances of the racetrack problem. From left to right, results corresponding to G1, G2, and G3 in Figure 6.

As seen in Figure 7, in the large majority of scenarios the best anytime performance was obtained using $k = 1$, both with one and two primary outcomes (planners $\mathcal{M}_1^1$ and $\mathcal{M}_2^1$, respectively). Planner $\mathcal{M}_1^1$, in particular, significantly outperformed all other planners for all times lower or equal to 200 milliseconds per action. Planners $\mathcal{M}_2^1$ and $\mathcal{M}_1^2$ were able to perform better when the time increased, and for times of 800 milliseconds per action, or higher, the performance of $\mathcal{M}_1^1$, $\mathcal{M}_2^1$, and $\mathcal{M}_1^2$ was comparable ($\mathcal{M}_1^2$ was slightly better in problems G2 and G3, but not at the 95% significance level). On the other hand, planning with the full model required much larger times per action in order to result in comparable performance (at least 3.2 seconds per action), and its performance was much worse than the other models for lower times. Note that Figure 7 does not include the results for $\mathcal{M}_1^0$, and $\mathcal{M}_2^0$. We did not include these results to maintain clarity in the figure, but note that the observed performance matched that shown in Table 3 (a straight horizontal line), which was worse than all other planners considered. Unlike the more complex models, using $k = 0$ meant that the planner could not take advantage of having more time per action, when available.

Interestingly, to address some of the downsides of choosing a low value of k, we experimented with a variant of $\mathcal{M}_l^k$-ANYTIME that leverages labels produced by an algorithm like LRTDP. In particular, when a plan is about to be computed (line 7 of Algorithm 2), instead of using $\langle s, k\rangle$, the new algorithm checks for the lowest value of $j \geq k$ such that $\langle s, j\rangle$ has not been previously labeled as solved, and uses $\langle s, j\rangle$ as the initial state for planning. When picking an action (line 4 of Algorithm 2), the algorithm uses the best action associated with $\langle s, j'\rangle$, where $j' \geq k$ is the highest value such that $\langle s, j'\rangle$ has been labeled as solved; if no such value exists, then it uses a greedy action on $\langle s, k\rangle$. This means that the planner can continuously increase k throughout execution in order to compute more robust plans. As it turns out, this strategy is effective, and the results are illustrated in the plot labeled as $\mathcal{M}_2^0+$. While the results are not better than those of the other reduced models considered, the approach has good anytime performance, and may be a good choice when it is not clear what a good value for k is.

7.2 Evaluating FF-LAO* on IPPC'08 Benchmarks

In this section we evaluate FF-LAO* on a set of challenging PPDDL problems. Note that, while $\mathcal{M}_l^k$-REPLAN is in principle applicable to the domains considered in this section, the scale of the problem instances is such that even their determinizations are difficult to solve optimally in a short amount of time. On the other hand, FF-LAO* circumvents this complexity by leveraging a classical planner, thus we focus the following discussion in on the performance of this algorithm.

7.2.1 Domains and Methodology

We evaluated FF-LAO* and LEARNING-DET on a set of problems taken from IPPC'08 (Bryce & Buffet, 2008). Specifically, we used the first 10 problem instances of the following four domains: TRIANGLE-TIREWORLD, BLOCKSWORLD, EX-BLOCKSWORLD, and ZENOTRAVEL. Unfortunately, the rest of the IPPC'08 domains are not supported by our PPDDL parser (Bonet & Geffner, 2005). Additionally, we modified the EX-BLOCKSWORLD domain to avoid the possibility of blocks to be put on top of themselves (Trevizan & Veloso, 2014).

Domain	Chosen Determinization
TRIANGLE-TIREWORLD	The car always gets a flat tire ($p = 0.5$)
EX-BLOCKSWORLD	Blocks detonate when trying to put them on the table ($p = 0.4$), but they do not detonate when trying to put them on another block ($p = 0.9$)
BLOCKSWORLD	Actions PICK-UP and PUT-TOWER-ON-BLOCK fail ($p = 0.25$ and $p = 0.9$, respectively), and all other three actions succeed ($p = 0.75$, $p = 0.75$, $p = 0.1$)
ZENOTRAVEL	All five complete actions succeed ($p = 0.5$, $p = 0.25$, $p = 1/25$, $p = 1/15$, $p = 1/7$)

Table 4: Determinizations learned by LEARNING-DET on IPPC'08 benchmarks.

The evaluation methodology was similar to the one used in past planning competitions: we give each planner 20 minutes to solve 50 rounds of each problem (i.e., reach a goal state starting from the initial state). Then we measure its performance in terms of the number of rounds that the planner was able to solve during that time. All experiments were conducted on an Intel Core i7-6820HQ machine running at 2.70GHz with a 4GB memory cutoff.

We evaluated the planners using the MDPSIM (Younes et al., 2005) client/server program for simulating SSPs, by having planners repeatedly perform the following three steps: 1) connect to the MDPSIM server to receive a state, 2) compute an action for the received state and send the action to the MDPSIM server, and 3) wait for the server to simulate the result of applying this action and send a new state. A simulation ends when a goal state is reached, when an invalid action is sent by the client, or after 2500 actions have been sent by the planner.

We compared the performance of FF-LAO* with our own implementations of FF-REPLAN and RFF, as well as the original author's implementation of SSIPP (Trevizan & Veloso, 2014). We evaluated two variants of FF-REPLAN, one using the most likely outcome determinization, MLO, (FF_S) and another one using the all-outcomes determinization, AO, (FF_A). For RFF we used MLO and the *Random Goals* variant, in which before every call to FF, a random subset (size 100) of the previously solved states are added as goal states. Additionally, we used a probability threshold $\rho = 0.2$. The choice of these parameters was informed by analysis in the original work (Teichteil-Königsbuch et al., 2010). For SSIPP we used $t = 3$ and the h_{add} heuristic, parameters also informed by the original work (Trevizan & Veloso, 2014).

For FF-LAO*, we learned a good determinization to use by applying LEARNING-DET on the first problem of each domain (p01), with $k = 0$. This choice of k was motivated both by time considerations, and by the rationale that $k = 0$ should better reflect the impact of each determinization (since FF-LAO* becomes a fully determinization-based planner). We used a dead-end cap $\mathcal{D} = 500$ throughout our experiments. We initialized values with the non-admissible FF heuristic (Bonet & Geffner, 2005). The learned determinizations, and their probabilities under the original model, are listed in Table 4. Note that these determinizations do not correspond to a single one-size-fit-alls rule, such as most-likely- or least-likely-outcomes.

A diverse range of behaviors can be observed in these determinizations. In TRIANGLE-TIREWORLD, the best determinization induces a conservative policy, while in ZENOTRAVEL an optimistic model is a better choice. This is not surprising, since negative outcomes in ZENOTRAVEL are not catastrophic, so ignoring them does not have the large impact it would have in TRIANGLE-TIREWORLD; we emphasize that the choice of reduction was determined automatically by LEARNING-DET, without using any prior domain knowledge. For EX-BLOCKSWORLD and BLOCKSWORLD the chosen determinizations present a mix between low and high probability outcomes, some desirable and some not. In EX-BLOCKSWORLD the chosen determinization pushes the planner towards favoring blocks on the table (as support of other blocks), which is a less risky behavior.

We ran LEARNING-DET offline, prior to the MDPSIM evaluation. Note, however, that the time taken by the brute force search plus the time used to solve problem p01 with the chosen determinization was, in all cases, well below the 20 minute limit (approx. 2 minutes in the worst case). The remaining parameter for FF-LAO* is the value of k. We report the best performing configuration in the range $k \in [0, 3]$, which was $k = 0$ for most domains, with the exception of EX-BLOCKSWORLD, which required $k = 3$. Note that FF-LAO* with $k = 0$ is essentially equivalent to FF-REPLAN, so any advantage obtained over FF_S and FF_A is completely derived from the choice of determinization.

7.2.2 Results and Discussion

Figure 8 shows the number of successful rounds obtained by each planner in the benchmarks. In general, FF-LAO* either tied for the best, or outperformed the baselines. All planners had a 100% success rate in BLOCKSWORLD, so there is not much room for comparison.

In the TRIANGLE-TIREWORLD domain, FF-LAO* and FF_S had 100% success rate, while RFF ran out of time in the last 3 problems. On the other hand, the performance of SSIPP and FF_A deteriorated quickly as the problem instance increased. It is worth pointing out that the performances of FF_S and RFF in this domain are quite sensitive to tie-breaking—there are only two outcomes to choose from, each occurring with 0.5 probability. As the results of FF_A suggest, a different choice would have resulted in a much worse success rate. On the other hand, the use of LEARNING-DET gets around this issue by automatically choosing the best determinization to use, a process that took seconds. While we do note that the *best goals* parameterization of RFF gets around this issue, its computational cost is much harder, so it is not obvious that it would actually improve performance in this case (Teichteil-Königsbuch et al., 2010).

In the EX-BLOCKSWORLD domain, FF-LAO* (with $k = 3$) and SSIPP significantly outperform the other two planners, solving 252 and 250 rounds, respectively, against 187 for both FF_S and RFF, and 200 for FF_A. Interestingly, in this domain the determinization found by LEARNING-DET is not sufficient to obtain good performance; in fact, only 3 problems had a non-zero success rate with $k = 0$. This highlights the utility of doing probabilistic reasoning with FF-LAO*. Although not shown in the figure for the sake of clarity, the performance of FF-LAO* with $k = 1$ (214 successful rounds) was already better than most of the baselines, except for SSIPP.

In ZENOTRAVEL, FF-LAO* and FF_A were remarkably better than the other two planners: they achieved 100% success rate in all domain instances, while the other baselines failed

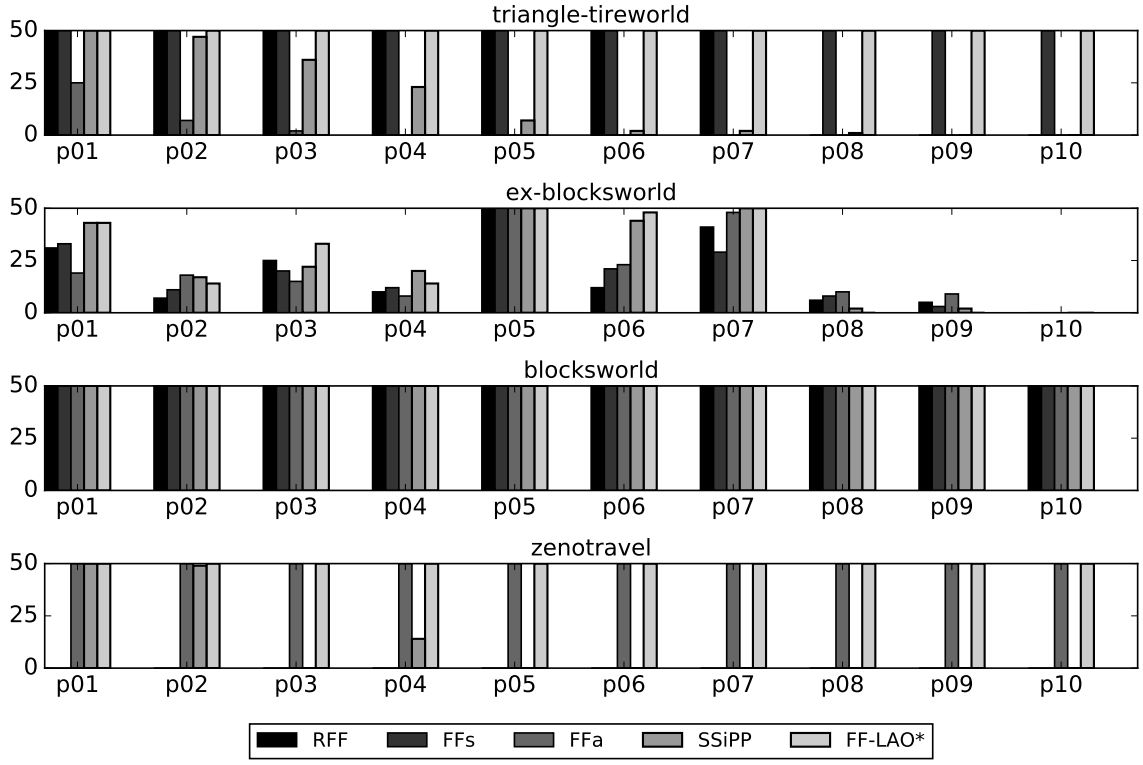

Figure 8: Number of solved rounds by 5 different planners in IPPC'08 benchmarks.

almost all of the rounds. In the case of the determinization-based planners, this is due to the goal becoming unreachable under MLO, so the choice of determinization has a significant impact on performance.

Finally, we also tried to perform a comparison with FF-H$^+$, a state-of-the-art planner based on hindsight optimization that has been shown to outperform RFF on the IPPC benchmarks (Yoon et al., 2010). Unfortunately, we were not able to obtain or replicate the code and thus we report results for FF-H$^+$ from the original paper. While this means that our results are not directly comparable, they still provide some useful insights. In particular, the results obtained by FF-LAO* appear to be comparable to those reported for FF-H$^+$ on the same domains. Considering a maximum of 30 rounds per problem, as reported for FF-H$^+$ in (Yoon et al., 2010), FF-LAO* was able to solve 1,078 rounds successfully, under a time limit of 20 minutes per problem instance (30 rounds each). Conversely, FF-H$^+$ is reported to have solved 1,084 instances—at most—using a 30 minute limit per instance; note that the reported planning times for FF-H$^+$ are higher than 20 minutes in all cases, except for TRIANGLE-TIREWORLD. In general, the results suggest that FF-LAO* can obtain comparable success rate, with potentially less overall planning time. With the caveat mentioned earlier, this comparison suggests that it is beneficial to plan with determinizations that are automatically tailored to the specific characteristics of a domain.[7]

7. The results reported in (Yoon et al., 2010) are not broken down by problem instance. Since they experimented on 15 problem instances for each domain, rather than 10, as we did, we have computed

8. Conclusions

We introduce a spectrum of MDP reduced models that can help reduce planning time and improve the overall performance of stochastic planning algorithms, when the cost of planning is taken into consideration. Each reduced model is characterized by two parameters: the maximum number of primary outcomes per action and the maximum number of exceptional outcomes in the plan. We show that reduced models can accelerate planning by orders of magnitude. We also introduce a continual planning approach that generates new plans in parallel to plan execution when the number of exceptions reaches the maximum allowed. A benefit of this approach is that it is amenable to an exact analytical evaluation of the benefits of planning with reduced models.

Some commonly used determinization approaches are instances of this spectrum of reductions, with one primary outcome per action and zero exceptional outcomes. We show that reduced models with either more than one primary outcome per action or with some exceptional outcomes (or both) can be beneficial, producing significantly higher comprehensive value relative to the best possible determinization.

We also investigate how to generate a good reduced model, be it a determinization or not, showing that the choice of primary outcomes is non-trivial, even when reductions are limited to determinization. To address the challenge of choosing the set of primary outcomes, our work introduces and evaluates a greedy algorithm that can produce good reduced models automatically, with the results indicating that performance can be improved relative to a baseline determinization technique, or to directly solving the original model. These results extend previous work demonstrating the value of model simplification in reinforcement learning (Ravindran & Barto, 2002) and planning under uncertainty (Dean et al., 1997; Dean & Givan, 1997; Givan et al., 2003). In particular, they place work on determinization in a broader context and lay the foundation for further work to increase the benefits offered by planning with reduced models.

There are a number of aspects of our approach that can be further improved. We are exploring better ways to redistribute the probabilities of exceptional outcomes among the primary outcomes based on various measures of *structural similarity* of the outcomes (e.g., similarity in outcome states or their values). Additionally, there is potential in extending our continual planning approach to handle a more diverse set of strategies, such as replanning as soon as the first exception occurs, rather than waiting for k exceptions. In fact, planning could even start before any exception happens, simply by projecting the most likely exceptions to happen and planning ahead just in case, as in the *continual computation* framework (Horvitz, 2001). Finally, we are developing more efficient ways to explore the space of $\mathcal{M}_l^k$-reductions and find good ones, for example, by using sampling to estimate the regret of labeling an outcome as primary. These promising research avenues will further increase the impact of planning with reduced models.

the maximum possible number of successful rounds obtained in the first 10 problems as $rounds = \min\{300, rounds\}$.

Acknowledgments

We thank Kyle Wray and Sandhya Saisubramanian for their helpful feedback and suggestions on earlier versions of this work. This work was supported by the National Science Foundation Grants No. IIS-1405550 and IIS-1524797.

References

Barto, A. G., Bradtke, S. J., & Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, *72*, 81–138.

Bellman, R. E. (1957). *Dynamic Programming*. Princeton University Press.

Bonet, B., & Geffner, H. (2003). Labeled RTDP: Improving the convergence of real-time dynamic programming. In *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling*, pp. 12–21, Trento, Italy.

Bonet, B., & Geffner, H. (2005). mGPT: A probabilistic planner based on heuristic search. *Journal of Artificial Intelligence Research*, *24*, 933–944.

Boutilier, C., & Poole, D. (1996). Computing optimal policies for partially observable decision processes using compact representations. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pp. 1168–1175, Portland, Oregon.

Brenner, M., & Nebel, B. (2009). Continual planning and acting in dynamic multiagent environments. *Autonomous Agents and Multi-Agent Systems*, *19*(3), 297–331.

Bryce, D., & Buffet, O. (2008). Sixth international planning competition: Uncertainty part. In *Proceedings of the Sixth International Planning Competition (IPC'08)*.

Chanel, C. P. C., Lesire, C., & Teichteil-Königsbuch, F. (2014). A robotic execution framework for online probabilistic (re)planning. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling*, pp. 454–462, Portsmouth, New Hampshire.

Cheung, T. L., Okamoto, K., Maker, F., Liu, X., & Akella, V. (2009). Markov decision process (MDP) framework for optimizing software on mobile phones. In *Proceedings of the Seventh ACM International Conference on Embedded Software*, pp. 11–20.

Chien, S., Knight, R., Stechert, A., Sherwood, R., & Rabideau, G. (2000). Using iterative repair to improve responsiveness of planning and scheduling. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling*, pp. 300–307, Breckenridge, Colorado.

Dean, T., & Boddy, M. (1988). An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pp. 49–54, Saint Paul, Minnesota.

Dean, T., & Givan, R. (1997). Model minimization in Markov decision processes. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pp. 106–111, Providence, Rhode Island.

Dean, T., Givan, R., & Leach, S. (1997). Model reduction techniques for computing approximately optimal solutions for Markov decision processes. In *Proceedings of the*

Thirteenth Conference on Uncertainty in Artificial Intelligence, pp. 124–131, Providence, Rhode Island.

Dean, T., Kaelbling, L. P., Kirman, J., & Nicholson, A. (1995). Planning under time constraints in stochastic domains. *Artificial Intelligence*, *76*, 35–74.

desJardins, M. E., Durfee, E. H., Ortiz, C. L., & Wolverton, M. J. (1999). A survey of research in distributed, continual planning. *AI Magazine*, *20*(4), 13–22.

Garland, A., & Lesh, N. (2002). Plan evaluation with incomplete action descriptions. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pp. 461–467, Edmonton, Alberta.

Ginsberg, M. L. (1989). Universal planning: An (almost) universally bad idea. *AI Magazine*, *10*(4), 40–44.

Givan, R., Dean, T., & Greig, M. (2003). Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence*, *147*(1–2), 163–223.

Hansen, E. A., & Zilberstein, S. (1998). Heuristic search in cyclic AND/OR graphs. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pp. 412–418, Madison, Wisconsin.

Hansen, E. A., & Zilberstein, S. (2001). LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, *129*(1-2), 35–62.

Hoffmann, J., & Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, *14*(1), 253–302.

Horvitz, E. (1987). Reasoning about beliefs and actions under computational resource constraints. In *Proceedings of the Third Conference on Uncertainty in Artificial Intelligence*, pp. 429–444, Seattle, Washington.

Horvitz, E. (2001). Principles and applications of continual computation. *Artificial Intelligence*, *126*(1-2), 159–196.

Hostetler, J., Fern, A., & Dietterich, T. (2014). State aggregation in Monte Carlo tree search. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, pp. 2285–2292, Quebec City, Canada.

Howard, R. A. (1960). *Dynamic Programming and Markov Processes*. MIT Press.

Hsu, D., Latombe, J.-C., & Kurniawati, H. (2006). On the probabilistic foundations of probabilistic roadmap planning. *International Journal of Robotics Research*, *25*(7), 627–643.

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, *101*, 99–134.

Keller, T., & Eyerich, P. (2011). A polynomial all outcome determinization for probabilistic planning. In *Proceedings of the Twenty-First International Conference on International Conference on Automated Planning and Scheduling*, pp. 331–334. AAAI Press.

Keyder, E., & Geffner, H. (2008). The HMDPP planner for planning with probabilities. In Bryce, D., & Buffet, O. (Eds.), *ICAPS Third International Probabilistic Planning Competition*. IPPC'08.

Koenig, S., Goodwin, R., & Simmons, R. G. (1996). Robot navigation with Markov models: A framework for path planning and learning with limited computational resources. In Dorst, L., Lambalgen, M., & Voorbraak, F. (Eds.), *Reasoning with Uncertainty in Robotics*, Vol. 1093 of *Lecture Notes in Computer Science*, pp. 322–337. Springer Berlin Heidelberg.

Kolobov, A., Mausam, & Weld, D. S. (2010). Classical planning in MDP heuristics: With a little help from generalization. In *Proceedings of the Twelfth International Conference on Automated Planning and Scheduling*, pp. 97–104, Toronto, Canada.

Kolobov, A., Mausam, & Weld, D. S. (2012). A theory of goal-oriented MDPs with dead ends. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pp. 438–447, Catalina Island, California.

Krause, A., & Golovin, D. (2014). Submodular function maximization. In *Tractability: Practical Approaches to Hard Problems*, pp. 71–104. Cambridge University Press.

Little, I., & Thiébaux, S. (2007). Probabilistic planning vs. replanning. In *Proceedings of the ICAPS'07 Workshop on the International Planning Competition: Past, Present and Future*.

Littman, M. L. (1997). Probabilistic propositional planning: Representations and complexity. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pp. 748–754, Providence, Rhode Island.

Myers, K. L. (1999). CPEF: A continuous planning and execution framework. *AI Magazine*, *20*(4), 63–69.

Nemhauser, G. L., Wolsey, L. A., & Fisher, M. L. (1978). An analysis of approximations for maximizing submodular set functions–I. *Mathematical Programming*, *14*(1), 265–294.

Nguyen, T. A., & Kambhampati, S. (2014). A heuristic approach to planning with incomplete STRIPS action models. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling*, pp. 190–198, Portsmouth, New Hampshire.

Nguyen, T. A., Kambhampati, S., & Do, M. B. (2013). Synthesizing robust plans under incomplete domain models. In *Proceedings of the Neural Information Processing Systems*, pp. 2472–2480, Lake Tahoe, Nevada.

Petrik, M., & Subramanian, D. (2014). RAAM: The benefits of robustness in approximating aggregated MDPs in reinforcement learning. In *Proceedings of Neural Information Processing Systems*, pp. 1979–1987, Montreal, Canada.

Petrik, M., & Zilberstein, S. (2011). Linear dynamic programs for resource management. In *Proceedings of the Twenty-Fifth Conference on Artificial Intelligence*, pp. 1377–1383, San Francisco, California.

Pineda, L., Takahashi, T., Jung, H.-T., Zilberstein, S., & Grupen, R. (2015). Continual planning for search and rescue robots. In *Proceedings of the IEEE-RAS Fifteenth International Conference on Humanoid Robots*, pp. 243–248, Seoul, Korea.

Pineda, L., & Zilberstein, S. (2014). Planning under uncertainty using reduced models: Revisiting determinization. In *Proceedings of the Twenty-Fourth International Con-*

ference on Automated Planning and Scheduling, pp. 217–225, Portsmouth, New Hampshire.

Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA.

Ravindran, B., & Barto, A. G. (2002). Model minimization in hierarchical reinforcement learning. In *Proceedings of the 5th International Symposium on Abstraction, Reformulation and Approximation*, pp. 196–211, Kananaskis, Alberta.

Russell, S. J., & Wefald, E. (1991). Principles of metareasoning. *Artificial Intelligence*, *49*(1-3), 361–395.

Schoppers, M. J. (1987). Universal plans for reactive robots in unpredictable environments. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pp. 1039–1046, Milan, Italy.

Singh, J. P., Alpcan, T., Agrawal, P., & Sharma, V. (2010). A Markov decision process based flow assignment framework for heterogeneous network access. *Wireless Networks*, *16*(2), 481–495.

Steinmetz, M., Hoffmann, J., & Buffet, O. (2016). Revisiting goal probability analysis in probabilistic planning. In *Proceedings of the 26th International Conference on Automated Planning and Scheduling*.

Sutton, R. S., & Barto, A. G. (1998). *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA.

Svegliato, J., Wray, K. H., & Zilberstein, S. (2018). Meta-level control of anytime algorithms with online performance prediction. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, pp. 1499–1505, Stockholm, Sweden.

Tarjan, R. (1972). Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, *1*(2), 146–160.

Teichteil-Königsbuch, F., Kuter, U., & Infantes, G. (2010). Incremental plan aggregation for generating policies in MDPs. In *Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems*, pp. 1231–1238, Toronto, Canada.

Thrun, S., Burgard, W., & Fox, D. (2005). *Probabilistic Robotics*. MIT Press, Cambridge, MA, USA.

Trevizan, F. W., & Veloso, M. M. (2014). Depth-based short-sighted stochastic shortest path problems. *Artificial Intelligence*, *216*, 179–205.

Weber, C., & Bryce, D. (2011). Planning and acting in incomplete domains. In *Proceedings of the Twenty-First International Conference on Automated Planning and Scheduling*, pp. 274–281, Freiburg, Germany.

Yoon, S. W., Fern, A., & Givan, R. (2007). FF-Replan: A baseline for probabilistic planning. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling*, pp. 352–359, Providence, Rhode Island.

Yoon, S., Fern, A., Givan, R., & Kambhampati, S. (2008). Probabilistic planning via determinization in hindsight. In *Proceedings of the Twenty-Third National Conference on Artificial Intelligence*, pp. 1010–1016, Chicago, Illinois.

Yoon, S., Ruml, W., Benton, J., & Do, M. B. (2010). Improving determinization in hindsight for online probabilistic planning. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling*, pp. 209–216, Toronto, Canada.

Younes, H. L. S., Littman, M. L., Weissman, D., & Asmuth, J. (2005). The first probabilistic track of the international planning competition. *Journal of Artificial Intelligence Research*, *24*(1), 851–887.

Zilberstein, S. (1996). Using anytime algorithms in intelligent systems. *AI Magazine*, *17*(3), 73–83.

Zilberstein, S. (2011). Metareasoning and bounded rationality. In Cox, M., & Raja, A. (Eds.), *Metareasoning: Thinking about Thinking*, pp. 27–40. MIT Press.