March 2019

Efficient Probabilistic Reasoning Using Partial State-Space Exploration

Luis Pineda

EFFICIENT PROBABILISTIC REASONING USING PARTIAL STATE-SPACE EXPLORATION

A Dissertation Presented

by

LUIS PINEDA

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

February 2019

College of Information and Computer Sciences

EFFICIENT PROBABILISTIC REASONING USING PARTIAL STATE-SPACE EXPLORATION

A Dissertation Presented

by

LUIS PINEDA

Approved as to style and content by:

ACKNOWLEDGMENTS

It is difficult to properly express my appreciation for the many people who helped me achieve this goal. Be that as it may, I will try in the next few lines, hoping that the result does them at least a modicum of justice.

From my advisor, Shlomo Zilberstein, I have learned countless lessons on how to conduct world-class research under a spirit of collaboration and cordiality. He has also instilled in me a drive to always dig deeper, and to not get discouraged by the inevitable roadblocks that are found when solving a difficult problem.

I am also thankful to the other members of my committee. Daniel Sheldon's insightful suggestions and sharp intellect have been a source of inspiration. Likewise, I have enjoyed many enlightening and entertaining conversations with Akshay Krishnamurthy, not only covering my thesis work, but also mathematics, statistics and general research advice. Weibo Gong's insights (and his extraordinary optimal control class!) helped me put my work under a wider perspective. I would also like to thank my former (pre-UMass) advisor and mentor, Nestor Queipo, for providing additional proof-reading and advice.

I will fondly remember my times at the Resource-Bounded Reasoning lab. My regular conversations (and debates) with Kyle and Sandhya have made me a better researcher, and made my time at RBR much enjoyable. I am also thankful for my other great RBR lab mates, Rick, Xiaojian, Justin, Connor, Shuwa, and John. Thanks for the good times!

I gained many friends during my time at CICS, whose presence brightened these years of hard-work and long winters, and whom I'll miss a lot. These include Aaron, Addison, Dirk, Emma S., Francisco, Garret, Ian, James, Jesse, Katerina, Keen, Kevin, Matteo, Mike, Myungha, Pat, Pinar, Roy and Samer. I hope we get to hang out in the future, so let's publish and travel a lot!

I have been lucky to be part of a great CS department, with a unique mix of top quality research and a friendly atmosphere. I took many fun courses taught by brilliant faculty, which include Barna Saha, Erik Learned-Miller, Philip Thomas, Ramesh Sitaraman, and Roderic Grupen. I also owe much to the advice and administrative support of Leeanne Leclerc and Michele Roberts, who made everything run impressively smoothly.

My parents have been a constant pillar of support and advice, and, by their example, were the ones that first shaped my intellectual curiosity and constant drive for learning and improvement. My brother and sister have also been a much welcome source of happiness during my travels back home. I am also grateful for the visits of my mother- and sister-in-law, which have been an amazing source of support for me and my family.

This thesis is dedicated to my beloved wife Niri and our beautiful daughter Sarah. Doing a PhD is a lot of work for everyone involved, and I'll be eternally grateful for Niri's patience, empathy and wisdom. Sharing my life with her helps me become a better person every day. Sarah, watching you grow, play, and learn gives me more joy than I ever thought was possible.

ABSTRACT

EFFICIENT PROBABILISTIC REASONING USING PARTIAL STATE-SPACE EXPLORATION

FEBRUARY 2019

LUIS PINEDA

B.Sc., UNIVERSIDAD DEL ZULIA, VENEZUELA

M.Sc., UNIVERSIDAD DEL ZULIA, VENEZUELA

M.Sc., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Shlomo Zilberstein

Planning, namely the ability of an autonomous agent to make decisions leading towards a certain goal, is one of the fundamental components of intelligent behavior. In the face of uncertainty, this problem is typically modeled as a Markov Decision Process (MDP). The MDP framework is highly expressive, and has been used in a variety of applications, such as mobile robots, flow assignment in heterogeneous networks, optimizing software in mobile phones, and aircraft collision avoidance. However, its wide adoption in real-world scenarios is still impaired by the complexity of solving large MDPs. Developing effective ways to tackle this complexity barrier is a challenging research problem.

This thesis focuses on the development of scalable and robust MDP solution approaches for partially exploring the state space of an MDP. The main contribution is a series of mathematical and algorithmic techniques for selecting the parts of the state space that are the most critical for effective planning, with the ultimate goal of maximizing performance in the presence of bounded resources. The proposed approaches work on two distinct axes: i) constructing reduced MDP models that are computationally easier to solve, but whose policies still result in near-optimal performance when applied to the original model, and ii) using sampling-based exploration that is biased towards states for which additional computation can be more productive, in a well-defined sense.

The first part of the thesis addresses the model reduction component, introducing an MDP reduction framework that generalizes popular solution approaches based on determinization. In particular, the framework encompasses a spectrum of MDP reductions differing along two dimensions: i) the number of outcomes per state-action pair that are fully accounted for, and ii) the number of occurrences of the remaining, exceptional, outcomes that are planned for in advance. An important insight resulting from this work is that the choice of reduction is crucial for achieving good performance, an issue under-explored by the planning community, even for determinization-based planners.

The second part of the thesis presents a sampling-based approach that does not require modification of the MDP model. The key idea is to avoid computation in states whose estimated optimal values are more likely to be correct, and rather direct it towards states whose values (which are closely related to policy quality) can be improved the most. The proposed approach represents a novel algorithmic framework that generalizes MDP algorithms based on labeling, a widely used technique in state-of-the-art planners. The framework can be leveraged to create a variety of MDP solvers with different trade-offs between computational complexity and policy quality, and its application to a variety of standard MDP benchmarks results in state-of-the-art performance.

TABLE OF CONTENTS

Page

ACKNOWLEDGMENTS . . . . **iv**

ABSTRACT . . . . **vi**

LIST OF TABLES . . . . **xi**

LIST OF FIGURES . . . . **xii**

CHAPTER

1. INTRODUCTION **1**

1.1 Overview 1
1.2 Brief Overview of Solution Methods for MDPs 2
1.3 Summary of Contributions 7
1.4 Relevant Publications 8
1.5 Thesis organization 9

2. BACKGROUND **11**

2.1 Markov Decision Process 11
2.2 Methods for Solving Markov Decision Processes 14

2.2.1 Fundamental Solution Methods for MDPs 14
2.2.2 Overview of Approximate Planning Methods for Solving MDPs 16
2.2.3 Determinization-based Planners 17

2.2.3.1 FF-Replan 19
2.2.3.2 FF-Hindsight 20
2.2.3.3 RFF 21
2.2.3.4 HMDPP 22

2.2.4 State Abstraction 23
2.2.5 Sparse Sampling Methods 25

2.2.5.1 Kearns et al.'s Sparse Sampling 25
2.2.5.2 Monte-Carlo Tree Search and UCT 27
2.2.5.3 Trial-based Heuristic Tree-Search 30

2.2.6 Solvers with Alternative Outcome Selection 32

3. $\mathcal{M}_l^k$-REDUCTIONS - GENERALIZING DETERMINIZATION 34

3.1 A Broad Spectrum of MDP Model Reductions 34
3.2 Planning for More than k Exceptions 39

3.2.1 Continual Planning Using Reduced Models 39
3.2.2 Evaluating the Performance of the Continual Planning Approach 41

3.3 The Choice of Reduced Model Matters 43

3.3.1 The Value of Going Beyond Single-Outcome Determinization 43
3.3.2 Choosing the Right Outcomes 46

3.4 A Greedy Approach for Learning Reduced Models 48
3.5 Experimental Results 53

3.5.1 Evaluating $\boldsymbol{\mathcal{M}_l^k}$-REPLAN 53

4. COMBINING $\mathcal{M}_l^k$-REDUCTIONS WITH CLASSICAL PLANNING TECHNIQUES 59

4.1 FF-LAO*: Leveraging classical planners 59

4.1.1 Handling plan deviations during execution 61
4.1.2 Theoretical considerations 62
4.1.3 Learning a Good Determinization 64

4.2 Experiments 66

4.2.1 Domains and methodology 66
4.2.2 Results and Discussion 68

5. A NEW LABELING MECHANISM FOR EFFICIENT STATE EXPLORATION 71

5.1 An MDP Solver Based on Short-Sighted Labeling 71

5.1.1 The FLARES Algorithm 71

5.1.2 Illustrative Example: A Simple Grid World Problem 78

5.2 Soft Labeling in SSPs 80

5.2.1 Generalizing Labeling 80
5.2.2 Soft Labeling 84

5.3 Experiments 90

5.3.1 Sailing Domain 92
5.3.2 Racetrack Domain 94
5.3.3 International Planning Competition Domains 94

6. CONCLUSION 96

6.1 Summary of Contributions 96

6.1.1 $\mathcal{M}_l^k$-reductions 96
6.1.2 Soft Labeling 98

6.2 Future Work 99

BIBLIOGRAPHY 100

LIST OF TABLES

Table **Page**

3.1 Comparison of the best determinization (M_1) and the best $\mathcal{M}_2^0$-reduction (M_2) for three racetrack problems. 45

3.2 Number of successful trials, out of a maximum of 50, using two different $\mathcal{M}_1^0$-reductions on ten TRIANGLE-TIREWORLD problems. 46

3.3 Expected cost and average planning time obtained with several reduced models of the racetrack domain. 55

5.1 Results on the grid world shown in Figure 5.2. 79

5.2 Expected cost and total planning of several planning algorithms on the sailing domain (middle-goal). 91

5.3 Expected cost and total planning of several planning algorithms on an instance of the sailing domain (corner-goal). 91

5.4 Expected cost and total planning of several planning algorithms on the racetrack domain (ring-5). 93

5.5 Expected cost and total planning of several planning algorithms on the racetrack domain (square-4). 93

5.6 Number of runs in which planners were able to successfully reach the goal for two IPPC'08 domains. 94

LIST OF FIGURES

Figure **Page**

3.1 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. 37

3.2 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. 44

3.3 Three instances of the racetrack domain. 45

3.4 Three instances of the TRIANGLE-TIREWORLD domain. Locations with spare tires are marked in black (Little and Thiebaux, 2007). 47

3.5 Example showing that $-V_{cp}^{M}(\langle s_0, k\rangle)$ is not submodular. Actions A and B have cost 1. 50

3.6 An instance of the racetrack domain. 54

3.7 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 3.6. 56

4.1 Number of solved rounds by 5 different planners in IPPC'08 benchmarks. 69

5.1 Problem with large optimal policy but small high-probability envelope (S: start state, G: goal, scale shows log-probability). 72

5.2 Grid world illustrating the advantages of FLARES. 79

5.3 Illustration of the ϵ-distance of a state under a given policy, for two different distance functions. 83

CHAPTER 1
INTRODUCTION

1.1 Overview

The ability to form a sequence of actions for achieving a goal is one of the most distinctive features that characterize humans. Except for the most trivial tasks, every time we make a decision, most of us typically evoke some form of internal model—perhaps simple and inaccurate—that predicts the consequences of our potential decisions. Using predictions produced by this model, we are then able to decide the most favorable plan of action, at least under the assumptions that the model implies. In the Artificial Intelligence (AI) literature, this model-based approach to action selection is known as planning (Tate and Hendler, 1994; Weld, 1999).

Most interesting real-world planning scenarios involve some form of uncertainty. Actions may fail, sensing capabilities might be imperfect, and the environment could respond to events in a multitude of possible ways. In general, agents operating in the real world have to consider the possibility of unexpected events occurring. The problem of producing intelligent behavior in these situations is known as *planning under uncertainty*, or *probabilistic reasoning*.

In the presence of uncertainty, a planning problem can be modeled as a Markov Decision Process (MDP) (Puterman, 1994). This model has been widely used in AI for planning (Kaelbling et al., 1998; Kolobov and Mausam, 2012) and learning (Sutton and Barto, 1998b) under uncertainty, with applications including mobile robots (Koenig et al., 1996; Thrun et al., 2005; Hsu et al., 2006), flow assignment in heterogenous networks (Singh et al., 2010), aircraft collision avoidance (Temizer et al., 2010), and semi-autonomous driving (Wray et al., 2016). Unfortunately, despite the high expressiveness of the framework, solving MDPs is computationally demanding, a difficulty that has limited its application to real-world scenarios.

Early MDP solution methods produced a decision (an *action*) for every possible situation (or *state*) the agent could encounter (Bellman, 1957; Howard, 1960). In more recent AI literature, planning methods have been refined so that only a subset of all states need to be considered to produce optimal solutions; this is important, since the complexity of planning is directly related to the number of states the plan needs to account for. The pioneering examples of this refinement approach, based on *heuristic search*, are the RTDP (Barto et al., 1995), LAO* (Hansen and Zilberstein, 2001) and LRTDP (Bonet and Geffner, 2003a) algorithms. Unfortunately, while these methods can drastically reduce the number of states that need to be considered during planning, their computational complexity is still in the order of the total number of states in the problem, a set whose cardinality is exponential in the number of variables that describe the problem (Littman, 1997). This is not only a theoretical concern, since most interesting real world applications involve a large number of state variables.

1.2 Brief Overview of Solution Methods for MDPs

There is a vast body of work concerned with solution methods for MDPs, spanning decades of research in control, operations research and planning. A comprehensive survey of the existing work is thus outside of the scope of this thesis. Nevertheless, we will provide a brief high level summary of the most popular solution approaches in the planning literature; a detailed description of the methods summarized in this section can be found in (Kolobov and Mausam, 2012). In Chapter 2, we provide a deeper discussion on the subset of MDP solution approaches most related to our work. Importantly, the discussion here and in Chapter 2 will be centered around

MDPs with discrete state and action spaces, which has been the most popular model used in planning research.

Many algorithms for solving MDPs require the computation of a value function, which maps states to an estimate of the optimal expected utility that can be obtained from that state; the output of the algorithm is a policy, which is a function that maps states to actions. The two must fundamental algorithms for finding optimal solutions to MDPs are Value Iteration (Bellman, 1957) and Policy Iteration (Howard, 1960).

Both of these algorithms are optimal, but they require computing state values for all of the states in the MDP, a cost that is clearly not scalable, particularly in high dimensional problems. An important improvement over these methods arose with the introduction of Asynchronous Value Iteration (Bertsekas and Tsitsiklis, 1989). Rather than iteratively updating the values of all states, Asynchronous VI works by iteratively updating the value of an arbitrarily chosen state. The crucial result was proving that this method converges to an optimal solution, as long as no state gets starved, i.e., all states have values updated an infinite number of times.

Asynchronous VI prompted a variety of improvements to VI. Some prominent examples are Prioritized Sweeping (Moore and Atkeson, 1993), Prioritized VI (Wingate and Seppi, 2005), and Topological VI (Dai and Goldsmith, 2007). All of these algorithms attempt to find a good ordering of states for performing value updates, in order to accelerate convergence to an optimal (or near-optimal) solution. However, all of these methods still require an amount of computation and memory proportional to the MDP's state space size.

An important development was the introduction of heuristic search algorithms, briefly mentioned in the previous section. These methods rely on a heuristic function (an optimistic bound on state values) to direct a graph search procedure towards the more relevant parts of the state space. Generally, these approaches consider computing a plan that starts from a given initial state, and the search procedure only considers states reachable from that state. The search is directed by choosing actions greedily on the current value estimates, initialized using the heuristic function. Since value updates are only performed in the states that are visited, as opposed to all possible states, this approach potentially results in large computational gains, particularly with accurate heuristics. The seminal methods in this area are RTDP (Barto et al., 1995), LAO* (Hansen and Zilberstein, 2001) and LRTDP (Bonet and Geffner, 2003a). Other extensions are HDP (Bonet and Geffner, 2003b), BRTDP (McMahan et al., 2005), FRTDP (Smith and Simmons, 2006), and VPI-RTDP (Sanner et al., 2009). We provide a more detailed description of these methods in Chapter 2.

While generally more efficient than VI variants, heuristic search methods still suffer from scalability issues. In particular, any optimal algorithm must, at a minimum, compute an action for every state reachable by an optimal policy. This incurs both a cost in terms of memory and time, which can be impractical in very large problems. Attempts to address this difficulties has led to the use of *abstraction* (Mccallum, 1993; Ravindran and Barto, 2002; Givan et al., 2003; Li et al., 2006) and *symbolic* algorithms (Hoey et al., 1999; Feng and Hansen, 2002), which seek to group similar states and perform value updates that modify values for large groups of states at once. However, these reduction approaches have seen limited use, as most of the work for the past decade has focused on the development of approximate algorithms for solving MDPs, which have scaled to much larger problems in practice.

There is a wide variety of algorithms for approximately solving MDPs. The most popular ones in the planning literature can be roughly categorized into one of the following general approaches: *determinization*, *short-sightedness*, *sparse sampling*, and *dimensionality-reduction*. A common thread among approximate MDP solvers is the use of *re-planning*, also known as *online* planning. Concretely, an online algorithm is concerned with repeatedly finding a good action for the current state of execution, rather than finding a complete plan of action from the start state to a goal, requiring no additional computation afterwards.

Determinization-based algorithms became popular in the mid-late 2000s, after the surprising success of an algorithm called FF-REPLAN (Yoon et al., 2007) in the first International Probabilistic Planning Competition (Younes et al., 2005). The idea was simple: create a deterministic version of the MDP (e.g., always choose the most probable outcome), and solve it with a highly efficient deterministic planner. This allows FF-REPLAN to scale to extremely large problems, at the cost of reduced policy quality (which can be arbitrarily bad in the worst case). To address this drawback, more robust determinization-based algorithms have been developed, such as HMDPP (Keyder and Geffner, 2008b), RFF (Teichteil-Königsbuch et al., 2010), FF-HINDSIGHT (Yoon et al., 2008, 2010). Notably, a great part of this thesis is concerned with directly addressing some of the drawbacks of determinization-based methods.

Short-sighted algorithms (sometimes referred to as *myopic*) work by restricting computation to only a subset of states close to the current state of execution. The earliest well-known short-sighted algorithm for solving MDPs is HDP(I,J) (Bonet and Geffner, 2003b), which creates an envelope of states by following the most likely outcomes of actions. A more recent short-sighted algorithm is SSIPP and its variants (Trevizan and Veloso, 2014). This algorithm works by restricting the search to states reachable in a small number of steps from the current state. There is also a trajectory-based variant that restricts the search to states reachable by trajectories whose probability is within some predefined threshold. Chapter 5 discusses limitations of existing short-sighted solvers.

Sparse sampling algorithms have recently become popular, particularly in problems in which the number of outcomes of an action is very large. The algorithms described earlier typically rely on dynamic programming value updates, whose complexity is linear in the number of outcomes. In problems in which this number is very large, a more practical alternative is to sample the outcomes and update values using Monte Carlo averaging. The more popular examples of this approach are SS (Kearns et al., 2002), FSSS (Walsh et al., 2010), UCT (Kocsis and Szepesvári, 2006), and the THTS framework (Keller and Helmert, 2013). The THTS framework, in particular, was the first work, to the best of our knowledge, that explicitly formulated the *outcome selection* problem for sampling algorithms; that is, to devise a mechanism to choose outcomes during the search process to result in improved performance. Indeed, the outcome selection problem is one of the central question of this thesis, and all of our methods can be positioned as attempts to answer this question.

Finally, we briefly describe the dimensionality-reduction approach to solving MDP. In contrast with the algorithms described above, which attempt to reduce the amount of search needed to solve an MDP, a dimensionality reduction algorithm attempts to represent the optimal value function (or the policy) in a parameterized way (e.g., as a linear combination of state features), and then tries to find an optimal value for the parameters. This changes the complexity of solving the problem so that it is now dependent on the number of parameters, rather than the number of states of the problem. On the other hand, the quality of the resulting policy is dependent on the expressiveness of the value/policy representation.

Some examples of dimensionality-reduction algorithms in the planning literature are the approximate versions of Value and Policy Iteration (Guestrin et al., 2003), and, more generally, a wide variety of approximate dynamic programming methods (de Farias and Van Roy, 2003; Powell, 2007). In more recent years, the approach has fell out of favor in the planning community, due to the focus on domains factored using symbolic languages, for which the other techniques described above have worked better in practice. Some recent dimensionality-reduction examples within this context are the FPG (Buffet and Aberdeen, 2009) and RETRASE (Kolobov et al., 2009) planners. However, we note that the dimensionality-reduction approach is the backbone of the vast amount of research in reinforcement learning (Bertsekas and Tsitsiklis, 1996; Sutton and Barto, 1998a), which has gained popularity in recent years after its success in solving complex high dimensional spaces, such as the game of Go (Silver et al., 2016), and learning to play video games directly from pixels (Mnih et al., 2015).

1.3 Summary of Contributions

In this thesis, we address the limitations of existing reduction approaches for solving Markov Decision Process, particularly, determinization and short-sightedness, by developing efficient and adaptable frameworks for state-space exploration in probabilistic planning. The result is a series of algorithmic and mathematical techniques that consistently produce near-optimal plans with little computation. In more detail, our main contributions are the following:

- **A scalable and robust reduction paradigm for Markov Decision Processes**. We introduce the $\mathcal{M}_l^k$-reduction for MDPs, a reduction paradigm that generalizes single-outcome determinization as just one extreme point from a spectrum of MDP reductions that differ along two dimensions: i) the number of outcomes per state-action pair that are fully accounted for in the reduced model, and ii) the number of occurrences of the remaining, *exceptional*, outcomes that are planned for in advance. For example, a single-outcome determinization can be represented in this framework as an instance of an $\mathcal{M}_1^0$-reduction, but more robust reductions can be created by increasing the number of exceptions handled by the planner.

- **Methods for learning $\mathcal{M}_l^k$-reductions automatically**. A crucial insight arising from this thesis is the fact that the choice of reduction can have a substantial impact on the quality of the resulting plans. We show that in many problems the choice of reduction makes the difference between catastrophic and optimal behavior, even when using determinization. Building on this observation, we introduce two methods to automatically choose an $\mathcal{M}_l^k$-reduction that can be used to produce high quality plans for a given MDP. The first is a greedy algorithm that adds a single outcome to the set of exceptions at every iteration, stopping when the quality of the resulting plan falls below a certain threshold. Furthermore, we show that it is possible to learn a reduction on a small problem instance, and apply the same reduction to a larger instance to produce near-optimal policies efficiently. The second method is an exhaustive approach for finding determinizations in MDPs described using factored domain languages.

- **Sampling-based algorithms for efficient state-space exploration**. We also introduce an algorithmic framework for solving MDPs that does not require modifying the model used for planning. We introduce the notion of short-sighted soft labeling, a generalized version of popular sampling algorithms that rely on state labeling. We show that short-sighted labeling can be used to compute approximate plans very efficiently, while still achieving deeper exploration (and thus, better policy quality) than previous short-sighted approaches. Moreover, our notion of soft-labeling allows us to provide theoretical guarantees, while also improving efficiency by biasing the search. In particular, a soft labeling algorithm modifies the transition function used for sampling, so that the search is biased towards states for which further computation is more likely to improve policy quality. Our experiments show that soft labeling delivers state-of-the-art performance in a wide variety of popular benchmark domains.

1.4 Relevant Publications

The work presented in this thesis builds upon previous publications:

- L. Pineda, Y. Lu, S. Zilberstein, and C. V. Goldman. Fault-tolerant planning under uncertainty. *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, 2013 (Pineda et al., 2013).

- L. Pineda and S. Zilberstein. Planning under uncertainty using reduced models: Revisiting determinization. *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling*, 2014 (Pineda and Zilberstein, 2014).

- L. Pineda, T. Takahashi, H. Jung, S. Zilberstein, and R. Grupen. Continual Planning for Search and Rescue Robots. *Proceedings of the IEEE-RAS 15th International Conference on Humanoid Robots*, 2015 (Pineda et al., 2015).

- L. Pineda, K. H. Wray, and S. Zilberstein. Fast SSP Solvers Using Short-Sighted Labeling. *Proceedings of the Thirty-First Conference on Artificial Intelligence*, 2017 (Pineda et al., 2017).

1.5 Thesis organization

The rest of this thesis is organized as follows. Chapter 2 presents mathematical background, as well as a review of methods for solving Markov Decision Processes. The review covers determinization, state-abstraction, sparse sampling methods, and methods combining heuristic search with biased sampling. Chapter 3 introduces the $\mathcal{M}_l^k$-reduction, a greedy algorithm to choose reduced models, and experimental results. Chapter 4 presents an algorithm that combines $\mathcal{M}_l^k$-reductions with classical planning, allowing it to scale to problems with billions of states, as we show in our experiments; additionally, this chapter also presents the automatic approach for choosing determinizations using a factored domain language representation. Finally, Chapter 5 presents our work on soft-labeling, first describing the benefits associated with the use of short-sighted labeling, as opposed to using short-sightedness or labeling individually. We then explain our general soft labeling framework, analyze its theoretical properties, and empirically evaluate its benefits. We conclude in Chapter 6, with a summary of this work and directions for future research.

CHAPTER 2
BACKGROUND

In this chapter, we formally define the Markov Decision Process (MDP) and Stochastic Shortest Path (SSP) models. We then present a brief survey of the most popular solution methods for MDPs and SSPs in planning.

2.1 Markov Decision Process

The MDP model encapsulates a wide variety of sequential decision problems under uncertainty. The common elements among these are the concepts of *states*, *actions*, *transition function*, and *reward/cost function*; we will describe these in more detail below. On the other hand, MDP varieties differ in elements such as whether the system dynamics are continuous or discrete, or whether execution time is finite or not (Kolobov and Mausam, 2012). In this thesis we will primarily focus on a class of MDPs known as Stochastic Shortest Path problems (SSPs) (Bertsekas and Tsitsiklis, 1991), which generalizes other well-known discrete MDP sub-classes such as Finite-Horizon MDPs and Infinite Horizon Discounted-Reward MPDs (Puterman, 1994; Bertsekas and Tsitsiklis, 1996).

Definition 1. ***Stochastic Shortest Path problem****. A Stochastic Shortest Path problem is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{C}, s_0, \mathcal{G} \rangle$, where:*

- *$\mathcal{S}$ is the finite set of all possible* ***states*** *of the system,*
- *$\mathcal{A}$ is the finite set of all possible* ***actions*** *the agent can take,*

- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ *is a* ***transition function*** *specifying the probability* $\mathcal{T}(s, a, s')$ *of going to state* s' *whenever action* a *is executed in state* s*,*

- $\mathcal{C} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ *is a* ***cost function*** *that gives the cost* $\mathcal{C}(s, a)$ *incurred whenever the agent executes action* a *and the system is in state* s*,*

- $s_0 \in \mathcal{S}$ *is the* ***initial state*** *of the system, and*

- $\mathcal{G} \subseteq \mathcal{S}$ *is the non-empty set of* ***goal states****, s.t. for every* $s_g \in \mathcal{G}$*, for all* $a \in \mathcal{A}$*, and for all* $s' \neq s_g$*, the transition function obeys* $\mathcal{T}(s_g, a, s_g) = 1$*,* $\mathcal{T}(s_g, a, s') = 0$*, and* $\mathcal{C}(s_g, a, s_g) = 0$*,*

Informally, in an SSP, at every discrete time step the system is in some state $s \in \mathcal{S}$, the agent executes an action $a \in \mathcal{A}$, and this moves the system to state $s' \in \mathcal{S}$ with probability $\mathcal{T}(s, a, s')$, incurring cost $\mathcal{C}(s, a)$; the objective is to bring the system from the initial state s_0 to a goal state $s_g \in \mathcal{G}$ with minimum total cost, in expectation.

The behavior of an agent is described in terms of a *policy*, which in the broadest sense maps a history of interactions to a probability distribution over actions. However, for optimal behavior in SSPs it suffices to consider policies that are *Markovian*, *stationary*, and *deterministic*. That is, where the actions: are non-random functions of the current state (deterministic), ignore the previous history of states (Markovian), and ignore the notion of time (stationary). Under these constraints, we can say that a solution to an SSP is a **policy**, a mapping $\pi : \mathcal{S} \rightarrow \mathcal{A}$. An **execution history** that terminates at state s for policy π, is a sequence of tuples $h_s = ((s_0, \pi(s_0)), (s_1, \pi(s_1)), ..., (s_{t-1}, \pi(s_{t-1})), s)$ of pairs of states the agent has visited and actions the agent has taken in those states, plus the state s in which the history ends. Execution histories are also sometimes referred to as **trials** or **episodes**. The following type of policy plays an essential role in SSPs (Kolobov and Mausam, 2012).

Definition 2. ***Proper policy***. *For a given SSP $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{C}, s_0, \mathcal{G} \rangle$, let h_s be an execution history that terminates at state s. For a given set $S' \in \mathcal{S}$, let $P_t^\pi(h_s, S')$ be the probability that after execution history h_s, the agent transitions to some state in S' within t time steps if it follows policy π. A policy π is called* ***proper at state*** *s if $\lim_{t\to\infty} P_t^\pi(h_s, \mathcal{G}) = 1$ for all histories that terminate at s. If for some h_s, $\lim_{t\to\infty} P_t^\pi(h_s, \mathcal{G}) < 1$, π is called* ***improper at state s****. A policy π is called* ***proper*** *if it is proper at all states $s \in \mathcal{S}$. Otherwise, it is called* ***improper****.*

Given a policy π, we can define the **value function** V^π that represents the expected total cost incurred when π is executed starting from state s. That is,

$$V^\pi(s) \triangleq \mathbb{E}\Big[\sum_{t=0}^{\infty} \mathcal{C}(s_{t+k}, \pi(s_{t+k}))|s_t = s, \pi\Big] \tag{2.1}$$

An optimal solution to an SSP, or **optimal policy**, denoted as π^*, and its **optimal value function**, V^*, are ones that satisfy,

$$V^*(s) = \min_\pi V^\pi(s) \tag{2.2}$$

and, for all $s \in \mathcal{S}$,

$$V^*(s) = \min_a \Big[\mathcal{C}(s,a) + \sum_{s'\in\mathcal{S}} \mathcal{T}(s,a,s')V^*(s')\Big] \tag{2.3}$$

$$\pi^*(s) = \arg\min_a \Big[\mathcal{C}(s,a) + \sum_{s'\in\mathcal{S}} \mathcal{T}(s,a,s')V^*(s')\Big] \tag{2.4}$$

as long as the following two conditions are satisfied (Bertsekas and Tsitsiklis, 1991):

- There exists at least one proper policy,
- For every improper policy π, and for every state $s \in \mathcal{S}$ where π is improper, $V^\pi(s) = \infty$.

One simple and intuitive case in which the second condition is satisfied is when $\mathcal{C}(s,a) > 0$ for all $s \notin \mathcal{G}$, and for all actions $a \in \mathcal{A}$, which is analogous to the condition for correctness of Dijkstra's algorithm for the problem of finding shortest paths in graphs—the deterministic counterpart of SSPs.

2.2 Methods for Solving Markov Decision Processes

In this section we describe several methods for solving MDPs. We begin our discussion with an overview of Value Iteration (VI), one of the fundamental methods for solving MDPs optimally. During our description of VI, we introduce several important concepts associated with more advanced solution techniques. After describing VI, we move on to a brief survey of several approximation methods for solving MDPs.

2.2.1 Fundamental Solution Methods for MDPs

In this section we describe **Value Iteration** (Bellman, 1957), a fundamental algorithm for finding optimal solutions to SSPs that forms the basis for the large majority of the state-of-the-art algorithms for solving SSPs/MDPs. The correctness of this algorithm for SSPs stems from an important result by Bertsekas and Tsitsiklis (Bertsekas and Tsitsiklis, 1991), who showed that under the two conditions listed above, the optimal value function for an SSP, V^*, is the fixed point of the set of *Bellman equations*,

$$V(s) = \min_a \left[\mathcal{C}(s,a) + \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s')V(s')\right] \tag{2.5}$$

Although Value Iteration (VI) follows directly from the Bellman equations, it is useful to introduce the concept of **Q-value under a value function** of a state-action pair, defined as

$$Q^V(s,a) \triangleq \mathcal{C}(s,a) + \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s')V(s') \tag{2.6}$$

and the **optimal Q-value** of a state-action pair, defined as

$$Q^*(s,a) \triangleq \mathcal{C}(s,a) + \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s')V^*(s') \tag{2.7}$$

In simple terms, $Q^*(s,a)$ is the expected cost the agent obtains by taking action a in state s and following the optimal policy thereafter. VI finds the optimal value function of an SSP by repeatedly applying the Bellman equation as an operator that improves our current estimate of the optimal value function. Concretely, the *Bellman update* or *Bellman backup* operator is defined as

$$V_n(s) \leftarrow \min_a Q^{V_{n-1}}(s,a) \tag{2.8}$$

VI initializes V_0 arbitrarily and iteratively computes $V_n(s)$ for all the states in $\mathcal{S}$, in a full sweep. Convergence is defined in terms of the **residual** or *Bellman error*,

$$Res^V(s) \triangleq |V(s) - \min_a Q^{V_{n-1}}(s,a)| \tag{2.9}$$

The algorithm terminates when $Res^{V_n} \triangleq \max_{s \in \mathcal{S}} Res^{V_n}(s)$ satisfies $Res^{V_n} < \epsilon$. The **greedy policy**, π^{V^n}, which forms the solution for the SSP, is obtained from the final value function V^n as

$$\pi^{V_n}(s) \triangleq \arg\min_{a \in \mathcal{A}} Q^{V_n}(s,a) \tag{2.10}$$

Note that, while $Res^{V_n} < \epsilon$, it is not necessarily the case that $|V_n(s) - V^*(s)| < \epsilon$ for any state $s \in \mathcal{S}$. For SSPs, bounding this error is complicated, as it depends on the expected number of steps needed by the policy to reach a goal from state s. Nevertheless, in practice, VI typically obtains optimal or near-optimal policies for small ϵ values. Value functions satisfying $Res^V < \epsilon$ are said to be *ϵ-consistent*.

Bellman backups also satisfy an useful property, *monotonicity*, which means that,

$$\forall s \in \mathcal{S}, V_k(s) \leq V^*(s) \implies \forall s \in \mathcal{S}, V_{k+1}(s) \leq V^*(s) \tag{2.11}$$

$$\forall s \in \mathcal{S}, V_k(s) \geq V^*(s) \implies \forall s \in \mathcal{S}, V_{k+1}(s) \geq V^*(s) \tag{2.12}$$

A value function that satisfies the left side of Eq. (2.11) is also called an **admissible heuristic**. These are commonly used to guide search algorithms towards parts of the state space that could potentially be relevant to an optimal policy.

Before, we mentioned that VI applies a Bellman backup for all states in a full sweep, which is sometimes referred to as *synchronous VI*. However, another important property of Bellman backups is that they can be performed *asynchronously*. This leads to the **Asynchronous VI** algorithm, which at each iteration selects a *single* state $s \in \mathcal{S}$ to update $V_n(s)$ using Eq.(2.8) (in contrast to the synchronous version, which updates state values all at once). Moreover, states can be chosen in any order, with the only restriction that no state gets starved; i.e., all states are backed up an infinite number of times and infinitely often (Bertsekas and Tsitsiklis, 1989). The importance of this asynchronous property cannot be overstated: most search-based methods for solving SSP/MDPs can be seen as variants of Asynchronous VI with clever mechanisms for selecting the order of state updates.

2.2.2 Overview of Approximate Planning Methods for Solving MDPs

There is a large body of research on solving MDPs and SSPs, starting with the seminal work of Bellman (Bellman, 1957) and Howard (Howard, 1960) in dynamic programming (from which the VI algorithm derives). In the previous section, we discussed an algorithm that computes optimal solutions by computing optimal values for *all* states in $\mathcal{S}$. Unfortunately, many interesting scenarios involve a number of states that increases exponentially with respect to the size of factored representations of the problem domain. Consider, for instance, an MDP formulation of the problem of controlling wildfire propagation that models the terrain as a rectangular grid. If there are n cells in the grid and each has two possible states (burning and not-burning), then the number of states of the system is 2^n. With a modest-sized 10×10 grid this already gives more than 10^{30} states (for reference, a petabyte is 10^{18} bytes).

Therefore, in this thesis we are mostly concerned with approximate methods for solving MDPs. Among these, the large majority of methods proposed in the planning literature attempt to reduce the number of states that need to be considered during planning. Some of them do it by directly modifying the problem and creating a simplified, or *reduced*, MDP. The hope is that the resulting problem is much easier to solve, but that the resulting policy still performs near-optimally when applied to the original problem. This is the case of determinization-based methods and state abstraction methods; we devote Sections 2.2.3 and 2.2.4 to these, respectively.

Another way of reducing the number of states to be considered is by performing *sparse sampling*, a technique that avoids having to enumerate all the states to compute Q-value estimates. Instead, they sample the transition function sparsely and perform rollouts (i.e., simulations of the current policy) and aggregate the accumulated rewards in different ways (most typically using Monte-Carlo averaging) to produce an improved policy. We devote Section 2.2.5 to this type of methods.

Finally, we conclude the literature review in Section 2.2.6, by describing a set of sampling-based algorithms that use an alternative transition function to guide their search (as opposed to sampling from the transition function of the problem). These methods bear some similarities to the soft labeling framework that we introduce in Chapter 5, and are thus an important point of reference for comparison.

2.2.3 Determinization-based Planners

Arguably, the simplest form of reduction one can do to an MDP is to completely ignore stochasticity and assume the agent has total control over action outcomes. This is the main idea behind determinization-based approaches. We start our discussion by formally defining the concept of determinization. For that, it will be useful to introduce the notion of *successor set* of a state-action pair for SSPs. Given an SSP $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{C}, s_0, \mathcal{G} \rangle$, the transition function induces a successor set of a state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$, defined as $\mathsf{succ}(s, a) \triangleq \{s' | \mathcal{T}(s, a, s') > 0\}$. We say that state s' is a **successor** of state-action pair (s, a) if $s' \in \mathsf{succ}(s, a)$. We also refer to successors as **outcomes** of an action.

Definition 3. ***Deterministic SSP***. *Given an SSP $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{C}, s_0, \mathcal{G} \rangle$, an action $a \in \mathcal{A}$ is called a **deterministic action at state s** if it satisfies $|\mathsf{succ}(s, a)| = 1$. An action is called a **deterministic action** if it is deterministic at all states $s \in \mathcal{S}$. An SSP is called a **deterministic SSP** if all actions in $\mathcal{A}$ are deterministic.*

Definition 4. ***Determinization***. *Given an SSP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{C}, s_0, \mathcal{G} \rangle$ and a tuple $\langle \mathcal{M}^d, \delta \rangle$, a deterministic SSP $\mathcal{M}^d = \langle \mathcal{S}, \mathcal{A}^d, \mathcal{T}^d, \mathcal{C}^d, s_0, \mathcal{G} \rangle$ and injective mapping $\delta : \mathcal{A}^d \rightarrow \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ satisfying $\mathcal{T}^d(s, a^d, s') = 1$ iff $\delta(a^d) = (s, a, s')$, is called a **determinization** of $\mathcal{M}$.*

In simple terms, a determinization of SSP $\mathcal{M}$ is a deterministic SSP, $\mathcal{M}^d$, with the same state space as $\mathcal{M}$, and whose deterministic actions are each mapped to a successor of some state-action pair in $\mathcal{M}$. Several of the existing determinization-based planners use one of the following two determinizations.

Definition 5. ***Most-likely-outcome determinization***. *Given an SSP $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{C}, s_0, \mathcal{G}\}$ and its determinization $\mathcal{M}^d$, we call $\mathcal{M}^d$ the **most-likely-outcome determinization** of $\mathcal{M}$ iff,*

- $|A^d| = |\mathcal{S} \times \mathcal{A}|$, *and*
- *for all $(s, a, s') \in \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ s.t. $s' = \arg\max_{s'' \in \mathsf{succ}(s,a)} \mathcal{T}(s, a, s'')$, there exists $a^d \in \mathcal{A}^d$ s.t. $\delta(a^d) = (s, a, s')$.*

Definition 6. ***All-outcomes determinization***. *Given an SSP $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{C}, s_0, \mathcal{G}\}$ and its determinization $\mathcal{M}^d$, we call $\mathcal{M}^d$ the* ***all-outcomes determinization*** *of $\mathcal{M}$ iff,*

- $|A^d| = |\mathcal{S} \times \mathcal{A} \times \mathcal{S}|$, *and*
- *for all $(s, a, s') \in \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ s.t. $s' \in \mathsf{succ}(s, a)$, there exists $a^d \in \mathcal{A}^d$ s.t. $\delta(a^d) = (s, a, s')$.*

Put simply, the most-likely-outcome determinization is the one where each state-action pair in the original SSP is replaced with a deterministic action leading to its most probable successor. On the other hand, the all-outcomes determinization is the one where each successor of a state-action pair has its own associated action.

2.2.3.1 FF-Replan

FF-REPLAN (Yoon et al., 2007) was the catalyst for sparking interest in determinization-based approaches, after its surprising success in the 1st International Probabilistic Planning Competition (IPPC) (Younes et al., 2005) in 2004. The idea is very simple: construct a determinization of the original SSP, solve it using an efficient classical planner (in this case the FF planner (Hoffmann and Nebel, 2001)), and execute the plan. If, during execution, a state not covered by the current plan is encountered, the process is repeated starting from the current state. This is where the name FF-*Replan* comes from.

The earliest version of FF-REPLAN, which won the aforementioned competition, used the most-likely-outcome determinization. It works particularly well in problems where there is a high probable path leading towards a goal, and where deviations from this path are not too costly. On the other hand, it can completely fail to produce plans in problems where it is impossible to reach a goal following only most-likely outcomes. This can be alleviated by employing the all-outcomes determinization, which ensures that FF-REPLAN produces a sequence of actions with non-zero probability of reaching the goal. However, both versions suffer when the cost of plan deviations are high. This is particularly evident in problems that contain so-called *dead-ends*; states from which there is no sequence of actions that will lead to a goal. Intuitively, completely ignoring some outcomes can make FF-REPLAN being overly optimistic about the effects of actions, leading the agent to potentially dangerous paths with no consideration for the corresponding consequences.

2.2.3.2 FF-Hindsight

The FF-HINDSIGHT planner (Yoon et al., 2008) seeks to mitigate the consequences of using determinization by accounting for many possible determinizations at once. Intuitively, when choosing an action, instead of considering the first action in a plan that consists of a single path to the goal, it makes sense to take an action that, on average, serves as a starting point of multiple such paths. This is the idea behind FF-HINDSIGHT.

This algorithm takes two parameters: T, the number of lookahead steps, and W, the number of deterministic paths, or *futures*, to be considered. To select an action for state s, FF-HINDSIGHT samples multiple futures, each of which is non-stationary determinization (i.e., one where the state set includes time) of the original problem, constructed by sampling action successors for times $t = 1, ..., T$ from the original transition function, and assigning a deterministic action for each of these successors. Solving a future means finding a plan, using the FF planner, for the deterministic non-stationary problem.

After solving W futures, FF-HINDSIGHT computes an estimate of the optimal Q-value of an action by averaging the costs of the plans obtained for each of the futures (including a large penalty for futures it could not solve). It then selects for execution the action with the lowest Q-value estimate, and repeats the process for the next state. By reasoning about multiple scenarios, FF-HINDSIGHT mitigates some of the issues associated with the use of determinization, albeit at the cost of increasing computational cost—it requires solving $W \cdot |\mathcal{A}|$ determinizations at each state, instead of just one. Some improvements to address this issue were introduced in later work (Yoon et al., 2010). It is also worth mentioning a recent variant of this hindsight optimization approach that does not rely on FF but uses an Integer Linear Programming formulation instead (Issakkimuthu et al., 2015).

2.2.3.3 RFF

RFF (Robust FF) (Teichteil-Königsbuch et al., 2010), which won the Third Probabilistic Planning Competition (Bryce and Buffet, 2008), is a planner that seeks to construct plans with low probability of requiring re-planning, by iteratively extending the envelope of states covered by the plan. Concretely, RFF maintains an envelope of "solved" states, which is initialized to only include s_0. The first step is to find a plan to reach a goal from s_0, by calling the FF planner, and adding to the envelope all states in the solution path. Then, it creates a set of fringe states, consisting of the successors of states in the envelope for which no plan has been found yet. It then computes the probability of reaching a fringe state by using Monte-Carlo sampling. If this probability is lower than a parameter ϵ, it proceeds to execute the resulting plan. Otherwise, it goes back to the first step, but computing plans for *all* fringe states. In this manner, the envelope is increased at each iteration, until the probability of leaving the envelope is lower than ϵ.

While this version of the RFF planner is able to anticipate the most likely consequences of its plan, it suffers from the same set of drawbacks as FF-REPLAN—the final plan is just an aggregation of sub-plans that ignore the probabilistic nature of the problem. A more sophisticated version of the algorithm performs Bellman backups on the states in the envelop and uses this to inform the selection of actions for this states; however, the computational cost of this modification can be much higher.

2.2.3.4 HMDPP

The HMDPP planner (Keyder and Geffner, 2008b) takes a different approach to determinization than the planners previously discussed. It accounts for the probabilistic nature of the SSP by directly modifying the cost function used in an all-outcomes determinization, with the goal of discouraging actions associated to outcomes that have low probability of success. It does this by relying on the *self-loop determinization* of an SSP.

Definition 7. ***Self-loop determinization.*** *Given an SSP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{C}, s_0, \mathcal{G} \rangle$ and an all-outcomes determinization $\mathcal{M}^{sl} = \langle \mathcal{S}, \mathcal{A}^{sl}, \mathcal{T}^{sl}, \mathcal{C}^{sl}, s_0, \mathcal{G} \rangle$, we call $\mathcal{M}^{sl}$ the* ***self-loop determinization*** *of $\mathcal{M}$ iff its cost function is modified so that it satisfies $\mathcal{C}^{sl}(s, a^d) = \frac{\mathcal{C}(\delta(a^d))}{\mathcal{T}(\delta(a^d))}$*[1].

To see why this modified cost-function makes sense, consider a (self-loop) SSP where the only source of uncertainty is whether actions succeed or leave the state of the system unchanged. Uncertainty is reduced in this case to the number of times an action needs to be applied to obtain the desired outcome. The authors of HMDPP show that the optimal value function for such an SSP is the same as that of its self-loop determinization. Therefore, for an arbitrary SSP, computing an optimal solution to its self-loop determinization is equivalent to optimally solving a *self-loop relaxation* of the original problem.

The other difference with respect to the previous determinization-based planners is that HMDPP does not actually solve the deterministic problem. Instead it employs a cost-sensitive heuristic from the classical planning literature (Bonet and Geffner, 2001; Keyder and Geffner, 2008a) to get an estimate for the value of a state, h^{sl}_{add}. Additionally, it uses another heuristic, h_{pdb}, derived from an abstract and computationally tractable SSP, defined by abstracting states into patterns, which is then

[1]Here we abuse the notation so that $\mathcal{C}(\delta(a^d)) = \mathcal{C}(s, a)$ when $\delta(a^d) = (s, a, s')$.

solved using VI. The h^{sl}_{add} heuristic scales up well and provides guidance towards the goal, while h_{pdb} is harder to compute but can identify high-risk states that need to be avoided. They integrate the heuristics in a lexicographical ordering, by obtaining first the actions that minimize the expected value of h_{pdb}, and among these the one that minimizes the value of h^{sl}_{add}. Using this strategy, HMDPP outperforms RFF in a number of hard problems (Bryce and Buffet, 2008).

2.2.4 State Abstraction

Another form of reduction for MPDs/SSPs is *state abstraction.* State abstraction (or *state aggregation*) is a technique that has been extensively studied in AI and Operations Research (Li et al., 2006). In this approach, planning is performed in an *abstract state space*, in which the original states are grouped according to some criteria, so that the space of the abstracted MDP is smaller than the original one, and thus easier to work with. Several state abstraction approaches have been proposed in the planning and reinforcement learning communities, for instance, bisimulation (Givan et al., 2003), homomorphism (Ravindran and Barto, 2002), utile distinction (Mccallum, 1993), and policy irrelevance (Jong and Stone, 2005).

Much of the work in abstraction methods for MDPs has focused on bisimulation, a notion of equivalence between states that preserves one-step rewards and transitions (Givan et al., 2003). This concept leads naturally to a *model minimization* paradigm (Givan et al., 2003), in which states that are equivalent under a bisimulation are aggregated into a single abstract state. A related notion of equivalence is homomorphism (Ravindran and Barto, 2002), defined as a tuple of surjections–one over states, one over actions–that also preserve transition and reward structures; moreover, it has the advantage over bisimulation of being able to cleanly represent state-action equivalences, which can lead to capturing coarser abstractions.

Unfortunately, while bisimulation and homomorphism can be used to produce MDP reductions that preserve optimality, they are computationally expensive to produce (NP-hard for some problem representations (Givan and Dean, 1997; Ravindran and Barto, 2002)), and have seen limited use in practice. Some recent work on bisimulation has focused on using a metrics formulation of the equivalence relation (Ferns et al., 2004, 2006; Comanici et al., 2012), which significantly speeds up computation. Yet, despite these improvements, most results so far have only scaled up to problems with a few hundred of states, which is still not practical.

Currently, the most promising results using abstraction have been obtained in the context of sparse sampling methods (Browne et al., 2012). Hostetler et al. (2015) proposed an abstraction improvement over the FSSS sampling algorithm (Walsh et al., 2010). Starting with the coarsest abstraction possible, the algorithm runs FSSS in the abstract problem, and then iteratively refines the abstraction and runs FSSS again, until time runs out. They propose a refinement procedure that encourages all states in the same abstract class to have the same optimal action. Their method has the same guarantees as the underlying sampling method used, but exploits the computational benefits of coarse abstractions when the time budget is small. Jiang et al. (Jiang et al., 2014) considered local approximate homomorphisms for UCT (Kocsis and Szepesvári, 2006) constructed from sampled trajectories. Starting from an initial trivial abstraction, the algorithm operates over a batch of trajectories using the most current abstraction, updates the abstraction after the batch is finished, and repeats. A similar work, by Anand et al. (Anand et al., 2015), introduced a more flexible notion of equivalence named ASAP (*Abstractions of State-Action Pairs*), which operates over AND-OR graphs, and is able to find coarser abstractions than those obtained from homomorphism, while still maintaining optimality. They also introduce an algorithm, ASAP-UCT, which interleaves tree expansion with abstraction computation as in (Jiang et al., 2014), and show improvements over plain UCT in problems with thousands of states. A more recent improvement, called OGA-UCT (Anand et al., 2016), computes ASAP abstractions incrementally as the UCT tree is built.

2.2.5 Sparse Sampling Methods

We use the term sparse sampling to refer to a class of methods that sample from a simulator of the MDP and compute statistical estimates of action Q-values, instead of directly using the Bellman update operator defined in Eq.(2.8). These methods have two main advantages: i) they do not require knowledge of the transition function and can work with a generative model of the problem, and ii) they do not require enumerating all the successors of state-action pairs to explore the state space.

Note that in this section we will focus on MDPs where the objective is to maximize the total discounted reward accumulated by the system over an infinite time horizon, as this is the setting where sparse sampling methods more naturally apply. However, the algorithms presented here can be applied to the SSP case, albeit losing some of their theoretical guarantees. We use the notation $\mathcal{R}$ to refer to the reward function (the analogous of the cost function $\mathcal{C}$), and γ to refer to the discount factor.

2.2.5.1 Kearns et al.'s Sparse Sampling

The sparse sampling algorithm of Kearns et al. (2002), or SS, was the first algorithm to produce near-optimal policies with *no dependence* on the size of the state space. Instead, its running time is exponential in the ϵ-horizon time, where ϵ is the error tolerance desired. More precisely, it grows at a rate of $(1/\epsilon)^{O(log(1/\epsilon))}$.

SS is relatively simple and we give its pseudocode in Algorithms 1, 2, and 3. The inputs are an error tolerance ϵ, a discount factor, γ, the maximum reward that can be obtained after executing an action, R_{max}, a generative model of the problem, $\mathcal{M}$, and the initial state s_0. The algorithm starts by computing the width, C, and the depth, H, whose values are computed according to expressions that guarantee bounded error. C is the number of successors it needs to sample at each step, while H is the depth

Algorithm 1: ESTIMATE-Q function for SS.

```
input: $h$, $C$, $\gamma$, $\mathcal{M}$, $s$
output: A list $\big(\hat{Q}^*_h(s,a_1), \hat{Q}^*_h(s,a_2), ..., \hat{Q}^*_h(s,a_k)\big)$ of Q-value estimates
1 If $h = 0$, return $(0, ..., 0)$
2 For each $a \in \mathcal{A}$, use $\mathcal{M}$ to generate $C$ samples $s' \in \mathsf{succ}(s,a)$. Let $S_a$ be the
    set containing these $C$ successors
3 For each $a \in \mathcal{A}$ let
    $\hat{Q}^*(s,a) \leftarrow \mathcal{R}(s,a) + \gamma\frac{1}{C}\sum_{s' \in S_a}$ ESTIMATE-V$(h-1, C, \gamma, \mathcal{M}, s')$
4 Return $\big(\hat{Q}^*_h(s,a_1), \hat{Q}^*_h(s,a_2), ..., \hat{Q}^*_h(s,a_k)\big)$
```

Algorithm 2: ESTIMATE-V function for SS.

```
input: $h$, $C$, $\gamma$, $\mathcal{M}$, $s$
output: A state value estimate $\hat{V}^*_h(s)$
1 $\big(\hat{Q}^*_h(s,a_1), \hat{Q}^*_h(s,a_2), ..., \hat{Q}^*_h(s,a_k)\big) \leftarrow$ ESTIMATE-Q$(h, C, \gamma, \mathcal{M}, s)$
2 Return $\max_{a \in a_1, a_2, ..., a_k}\{Q^*_h(s,a)\}$
```

up to which the algorithm explores. SS works by creating a sparse directed tree of states in depth-first fashion, sampling C successors of each action at each state in the tree. It then propagates estimates of the actions Q-values up the tree, which are computed by averaging the values observed for their successors states in the tree (line 3 in Algorithm 1). Note that this expression is equivalent to a Bellman backup when the transition function leads to each sampled successor with probability $1/C$.

This early algorithm already incorporated the main advantages of the sampling approach over enumerative approaches. It's near-optimal with running time that is independent of the size of the state space, it does not require an explicit description of the transition function, and it does not need to compute the values of all successors of a state-action pair to estimte its Q-value. On the other hand, it still suffers from some problems, particularly because it can spend a lot of time in irrelevant parts of the tree, since there is no action selection mechanism to perform adaptive exploration—instead it expands all actions at every state. There is also an improved version, FSSS algorithm (Walsh et al., 2010), which uses lower and upper bounds for values and Q-values to prune actions and accelerate the search.

Algorithm 3: Kearns et al.'s Sparse Sampling algorithm.

```
input: $\epsilon$, $\gamma$, $R_{max}$, $\mathcal{M}$, $s_0$
output: An action $a$
1 $V_{max} \leftarrow \frac{R_{max}}{1-\gamma}$
2 $H \leftarrow \lceil \log_{\gamma}(\lambda / V_{max}) \rceil$
3 $C \leftarrow \frac{V_{max}^2}{\lambda^2}\left(2H \log \frac{kHV_{max}^2}{\lambda^2} + \log \frac{R_{max}}{\lambda}\right)$
4 $\left(\hat{Q}_H^*(s,a_1), \hat{Q}_H^*(s,a_2), ..., \hat{Q}_H^*(s,a_k)\right) \leftarrow$ ESTIMATE-Q$(H, C, \gamma, G, s_0)$
5 Return $\arg\max_{a \in a_1, a_2, ..., a_k} \{Q_H^*(s,a)\}$
```

2.2.5.2 Monte-Carlo Tree Search and UCT

A different take on sparse sampling is to use a *rollout-based* approach (Kocsis and Szepesvári, 2006; Browne et al., 2012), commonly known as a Monte-Carlo Tree Search (MCTS) method. A MCTS algorithm builds its search tree by repeatedly sampling episodes from the initial state, and incrementally adding to the tree the information gathered during each episode. Estimates of the action values are kept throughout the algorithm's operation, and are reused when the same state-action is re-encountered in future episodes. These estimates can in turn be used to bias the choice of what action to follow, potentially speeding up the convergence of the value estimates.

Algorithms 4 and 5 outline a typical MCTS algorithm (Kocsis and Szepesvári, 2006). The inputs are a lookahead horizon to cut the search, H, a generative model of the problem, $\mathcal{M}$, and the initial state, s_0, for which an action is required. A MCTS algorithm iteratively generates episodes (Algorithm 5, line 1) and returns the action with the largest observed long-term reward (function BEST-ACTION). When the search arrives at the cutoff horizon, an estimate of the value of the state is returned (function EVALUATE). This estimate can be obtained by using a heuristic or by simulating a base policy for some number of episodes. In line 3 of Algorithm 4, the algorithm selects an action for exploration, which can be done through a number of different ways. The most popular one is the use of the UCB1 rule (Auer et al.,

Algorithm 4: SEARCH PROCEDURE FOR GENERIC MCTS APPROACH.

```
    input: $s, d, H, \mathcal{M}$
    output: The accumulated reward for this episode, $R$
1   If state is terminal, return 0
2   If $d = H$, EVALUATE$(s, d)$
3   $a \leftarrow$ SELECT-ACTION(S,D)
4   Use $\mathcal{M}$ to sample a successor $s' \in \mathsf{succ}(s, a)$ and the reward $R(s, a)$
5   $R \leftarrow R(s, a) + \gamma$SEARCH$(s', d + 1, H, \mathcal{M})$
6   UPDATE-VALUE$(s, a, R, d)$
7   Return $R$
```

Algorithm 5: Generic MCTS approach.

```
    input: $H$, $\mathcal{M}$, $s_0$
    output: An action $a$
1   while remaning time $> 0$ do
        SEARCH$(s_0, 0, H, \mathcal{M})$
2   Return BEST-ACTION$(s_0$,0)
```

2002), which leads to the UCT algorithm (Kocsis and Szepesvári, 2006). The next step in a MCTS algorithm is to sample a successor for the state and action, using the generative model, and add the observed reward to the total cumulative reward (lines 4 asnd 5). Finally, function UPDATE-VALUE (Algorithm 4, line 6) adjusts the estimate of the Q-value for the given state-action pair; this will be explained in more detail below.

In order to compute Q-value estimates and select actions, MCTS algorithms keep counters of the number of times states and state-action pairs have been seen. Typically, this is done in a tree-structured way; that is, a different copy of a state is maintained for each possible path leading to that state, each with its own counters. We use the notation $N(s)$ to represent the number of times a node represented state s has been visited during the episodes, and $N(s, a)$ the number of times a node representing state-action pair (s, a) has been visited.

The UPDATE-VALUE function typically adjusts the Q-value estimates using Monte-Carlo averaging, through the following equation:

$$\hat{Q}^{(t+1)}(s,a) \leftarrow \hat{Q}^{(t)}(s,a) + \eta \frac{R - \hat{Q}^{(t)}(s,a)}{N(s,a)} \tag{2.13}$$

where η is a learning rate and R is the accumulated reward observed after performing a rollout starting at s, a. Additionally, Q-values can be initialized before performing this computation for the first time, in the same vein as done in EVALUATE. An useful property is that, unlike dynamic programming methods that select actions greedily, these estimate are not required to be admissible.

The performance of the algorithm depends a lot on the action selection procedure. As mentioned before, the most popular one is the UCB1 rule, which comes from the multi-armed bandit problem literature. This rule estimates an *upper confidence bound* for the Q-value of the actions, and selects the action with the highest bound. These bounds are computed using the equation

$$Q_{UCB1}(s,a) \leftarrow \hat{Q}^{(t)}(s,a) + C\sqrt{\frac{\ln N(s)}{N(s,a)}} \tag{2.14}$$

Kocsis and Szepesvári (2006) showed that the resulting MCTS approach converges to the optimal solution when the exploration constant, C, is chosen appropriately. Unfortunately, the choice of C greatly affects the performance of UCT. Nevertheless UCT was the basis for a successful planner for finite-horizon MDPs called PROST (Keller and Eyerich, 2012), which won the last three International Probabilitistic Planning Competitions, in 2011, 2014, and 2018. Moreover, MCTS is an active research area and it has recently been used in the creation of Alpha Go, the Go-playing program that beat the human champion Lee Sedol in 2016 (Silver et al., 2016).

On the other hand, without undermining the success of MCTS methods, we highlight the fact that these sampling methods ignore the declarative model when available, hence neglect useful information that could potentially improve performance. In fact, some have argued that the success of UCT in planning problems is mostly related to their ability to use non-admissible—but accurate—heuristics, which gives it an edge over previous dynamic programming methods (Bonet and Geffner, 2012). Part of the goal of this thesis is to devise better sampling mechanisms for problems in which access to a declarative model of the MDP is available.

2.2.5.3 Trial-based Heuristic Tree-Search

We conclude the section on sparse sampling approaches by briefly describing THTS, a recent framework that generalizes many trial-based solvers for finite-horizon MDPs (Keller and Helmert, 2013; Keller, 2015). The framework identifies the following components:

- **Initialization**: Initializes nodes in the tree with estimates for the values of states and actions.
- **Backup function**: Defines how value and Q-value estimates are propagated up the tree. Examples of this are the Bellman update equation 2.8 and the Monte-Carlo update equation 2.13.
- **Action Selection**: Defines how actions are selected to explore the tree (e.g., greedily or using UCB1).
- **Outcome Selection**: Defines how successors are selected for exploration (e.g., by sampling from the transition function).
- **Trial Length**: Determines if trials are stopped when a leaf node is reached or by some other mechanism (e.g., when a previously unseen note is visited for the first time).
- **Recommendation Function**: Returns an action given the current value estimates and statistics gathered during the trials.

The framework describes a generic THTS algorithm, following an outline very similar to the generic MCTS illustrated in Algorithm 5, except that the search alternates between visiting decision nodes (associated to states) and chance nodes (associated to actions). The interesting insight arising from the THTS work was showing that it is possible to produce new anytime optimal solvers[2] by combining dynamic programming backup updates with non-greedy action selection mechanisms. Moreover, they introduced several backup operators to illustrate this idea, for instance the following *partial Bellman backup* operator:

$$V_k(n_d) \leftarrow \begin{cases} 0 & \text{if } n_d \text{ represents a terminal state} \\ \max_{n_c \in \mathsf{succ}(n_d)} Q_k(n_c) & \text{otherwise} \end{cases} \tag{2.15}$$

$$Q_k(n_c) \leftarrow \mathcal{R}(\rho(n_c), n_c) + \frac{\sum_{n_d \in \mathsf{succ}(n_c)} \mathcal{T}\big(\rho(n_c), n_c, n_d\big) \cdot V_k(n_d)}{\sum_{n_d \in \mathsf{succ}(n_c)} \mathcal{T}\big(\rho(n_c), n_c, n_d\big)} \tag{2.16}$$

Here n_d and n_c represent a decision node and a chance node, respectively, whose associated state and actions can be recovered through functions $s(n_d)$ and $a(n_c)$; the parent of a chance node can be recovered through function $\rho(n_c)$. We abuse notation to let $\mathcal{R}$ and $\mathcal{T}$ operate over nodes directly, as if they were the states/actions they represent. Following this notation, it is straightforward to see that the partial Bellman backup converges to the usual Bellman backup as long as whole tree is eventually expanded—thus making $\sum_{n_d \in \mathsf{succ}(n_c)} \mathcal{T}\big(\rho(n_c), n_c, n_d\big) \rightarrow 1$. In the mean time, it performs backups in a manner similar to the operator used by SS (Algorithm 1, line 3), but incorporates information about the transition probabilities of the model. Combining this backup operator with the UCB1 action selection rule leads to the DP-UCT and UCT* algorithms (Keller and Helmert, 2013) (the latter being a version

[2]Anytime solvers are solvers that are optimal with infinite exploration, but can return a sub-optimal or partial plan if stopped prematurely.

of DP-UCT that stops trials whenever an unvisited node is expanded)—the UCT* algorithm formed the basis of PROST that won the 2014 IPPC.

Of particular interest to this thesis is the outcome selection component, which the THST authors recognize as an under-explored research topic. In their work they used the usual strategy of sampling from the transition distribution, but leave open the possibility of devising more informed mechanisms. In the next and final section of the literature review, we will describe the few existing algorithms (to the best of our knowledge) that have dealt with this topic.

2.2.6 Solvers with Alternative Outcome Selection

With the exception of determinization-based approaches, all algorithms discussed so far explore the state space by sampling from the (possibly unknown) transition function of the underlying MDP. However, when the transition function is given, it makes sense to instead sample states that are more *informative*, and use the known probability of transition to incorporate the result into the value estimate. The algorithms described in in this section attempt to do precisely this.

The first of these algorithms is Bounded RTDP (BRTDP) (McMahan et al., 2005), an extension of the RTDP algorithm (Barto et al., 1995) that incorporates upper and lower bounds on the state values. Instead of following the probabilities implied by transition function, BRTDP biases sampling towards states where the gap between the bounds is large. Specifically, given a state s and action a, BRTDP constructs a new transition function, $\mathcal{T}'(s, a, s') \propto \mathcal{T}(s, a, s') \cdot \mathsf{gap}(s')$, where $\mathsf{gap}(s')$ is the difference between the upper and lower bound on the value of s'. The authors show that this simple approach improves performance over RTDP and LRTDP.

Another algorithm that modifies the transition function is Focused RTDP (FRTDP) (Smith and Simmons, 2006). FRTDP tries to exploit the concept of *occupancy* of states in MDPs. The occupancy of a state, denoted as $W(s)$, is roughly defined as the expected number of steps per execution that a policy spends on s before it reaches a fringe state (one that has not been explored by the algorithm yet). The main observation is that the policy quality is directly related to its quality at fringe states. Therefore, when bounds on state values are available, it makes sense to explore the fringe state that can decrease the most the gap in the value of the initial state s_0—which represents the policy's quality. As it turns out, this is the fringe state s' with the largest value of $W(s') \cdot \mathsf{gap}(s')$. However, because $W(s')$ cannot be computed exactly, FRTDP maintains a priority value for each state, which approximates this quantity, and selects outcomes greedily according to this priority. This approach also shows improvement when compared to LRTDP.

Finally, the most recent algorithm following this trend is VPI-RTDP (Sanner et al., 2009). The idea of this approach is that, rather than decreasing the variance in our value estimates, the agent should attempt to directly improve policy quality. To do this, VPI-RTDP performs a *myopic Value of Perfect Imformation* analysis (Howard, 1966; Dearden et al., 1998) to estimate what the improvement in policy quality would be if we have perfect knowledge about each successor's value. The algorithm then samples from the distribution implied by the VPI scores for the successor states. The authors report significant improvements over BRTDP and FRTDP.

CHAPTER 3

$\mathcal{M}_l^k$-REDUCTIONS - GENERALIZING DETERMINIZATION

In this chapter we introduce the $\mathcal{M}_l^k$-reduction, a new form of reduction for MDPs that generalizes single-outcome determinization as just one extreme point on a spectrum of MDP reductions that differ from each other along two dimensions: i) the number of outcomes per state-action pair that are fully accounted for, and ii) the number of occurrences of the remaining, exceptional, outcomes that are planned for in advance. An interesting insight obtained from this thesis is that the choice of reduction is crucial for achieving good performance. We show experimental results that highlight the benefit of planning with reduced models and the effects of the reduction choice in the performance of the resulting plans.

3.1 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. 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.

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. (3.1), (3.2) and (3.3) 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 $s_0' \triangleq \langle s_0, 0 \rangle$;
- 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:

$$\forall s, a, s' \quad \mathcal{T}'(\langle s, j\rangle, a, \langle s', j'\rangle) \triangleq \begin{cases} p'_i & e^a_i \in \mathcal{P}_a \ \wedge \ j' = j = 0 \\ 0 & e^a_i \notin \mathcal{P}_a \ \wedge \ j' = j = 0 \end{cases} \tag{3.1}$$

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

$$\forall i : e^a_i \in \mathcal{P}_a \ \Rightarrow \ p'_i > 0 \ \wedge \sum_{i : e^a_i \in \mathcal{P}_a} p'_i = 1 \tag{3.2}$$

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:

$$\forall s, a, s', j, j' \quad \mathcal{T}'(\langle s, j\rangle, a, \langle s', j'\rangle) \triangleq \begin{cases} p^a_i & e^a_i \in \mathcal{P}_a \ \wedge \ j' = j \\ p^a_i & e^a_i \notin \mathcal{P}_a \ \wedge \ j' = j - 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

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* and LRTDP, 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 3.1 illustrates the pruning effect that can be achieved with a reduced model.

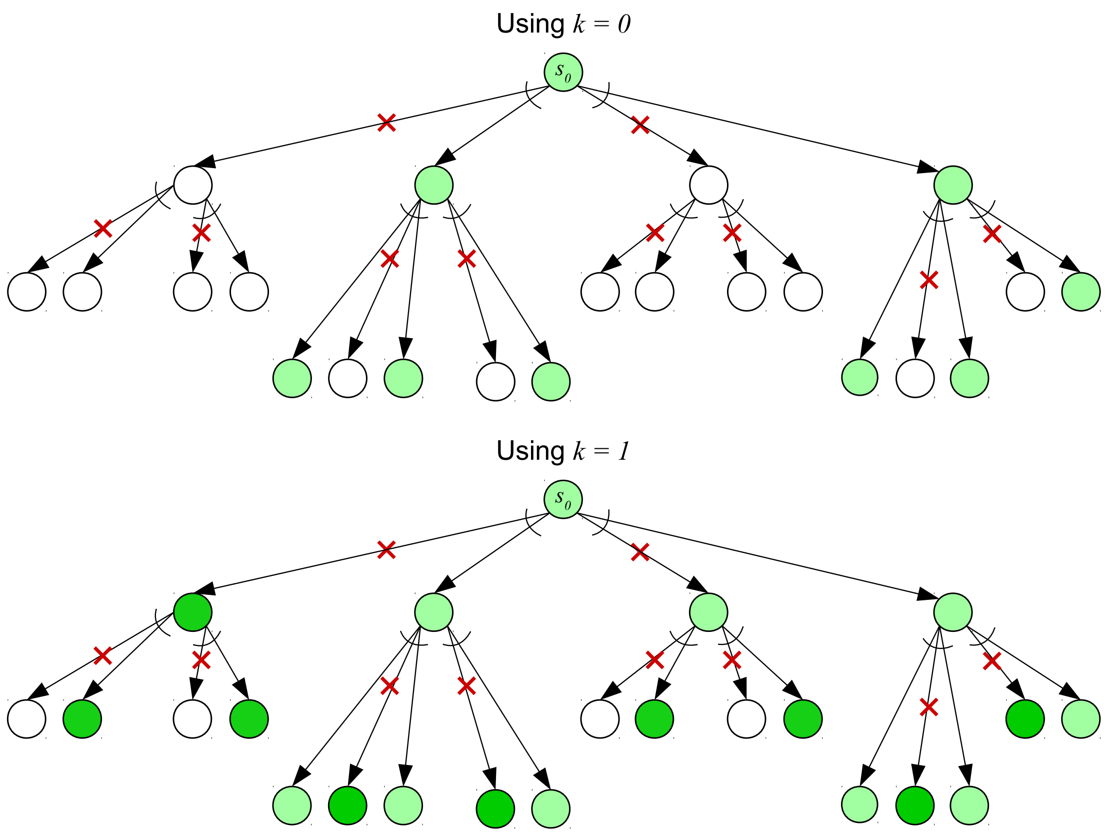

Figure 3.1: Illustration of the pruning effect of an $\mathcal{M}_l^k$-reduction, using two different values of k. Exceptional outcomes are marked with a red cross and reachable states are highlighted in green (darker color for those reachable with $k = 1$ but not $k = 0$). The value of k can be used to regulate the trade-off between computational efficiency and plan robustness.

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

Definition 8 ($\boldsymbol{\mathcal{M}_l^k}$-reduction of an MDP). *An $\mathcal{M}_l^k$-reduction of an MDP is an augmented MDP with the transition function defined by Eqs. (3.1), (3.2), and (3.3), 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 work (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 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. (3.1). In the thesis 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.

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 or $\mathcal{M}_1^0$-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.

3.2 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 is not included in the plan. Hence, in this section, 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.

3.2.1 Continual Planning Using Reduced Models

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 (desJardins et al., 1999; Myers, 1999; Chien et al., 2000; Brenner and Nebel, 2009). 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 6. We use the notation $\mathcal{P}$ to represent the choice of primary outcomes for a reduction; that is, a mapping[1] $\mathcal{P} : \mathcal{A} \rightarrow 2^{\{e_1^a,...,e_m^a\}}$, 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. 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

[1]This is a slight abuse notation, since the set of possible outcomes $\{e_1^a, ..., e_m^a\}$ is indexed by action a. Nevertheless, since the intended meaning should be clear, we argue that the gain in readability compensates for the loss of rigor.

Algorithm 6: $\mathcal{M}_l^k$-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.
            $\forall s' \in \mathcal{S}: Pr(\langle s', k \rangle | \hat{s}, \hat{a}) = \mathcal{T}(s' | s, \pi \langle s, j \rangle)$
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         $\langle s, j \rangle \leftarrow (s, k)$
```

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.

3.2.2 Evaluating the Performance of the Continual Planning Approach

Unlike existing continual planning methods (Chanel et al., 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 6.

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^M_{cp} denote the value function defined over this *continual planning Markov chain* with respect to a given reduction M. Then we have:

Proposition 1. $V^M_{cp}(\langle s_0, k\rangle)$ *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.

3.3 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 3.3.1, 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 3.3.2, 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.

3.3.1 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

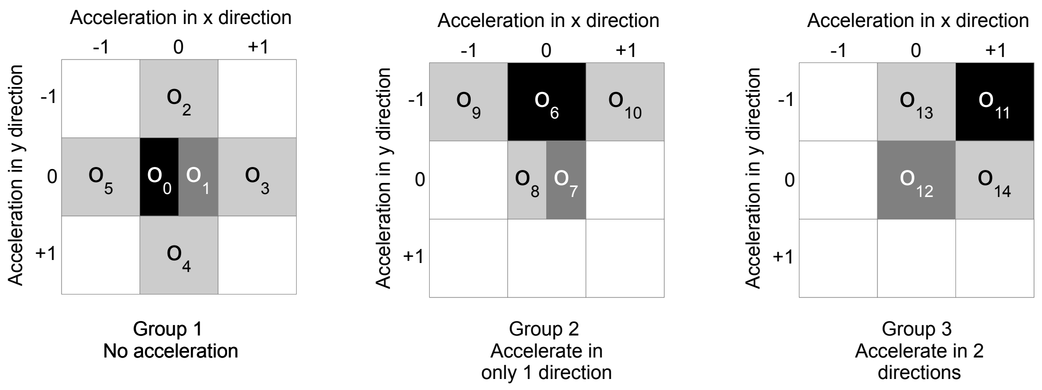
Acceleration in x direction
-1
0
+1
Acceleration in y direction
-1
0
+1
O2
O5
O0
O1
O3
O4
Group 1
No acceleration
O9
O6
O10
O8
O7
Group 2
Accelerate in only 1 direction
O13
O11
O12
O14
Group 3
Accelerate in 2 directions

Figure 3.2: 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.

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.

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 3.2 illustrates these groups of actions and their possible outcomes; for each group, a decomposition is specified by the set of outcomes, relative

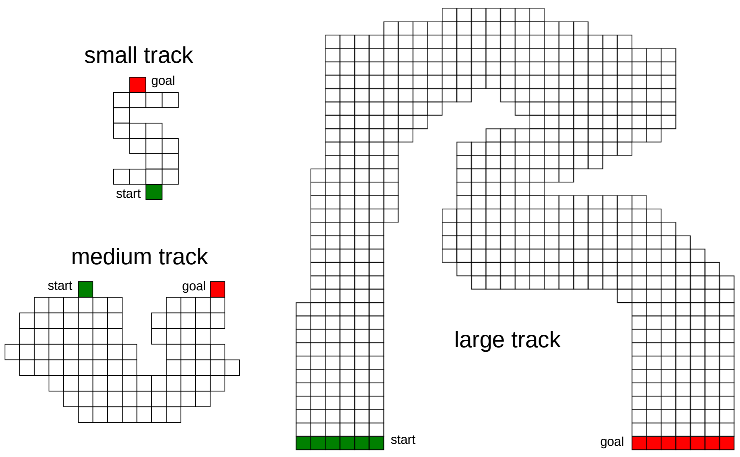

Figure 3.3: Three instances of the racetrack domain.

	$\|\mathcal{S}\|$	$V_{cp}^{M_1}\langle s_0, 0\rangle$	$V_{cp}^{M_2}\langle s_0, 0\rangle$	$\hat{V}^{ao}(s_0)$
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 3.1: Comparison of the best determinization (M_1) and the best $\mathcal{M}_2^0$-reduction (M_2) for three racetrack problems.

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 3.3).

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 3.1 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 con-

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

sidered, 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 5% 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 probability outcomes), but the goal of the previous analysis is to highlight the importance of the choice of reduced model, *ceteris paribus*.

3.3.2 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.

To illustrate this issue we experimented with different determinizations of the TRIANGLE-TIREWORLD domain (Little and Thiebaux, 2007). This problem involves a car traveling between locations on a graph shaped like a triangle (see Figure 3.4).

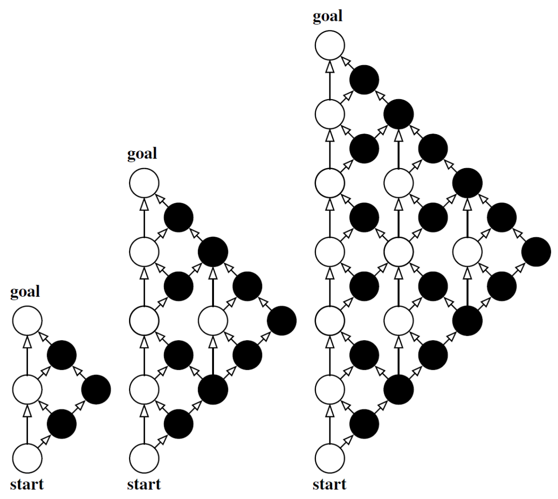

Figure 3.4: Three instances of the TRIANGLE-TIREWORLD domain. Locations with spare tires are marked in black (Little and Thiebaux, 2007).

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

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

which guarantees the convergence of heuristic search algorithms (Kolobov et al., 2012).

This domain has two possible determinizations, depending on whether getting a flat tire is considered an exception or a primary outcome. Table 3.2 shows the results (number of trials reaching the goal) of evaluating the two determinizations on 10 instances of this domain. 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).

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 show (Chapter 4) .

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.

3.4 A Greedy Approach for Learning Reduced Models

In this section, we propose 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 the 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 5.3 confirm that it can work well in practice.

Given k and l, every reduction $M \in \mathcal{M}_l^k$ is uniquely determined by the mapping $\mathcal{P}^M : A \to 2^{\{e_1^a,\dots,e_m^a\}}$, 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:

$$\begin{aligned} \max_{\mathcal{E}^M \subseteq \mathcal{E}} \quad & -V_{cp}^M(\langle s_0, k\rangle), \quad \mathcal{E} \equiv \bigcup_{a \in \mathcal{A}} \mathit{outcomes}(a) \\ \text{s.t.} \quad & \forall a \in \mathcal{A},\ 1 \leq |\{e : e \in \mathcal{E}^M \cap \mathit{outcomes}(a)\}| \leq l \end{aligned} \tag{3.4}$$

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 et al., 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 and 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. (3.4) is not submoduar, as the following proposition shows.

Proposition 2. *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 3.5. 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. $\square$

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. (3.4).

Our method, described in Algorithm 7, 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: i) the maximum number of primary outcomes is larger than the desired l (lines 19 and 21), and ii) the relative increase in comprehensive cost with respect to the value of the full model is lower than some threshold (line 21).

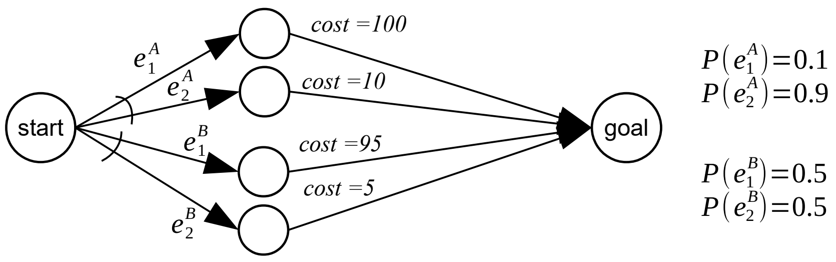

Figure 3.5: Example showing that $-V_{cp}^{M}(\langle s_0, k\rangle)$ is not submodular. Actions A and B have cost 1.

We use VALUE-ITERATION to compute $V_{cp}^{\hat{M}}(\langle s_0, k\rangle)$, as doing so requires a universal plan (see Section 3.2.2). 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

Algorithm 7: 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 $M$
1  $\mathcal{E}^M \leftarrow \bigcup_{a \in \mathcal{A}} outcomes(a)$
2  $M \leftarrow \mathcal{M}_l^k$-reduction of $\mathbb{M}$ with primary outcomes $\mathcal{E}^M$
3  $V_{opt} \leftarrow V_{cp}^M(\langle s_0, k \rangle)$
4  while true do
5      $V_{best} \leftarrow \infty$
6      $\alpha \leftarrow \emptyset$
7      $e_{best}^{\alpha} \leftarrow \emptyset$
8      for $a \in \mathcal{A}$ do
9          for $e^a \in (outcomes(a) \cap \mathcal{E}^M)$ do
10             $\hat{\mathcal{E}} \leftarrow \mathcal{E}^M \setminus \{e\}$
11             $\hat{M} \leftarrow$ CREATE-REDUCED-MDP$(\mathbb{M}, s_0, \hat{\mathcal{E}}, k)$
12             if SOLVABLE$(\hat{M}) \wedge V_{cp}^{\hat{M}}(\langle s_0, k \rangle) < V_{best}$ then
13                 $V_{best} \leftarrow V_{cp}^{\hat{M}}(\langle s_0, k \rangle)$
14                 $\alpha \leftarrow a$
15                 $e_{best}^{\alpha} \leftarrow e^a$
16     if $V_{best} = \infty$ then
17         break // Removing any outcome makes problem unsolvable
18     $\mathcal{E}^M \leftarrow \mathcal{E}^M \setminus \{e_{best}^{\alpha}\}$
19     $l_{max} \leftarrow \max_a |outcomes(a) \cap \mathcal{E}^M|$
20     $M \leftarrow$ CREATE-REDUCED-MDP$(\mathbb{M}, s_0, \mathcal{E}^M, k)$
21     if $\left(\frac{V_{best} - V_{opt}}{V_{opt}}\right) > \tau \wedge l_{max} \leq l$ then
22         break
```

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 that can be applied to other instances in the same 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.

3.5 Experimental Results

In this section we present experiments for evaluating the performance of $\mathcal{M}_l^k$-REPLAN, as well as the effectiveness of our strategy for learning reduced models. Additionally, in Section 3.5.1, we introduce an anytime version of $\mathcal{M}_l^k$-REPLAN, to evaluate how our approach performs under time constraints.

3.5.1 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. For the racetrack problem, we used $p_{slip} = 0.1$ and $p_{er} = 0.05$ (see description in Section 3.3.1). 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 MKL 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.

We learned the two sets of primary outcomes using GREEDY-LEARN (Algorithm 7), 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, o5, and o11 in Figure 3.2), and the $\mathcal{M}_2^k$-reduction added to that the possibility of slipping when accelerating in one direction (outcomes o0, o5, o11, and o8 in Figure 3.2). The racetrack used in our experiments is shown in Figure 3.6, which has 34,897 states.

Table 3.3 (bottom) shows the total CPU time spent on planning (bottom), which includes the time used to compute an initial plan, as well as the time needed for re-

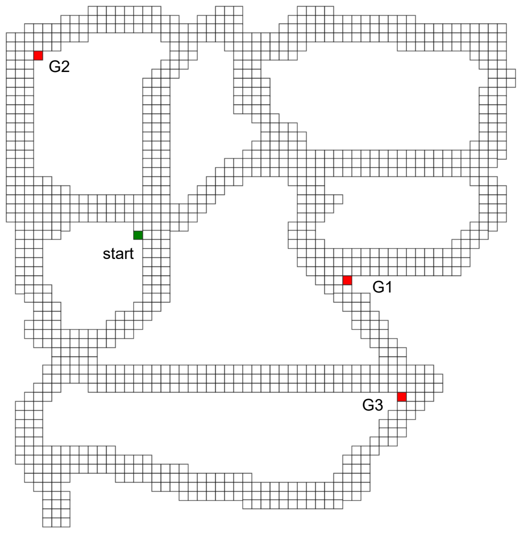

Figure 3.6: An instance of the racetrack domain.

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 on Table 3.3 (top), which are computed exactly using the Markov Chain discussed in Section 3.2.2. Note that planning with the most-likely-outcome determinization (M01), 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%). Adding a single outcome, without increasing k (M02), decreases the expected cost by at least 5% in two of the problems considered, with marginal increase in total planning time. Notice, however, that M02 resulted in a slight increase in expected cost for problem G3, which indicates that simply adding

	Expected Cost								
	LAO*	M01	M11	M21	M31	M02	M12	M22	M32
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*	M01	M11	M21	M31	M02	M12	M22	M32
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 3.3: Expected cost and average planning time obtained with several reduced models of the racetrack domain.

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 (M31).

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 evaluate an anytime variant of $\mathcal{M}_l^k$-REPLAN, referred to as $\mathcal{M}_l^k$-ANYTIME, and outlined in Algorithm 8. 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),

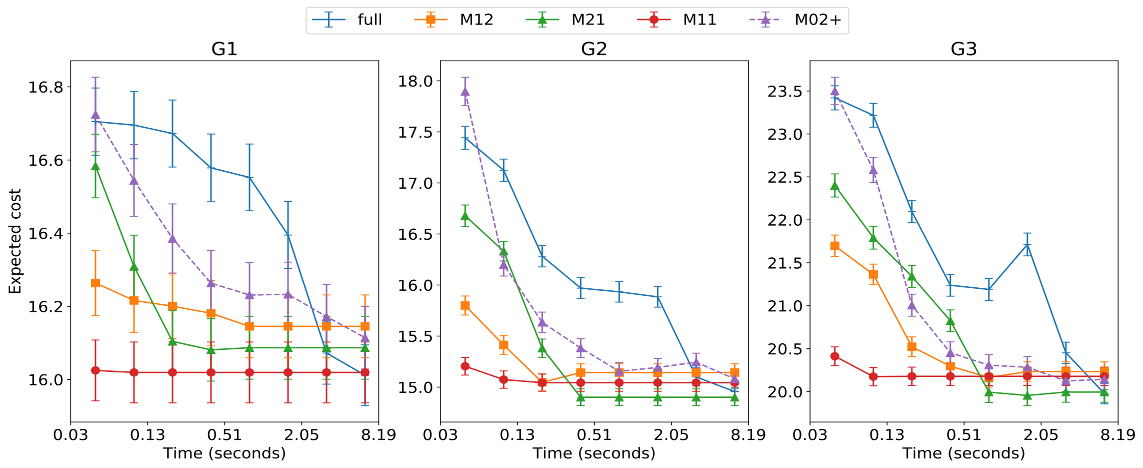

Figure 3.7: Anytime performance of several $\mathcal{M}_l^k$-reductions using $\mathcal{M}_l^k$-ANYTIME to solve three instances of the racetrack problem. From left to right, results corresponding to G1, G2, and G3 in Figure 3.6.

and the agent always chooses an action greedily 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 always uses the greedy action corresponding to $\langle s, k \rangle$.

We evaluated $\mathcal{M}_l^k$-ANYTIME using the same set of racetrack problems illustrated in Figure 3.6, 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 and Geffner, 2003a) as the underlying optimal planner, since it has better anytime properties than LAO*. The results for the best performing reductions are shown in Figure 3.7, 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.

As seen in Figure 3.7, in the large majority of scenarios, the best anytime performance was obtained using $k = 1$, both with one and two primary outcomes (planners

M11 and M12, respectively). Planner M11, in particular, significantly outperformed all other planners for all times lower or equal to 200 milliseconds per action. Planners M12 and M21 were able to perform better when the time increased, and for times of 800 milliseconds per action, or higher, the performance of M11, M12, and M21 was comparable (M21 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 3.7 does not include the results for M01, and M02. We did not include these results to maintain clarity in the figure, but note that the observed performance matched that shown in Table 3.3 (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 8), 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 8), 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 $M02+$. While the results are not better than those of the other reduced models considered,

Algorithm 8: $\mathcal{M}_l^k$-REPLAN-ANYTIME

```
   input: $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{C}, s_0, \mathcal{G} \rangle$, $k$, $\mathcal{P}$, $\tau$
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      $a \leftarrow$ GREEDY-ACTION$(M, \langle s, k \rangle)$
5      Create new state $\hat{s}$ with one zero-cost action $\hat{a}$ s.t.
         $\forall s' \in \mathcal{S}\colon Pr(\langle s', k \rangle | \hat{s}, \hat{a}) = \mathcal{T}(s' | s, a)$
6      $M \leftarrow$ CREATE-REDUCED-MDP$(\mathcal{M}, \hat{s}, \mathcal{P}, k)$
       do in parallel
7          PLAN-UNTIL-PREEMPTED$(M, \langle s, k \rangle)$
8          $s \leftarrow$ EXECUTE-ACTION$(s, a)$
```

the approach has good anytime performance, and may be a good choice when it is not clear what a good value for k is.

CHAPTER 4

COMBINING $\mathcal{M}_l^k$-REDUCTIONS WITH CLASSICAL PLANNING TECHNIQUES

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 chapter we present one such algorithm, specifically tailored to $\mathcal{M}_1^k$-reductions. The approach employs a classical planner to accelerate computation for states in which no more exceptions will be considered, i.e., states $\langle s, j \rangle$ such that $j = 0$. We also introduce an approach for learning determinizations over the space of factored domain language representations.

4.1 FF-LAO*: Leveraging classical planners

Using determinization has the advantage of making 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. This solver, FF-LAO* (Algorithms 9-12), 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*[1], 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 9, where the state expansion and test convergence procedures are replaced with versions that use FF (Algorithms 10 and 11, 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. 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 call to FF is done in Algorithm 12 (FF-BELLMAN-UPDATE). This procedure performs a Bellman update, as in Eq. (2.8), 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[2], 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 com-

[1]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.

[2]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 9: FF-LAO*

```
   input: $M = \langle S', A, T', C', \langle s_0, k\rangle, G'\rangle$, $k$, $\epsilon$
1  while true do
       // Node expansion step
2      while true do
3          $visited \leftarrow \emptyset$
4          $cnt \leftarrow$ FF-EXPAND$(M, \langle s, j\rangle, k, visited)$
5          if $cnt = 0$ then
               // No tip nodes were expanded, so current policy is closed
               break
       // Convergence test step
6      while true do
7          $visited \leftarrow \emptyset$
8          $error \leftarrow$ FF-TEST-CONVERGENCE$(M, \langle s, j\rangle, k, visited)$
9          if $error < \epsilon$ then
               return // solution found
10         if $error = \infty$ then
               break // change in partial policy, go back to expansion step
```

puted 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 $\pi[\langle s, 0\rangle] =$ 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.

4.1.1 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 3.2; this new approach is illustrated in Algorithm 13. The idea is simple: during execution, check if the current state has

Algorithm 10: FF-EXPAND

```
   input: $M = \langle S', A, T', C', \langle s_0, 0\rangle, G'\rangle$, $\langle s, j\rangle$, $k$, $visited$
1  if $\langle s, j\rangle \in visited$ then
       return 0
2  $visited \leftarrow visited \cup \{\langle s, j\rangle\}$
3  $cnt = 0$
4  if $\pi[\langle s, j\rangle] = \emptyset$ then
       // Expand this state for the first time
5      FF-BELLMAN-UPDATE$\big(M, \langle s, j\rangle, k\big)$
6      return 1
7  else if $j > 0$ then
8      forall $\langle s', j'\rangle$ s.t. $T'(\langle s', j'\rangle | \langle s, j\rangle, \pi[\langle s, j\rangle]) > 0$ do
9          $cnt$ += FF-EXPAND$\big(M, \langle s, j\rangle, k, visited\big)$
10 FF-BELLMAN-UPDATE$\big(M, \langle s', j'\rangle, k\big)$
11 return $cnt$
```

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).

4.1.2 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 3. *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 12, which essentially induces a new MDP in which the states

Algorithm 11: FF-TEST-CONVERGENCE

```
   input: $M = \langle S', A, T', C', \langle s_0, k\rangle, G'\rangle$, $\langle s, j\rangle$, $k$, visited
1  if $s \in visited$ then
       return 0
2  $visited \leftarrow visited \cup \{\langle s, j\rangle\}$
3  $error = 0$
4  $a \leftarrow \pi[\langle s, j\rangle]$
5  if $a = \emptyset$ then
       // The test reached a state that has not been expanded yet
6      return $\infty$
7  else if $j > 0$ then
8      forall $\langle s', j'\rangle$ s.t. $T'(\langle s', j'\rangle | \langle s, j\rangle, \pi[\langle s, j\rangle]) > 0$ do
9          $error = \max\big(error$, FF-TEST-CONVERGENCE$(M, \langle s, j\rangle, k, visited)\big)$
10 $error = \max\big(error$, FF-BELLMAN-UPDATE$(M, \langle s, j\rangle, k)\big)$
11 if $\pi[\langle s, j\rangle] \neq a$ then
12     return $\infty$ // the policy changed
13 return error
```

$s_i, s_{i+1}, ..., s_L$ 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 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 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. □

Algorithm 12: FF-BELLMAN-UPDATE

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

Algorithm 13: FF-LAO*-REPLAN

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

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 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.

4.1.3 Learning a Good Determinization

In this section we present an approach for learning a good single-outcome determinization, although its main idea can also 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 3.3.2, one of such domains is the TRIANGLE-TIREWORLD problem (Little and Thiebaux, 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 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 14 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 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

Algorithm 14: LEARNING-DET

```
input: $\mathcal{D}, \mathbb{M}_l = \langle S, A, T, C, s_0, G \rangle, k$
output: $\mathcal{P}$
1 $\{\mathcal{P}_1, ..., \mathcal{P}_\mu\} \leftarrow$ Create all possible determinizations of $\mathcal{D}$
2 forall $i \in \{1, ..., \mu\}$ do
3     $M \leftarrow$ CREATE-REDUCED-MDP$(\mathbb{M}_l, s_0, \mathcal{P}_i, k)$
4     Estimate probability of successs and expected cost of a continual planning
        approach with input $M$, $k$
5 $P^* \leftarrow \max_i P_i$
6 $\mathcal{P} \leftarrow \mathcal{P}_{\min_i \mathbb{C}_i \ s.t. \ P_i = P^*}$
```

we described in Section 3.3.2, 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 et al., 2016).

4.2 Experiments

4.2.1 Domains and methodology

We evaluated FF-LAO* and LEARNING-DET on a set of problems taken from IPPC'08 (Bryce and 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 and Geffner, 2005). Additionally, we modified the EX-BLOCKSWORLD domain to avoid the possibility of blocks to be put on top of themselves (Trevizan and Veloso, 2014).

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: i) connect to the MDPSIM server to receive a state, ii) compute an action for the received state and send the action to the MDPSIM server, and iii) 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 and 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 and 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 and Geffner, 2005).

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 minutes 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.

4.2.2 Results and Discussion

Figure 4.1 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 deter-

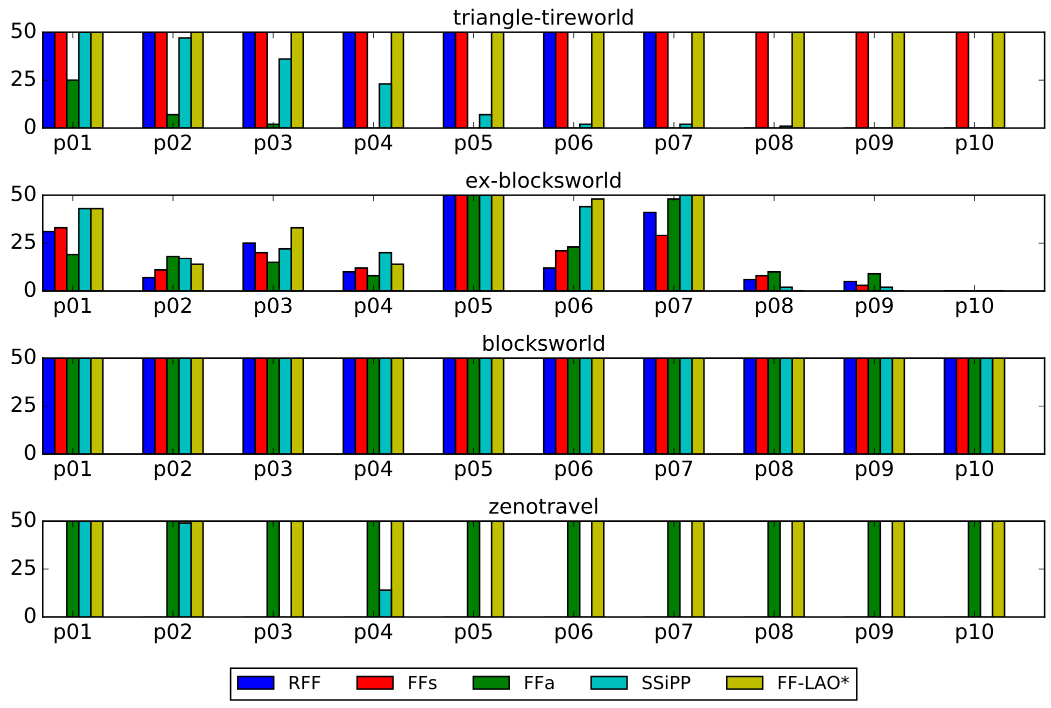

Figure 4.1: Number of solved rounds by 5 different planners in IPPC'08 benchmarks.

minization 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 here for space considerations, the performance with $k = 1$ (214 successful rounds) was already better than all 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 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 briefly mention another important state-of-the-art planner that we could not include in our experiments. FF-H^+ is a planner based on hindisight optimization that has been shown to outperform RFF in the IPPC benchmarks (Yoon et al., 2010). Unfortunately, we were not able to obtain code for this planner even after repeated email communication with two of the original authors; thus, our results are not directly comparable, and should be taken merely as suggestive of what a proper comparison would show.

With that consideration, we note that 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 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 authors of FF-H$^+$ report planning times 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. As mentioned before, this comparison should be taken with care, but they are still suggestive of the power of planning with determinizations that are automatically tailored to the specific characteristics of a domain.[3].

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

CHAPTER 5

A NEW LABELING MECHANISM FOR EFFICIENT STATE EXPLORATION

In this chapter we introduce an alternative approach to reduce the computational effort of search algorithms, not by modifying the transition model, but by directly changing the algorithm sampling's mechanisms through a short-sighted notion of *labeling*. The resulting algorithm, FLARES, achieves near-optimal performance with very low computational effort. Subsequently, we introduce the soft labeling framework, a generalization of state labeling that bridges the gap between FLARES and the optimal solver LRTDP. The notion of soft labeling allows us to offer improved theoretical properties for short-sighted labeling algorithms. It also results in a mechanism for biasing state-space exploration toward states for which computation is more likely to improve policy quality. Our experimental result show that both short-sighted labeling and soft labeling result in state-of-the-art performance in challenging SSP benchmarks used by the planning community.

5.1 An MDP Solver Based on Short-Sighted Labeling

5.1.1 The FLARES Algorithm

Heuristic search algorithms are some of the best methods for solving SSPs optimally, note-worthy examples being LAO* and LRTDP. These algorithms are characterized by the use of an initial estimate of the optimal value function (referred to as a *heuristic*, denoted h) to guide the search to the more relevant parts of the state space.

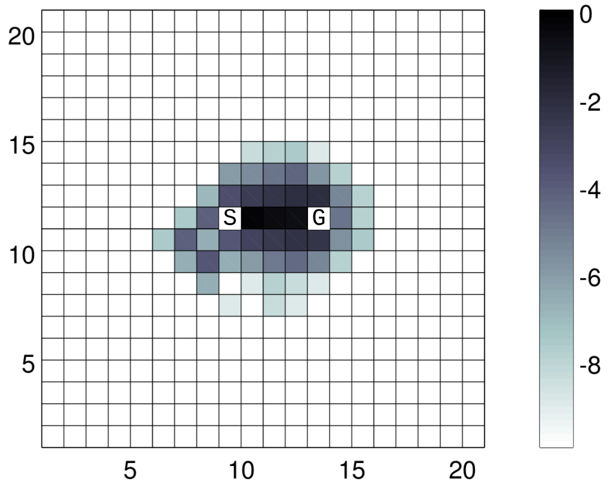

Figure 5.1: Problem with large optimal policy but small high-probability envelope (S: start state, G: goal, scale shows log-probability).

Typically the heuristic is required to be *admissible*, i.e., a lower bound on the optimal value function. Moreover, often the heuristic is required to be *monotone*, satisfying:

$$h(s) \leq \min_{a \in A} \left\{ C(s,a) + \sum_{s' \in S} T(s'|s,a) h(s') \right\} \tag{5.1}$$

Although heuristic search can result in significant computational savings over Value Iteration and Policy Iteration, their efficiency is highly correlated with the size of the resulting optimal policy. Concretely, in order to confirm that a policy is optimal, a solver needs to ensure that there is no better action for *any* of the states that can be reached by this policy. Typically, this involves performing one or more Bellman backups on all reachable states of the current policy, until some convergence criterion is met.

However, it is common to have an optimal policy in which many of the covered states can only be reached with very low probability. Thus, their costs have minimal impact on the expected cost of the optimal policy. For instance, consider the grid shown in Figure 5.1. Suppose that every time the agent tries to move in one direction, it succeeds with probability 0.7, or moves in each of the other directions with probability 0.1; the goal is to move from position (9,11) to position (13,11). Due to the nature of the transition function, the optimal policy for this problem covers the entire state space. Yet, as the color gradient shows, the (log)probability of visiting a state under the optimal policy quickly degrades with distance to the goal, resulting in a very small "envelope" of high-probability states, close to the most likely path between the start and the goal. This raises the question of how to better exploit this property to design faster approximate algorithms for SSPs.

We present the FLARES algorithm that leverages this property by combining short-sightedness and trial-based search in a novel way. Concretely, FLARES works by performing a number of trials from the start to the goal, while trying to label states as solved according to a short-sighted labeling criterion. The key property of FLARES, which distinguishes it from other short-sighted approaches, is that it can propagate information from the goal to the start state while simultaneously pruning the state-space, and do so without requiring a large search horizon. Intuitively, FLARES works by attempting to construct narrow corridors of states with low residual error from the start to the goal.

Readers familiar with heuristic search methods for solving MDPs will notice similarities between FLARES and the well-known LRTDP algorithm (Bonet and Geffner, 2003a). Indeed, FLARES is based on LRTDP with a particular change in the way states are labeled. For reference, LRTDP is an extension of RTDP that includes a procedure to label states as solved (CHECKSOLVED). In RTDP, trials are run repeatedly and Bellman backups are done on each of the states visited during a trial. This procedure can be stopped once the current greedy policy covers only ϵ-consistent states. In LRTDP, this is improved by pushing to a stack the states seen during a trial, and then calling CHECKSOLVED on each as they are taken out of the stack.

Algorithm 15: A depth limited procedure to label states.

```
DLCHECKSOLVED
    input : $s, t$
1   $solved = true$
2   $open = $ EMPTYSTACK
3   $closed = $ EMPTYSTACK
4   $all = true$
5   if $\neg(s.\text{SOLV} \vee s.\text{D-SOLV})$ then
6       $open.\text{PUSH}(\langle s, 0\rangle)$
7   while $open \neq$ EMPTYSTACK do
8       $\langle s, d\rangle = open.\text{POP}()$
9       if $d > 2t$ then
10          $all = false$
11          continue
12      $closed.\text{PUSH}(\langle s, d\rangle)$
13      if $s.\text{RESIDUAL}() > \epsilon$ then
14          $solved = false$
15      $a = \text{GREEDYACTION}(s)$
16      for $s' \in \{s' \in S | P(s'|s, a) > 0\}$ do
17          if $\neg(s'.\text{SOLV} \vee s'.\text{D-SOLV}) \wedge s' \notin closed$ then
18              $open.\text{PUSH}(\langle s', d + 1\rangle)$
19          else if $s'.\text{D-SOLV} \wedge \neg s'.\text{SOLV}$ then
20              $all = false$
21  if $solved$ then
22      for $\langle s', d\rangle \in closed$ do
23          if $all$ then
24              $s'.\text{SOLV} = true$
25              $s'.\text{D-SOLV} = true$
26          else if $d \leq t$ then
27              $s'.\text{D-SOLV} = true$
28  else
29      while $closed \neq$ EMPTYSTACK do
30          $\langle s', d\rangle = closed.\text{POP}()$
31          BELLMANUPDATE$(s)$
32  return $solved$
```

The CHECKSOLVED labeling procedure has the following property: it only labels a state s as solved if all states s' that can be reached from s following a greedy policy are ϵ-consistent. The main advantage of labeling is that, once a state is labeled as solved, the stored values and actions can be used if this state is found during future trials or calls to CHECKSOLVED.

While such a labeling approach could result in large computational savings, clearly CHECKSOLVED suffers from the same problem that affects optimal solvers—it may have to explore large low-probability sections of the state space because it must check *all* reachable states before labeling. To address this problem, we introduce the following depth-limited labeling property as a way to accelerate heuristic search methods: *a state s is considered* ***depth-t-solved*** *only if all states s' that can be reached with t or less actions following the greedy policy are ϵ-consistent.*

Algorithm 15 shows the procedure DLCHECKSOLVED that implements this idea: a call with state s and horizon t visits all states that can be reached from s by following at most $2t$ actions under the current greedy policy. If all states s' visited during this search satisfy $Res^V(s') < \epsilon$—where V is the current value funcion—the method then proceeds to label as depth-t-solved only those states found up to horizon t. Note that doing the search up to horizon $2t$ allows DLCHECKSOLVED to label several states during a single call, instead of only the root state if the residuals were only checked up to depth t.

The FLARES algorithm incorporates DLCHECKSOLVED into a trial based action selection mechanism (shown in Algorithm 16). Propositions 4 and 5 show the conditions under which FLARES, and more specifically DLCHECKSOLVED, maintains the labeling properties described above.

Proposition 4. *DLCHECKSOLVED labels a state s with $s.\text{SOLV} = true$ only if all states s' that can be reached from s following the greedy policy satisfy $Res^V(s') < \epsilon$, where $V : \mathcal{S} \rightarrow \mathbb{R}$ is the current value function.*

Proof. We prove this by contradiction. If a state x is labeled $x.\text{SOLV} = true$ incorrectly, then two things happen: i) $all = true$ at line 23, ii) there exists a descendant y in the greedy graph s.t. $Res^V(y) > \epsilon$ and $y \notin closed$. However, this implies some ancestor $u \neq x$ of y in the graph satisfies $\neg u.\text{SOLV} \wedge u.\text{D-SOLV}$ (line 18), which implies $all = false$ (line 20). $\square$

Algorithm 16: The FLARES algorithm.

```
FLARES
    input  : $s_0, t$
    output: action to execute
1   while $\neg s_0$.SOLVED $\vee$ $s_0$.D-SOLV do
2       $s = s_0$
3       $visited$ = EMPTYSTACK
4       while $\neg(s$.SOLVED $\vee$ $s$.D-SOLV$)$ do
5           $visited$.PUSH($s$)
6           if GOAL($s$) then break
7           BELLMANUPDATE($s$)
8           $a$ = GREEDYACTION($s$)
9           $s$ = RANDOMSUCCESSOR($s, a$)
10      while $visited \neq$ EMPTYSTACK do
11          $s$ = $visited$.POP()
12          if $\neg$DLCHECKSOLVED($s, t$) then
13              break
14      return GREEDYACTION($s$)
```

Proposition 5. *If, during the execution of FLARES, no call to* BELLMANUPDATE(s') *with* $Res^V(s') < \epsilon$ *results in* $Res^{V'}(s') \geq \epsilon$*, where V and V' are the value functions before and after the call, respectively, then DLCHECKSOLVED labels a state s with* s.D-SOLV *only if s is depth-t-solved.*

Proof. Proof by induction. For the induction step, note that calling DLCHECKSOLVED on state x with all previous labels being correct, results in new labels set correctly in line 27; this is because all the unlabeled descendants of x reachable within $2t$ steps will still be added to *closed*, but only those reachable within t steps are labeled. The base case, when no states have been previously labeled, is trivial, because in this case *all* descendants up to depth $2t$ are added to *open* (line 18). □

The assumption of Proposition 5 requires some explanation. State s can be labeled with s.D-SOLV while some of its low residual descendants with depth larger than t are not (DLCHECKSOLVED only labels states up to depth t after checking the residual on all states up to depth $2t$). Since FLARES can perform Bellman backups of unlabeled states, and because residuals are not guaranteed to be monotonically decreasing, it is possible for the residual of an unlabeled state to increment above ϵ during a trial, breaking the depth-limited labeling guarantee of its ancestors. This can lead to a sequence of events such as the following:

1. for some state, the algorithm correctly sets y.D-SOLV,
2. some unlabeled descendant z, reachable at depth d s.t. $t < d \leq 2t$, stops being ϵ-consistent, which means y is no longer depth-t-solved,
3. another state x s.t. y is a descendant at depth less than t, becomes labeled.

Note that when event 3 happens, state y is not visited by DLCHECKSOLVED (line 17). Now, unlike y, which was initially labeled correctly, x is not even depth-t-solved at the time of labeling. Therefore, the assumption is crucial for the correctness of Proposition 5. Unfortunately, there is no simple way fully address this issue without resorting to some cumbersome backtracking, and no way to predict whether such an increment will happen on a given run of FLARES. Nevertheless, our experiments suggest that this event is uncommon in practice (it was never observed). Moreover, we can obtain a revised labeling error guarantee during planning, by keeping track of all states for which a Bellman backup increased the residual above ϵ, and use the maximum of those residuals as the revised error.

Next we prove that FLARES is guaranteed to terminate in a finite number of iterations.

Theorem 1. *With an admissible and monotone heuristic, FLARES terminates after at most $1/\epsilon \sum_{s \in S}[V^*(s) - h(s)]$ trials.*

Proof. The proof follows from a similar argument to the proof of LRTDP's termination. Under the assumptions on the heuristic, the application of a Bellman backup always has a non-decreasing in the value function. So, each call to DLCHECKSOLVED either labels a state as solved, or increases the value of some state by more than ϵ. The trials of FLARES are guaranteed to terminate under an admissible heuristic, thus after each trial there are one or more calls to DLCHECKSOLVED. The bound in the theorem follows immediately from these two properties. □

Even though this is the same bound as LRTDP's, in practice convergence happens much faster because the final values computed by FLARES are only lower bounds on the optimal values. Unfortunately, like other methods that choose actions based on lower bounds, it is possible to construct examples where the final policy returned by FLARES can be arbitrarily bad. On the other hand, it is easy to see that FLARES is asymptotically optimal as $t \to \infty$ because it simply turns into the LRTDP algorithm.

In fact, as the following theorem shows, there exists a finite value of t for which FLARES returns the optimal policy. It is then easy to construct an optimal SSP solver using FLARES, by running FLARES with increasing values of t until $s_0.\text{SOLV} = \textit{true}$, and clearing all D-SOLV labels before each run.

Theorem 2. *With an admissible and monotone heuristic, there exists a finite t for which FLARES converges to the ϵ-optimal value function.*

Proof. Since the state space is finite, there exists a finite value of t for which all calls to DLCHECKSOLVED cover the same set of states as CHECKSOLVED (a trivial solution is $t \geq |S|$). Under these conditions, the algorithm becomes equivalent to LRTDP, and is thus optimal. □

5.1.2 Illustrative Example: A Simple Grid World Problem

We illustrate the advantages of FLARES over other short-sighted solvers by means of a simple toy domain. Consider the grid world shown in Figure 5.2. The agent can move in any of the four grid directions (up, down, right, left). After moving, there is a 0.7 probability of succeeding or a 0.3 probability of moving in another direction (chosen uniformly at random). The cost of moving is 1, except for some "dangerous" cells (highlighted in gray) with cost 20; additionally, some cells have obstacles that

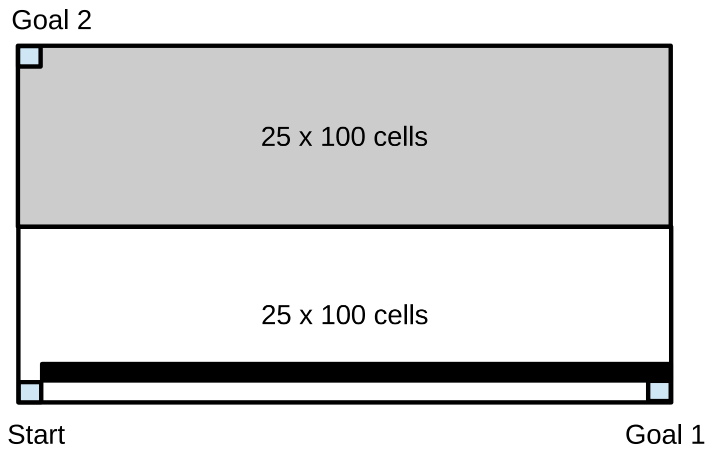

Figure 5.2: Grid world illustrating the advantages of FLARES.

algorithm	cost	time
LRTDP	135	34.02
FLARES(0)	134.43 ± 0.88	1.28
HDP(3,0)	135.28 ± 0.82	0.62
SSIPP(64)	136.85 ± 0.96	10.53

Table 5.1: Results on the grid world shown in Figure 5.2.

cannot be crossed (shown in black). The grid has width 100 and height 51, for a total of 5100 states. The start state is at the bottom left corner, and there are two goals, one at the top-left corner and one at the bottom-right. The optimal policy attempts to reach the goal state to the right, so that the agent avoids the dangerous states in the top part of the map.

Table 5.1 shows the expected cost (mean and standard error) and average planning time for each of the algorithms; the cost shown for LRTDP is the optimal cost estimated by the algorithm. Notably, FLARES with $t = 0$ already returns essentially the optimal policy, while being on average two orders of magnitude faster than LRTDP. Although HDP(I,J) is even faster on this problem, it required some parameter tuning to find appropriate values for i and j. The parameter settings shown are the lowest value of i for which results comparable to FLARES(0) are obtained.

On the other hand, SSIPP is slower than the other approximate methods, and substantial parameter tuning was also required. Table 5.1 shows only results obtained with $t = 64$, which is the first value of t (in powers of 2) that results in comparable expected costs to FLARES(0). Note the large horizons required to find a good policy, resulting in a very large running time, which is close to 8 times slower than FLARES(0).

This simple problem highlights several qualities of FLARES. First, although an optimal policy for this problem must cover the entire state space, every state outside the narrow corridor at the bottom is only reached with low probability. This is an example of a problem where an optimal solver would be unnecessarily slow. On the other hand, FLARES only needs to realize that the policy going up leads to a high cost, which happens during the first few trials. Then, once the algorithm switches to the policy that moves to the right, it quickly stops when all states in the corridor reach a low residual error. Second, since the algorithm is only short-sighted during labeling, but its trials are unrestricted, it can quickly account for the dangerous states far away from the start. This is the reason why $t = 0$ can already generate good policies. On the other hand, limiting the search to states close to the start, requires larger horizons to achieve comparable results.

5.2 Soft Labeling in SSPs

In this section we introduce *soft-labeling*, a generalization of the short-sighted labeling approach underlying FLARES, which is also able to capture the behavior of an optimal labeling algorithm such as LRTDP.

5.2.1 Generalizing Labeling

Labeling in SSPs can be interpreted as an outcome selection mechanism (Keller and Helmert, 2013) that continually modifies the transition function used for sampling states during planning. Specifically, it modifies the probabilities of sampling successor states guaranteed to remain ϵ-consistent, making these probabilities equal to zero. We introduce a generalization of this idea, which we call *soft labeling*, in which a label is interpreted a probabilistic factor that modifies the transition function used for sampling. We begin formalizing the notion of soft labeling by introducing a few key definitions.

Definition 9. ***Deterministic policy graph rooted at a state.*** *Given an SSP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{C}, s_0, \mathcal{G} \rangle$ and a state $s \in \mathcal{S}$, the deterministic policy graph rooted at state s is a directed graph $G_{s,\pi} = (\mathcal{S}_{s,\pi}, E_\pi)$, where the set of vertices, $\mathcal{S}_{s,\pi}$, is the set of all states reachable from s by following policy π, and E_π is a set of edges $\{\langle s', s'' \rangle \mid \mathcal{T}(s', \pi(s'), s'') > 0\}$.*

That is, $G_{s,\pi}$ is a directed graph containing a vertex for every state reachable from s following policy π, and an edge connecting two states whenever one is a possible outcome of the other under π.

Definition 10. ***Weighted distance between states.*** *Let $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{C}, s_0, \mathcal{G} \rangle$ be an SSP and s a state in $\mathcal{S}$. Furthermore, let $G^{(w)}_{s,\pi}$ be the deterministic policy graph, weighted with a function $w : E^\pi \rightarrow \mathbb{R}^+_0$ that assigns a non-negative weight to each edge. Then, the weighted distance between s and $s' \in \mathcal{S} \backslash \{s\}$, $\delta(s, s')$ is the total weight of the deterministic shortest path between s and s' in the weighted deterministic policy graph. When $s = s'$, $\delta(s, s') \triangleq 0$.*

This notion of weighted distance allows us to characterize different measures of short-sightedness using a single notation. For example, the most common—depth-based—form, which we used in the FLARES algorithm, can be represented by assigning $w(\langle s, s' \rangle) = 1$ to every edge in $G^{(w)}_{s,\pi}$. Additionally, we can represent other forms of short-sightedness based on trajectory probabilities (Trevizan and Veloso, 2014), using

$$w(\langle s, s' \rangle) = -\log_2 \mathcal{T}(s, \pi(s), s') \tag{5.2}$$

or plausibilities (Bonet and Geffner, 2003b), using

$$w(\langle s, s' \rangle) = \Big\lfloor -\log_2 \Big(\frac{\mathcal{T}(s, \pi(s), s')}{\max_{s''} \mathcal{T}(s, \pi(s), s'')} \Big) \Big\rfloor \tag{5.3}$$

Given a weight function, w, we define the ϵ-distance of state s, $\mathfrak{d}_\epsilon(s)$, as the shortest weighted distance from s to a state that is not ϵ-consistent.

Definition 11. ***ϵ-distance of a state.*** *Let $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{C}, s_0, \mathcal{G} \rangle$ be an SSP, s a state in $\mathcal{S}$, and $G^{(w)}_{s,\pi}$ be the weighted deterministic policy graph. Furthermore, let $V : \mathcal{S} \rightarrow \mathbb{R}$ be a value function. The ϵ-distance of state s, $\mathfrak{d}_\epsilon(s)$, is defined as $\mathfrak{d}_\epsilon \triangleq \min_{s' \in \mathcal{S}^{\epsilon+}_s} \delta(s, s')$, where $\mathcal{S}^{\epsilon+}_s \triangleq \{s' \in \mathcal{S}_{s,\pi} \mid Res^V(s) > \epsilon\}$, and $\mathfrak{d}_\epsilon = \infty$ if $\mathcal{S}^{\epsilon+}_s = \emptyset$.*

Note that $\mathfrak{d}_\epsilon(s)$ generally depends on the weight function as well as the current policy and value function. For the sake of clarity, we omit these details from the notation, but we make sure that in the rest of this chapter the correct conditions are clear from the context. Figure 5.3 illustrates the ϵ-distance of a few states on a small SSP, using a depth-based weight function. Assuming $\epsilon = 0.1$, the red edges illustrates the path to the state at the shortest weighted distance from A (G), using $w(\langle s, s' \rangle) = 1$, which results in $\mathfrak{d}_\epsilon(A) = 2$. The blue path shows the corresponding path (to state H) when w is defined as in (5.2), resulting in $\mathfrak{d}_\epsilon(A) = 0.46$.

We can use the concept of ϵ-distance to provide a concise definition of the typical criterion for labeling states in SSPs solvers (Bonet and Geffner, 2003a): *a state s should be labeled only if $\mathfrak{d}_\epsilon(s) = \infty$ under the current value function V and a greedy policy over V (irrespective of the weight function used).* Additionally, the depth-based short-sighted labeling criterion of FLARES can be described as: *a state s should be labeled only if $\mathfrak{d}_\epsilon(s) \geq t$, where t is an input parameter and the ϵ-distance is conditioned on the current value function V, a greedy policy π over V, and $w(e) = 1$ for every edge in the deterministic policy graph.*

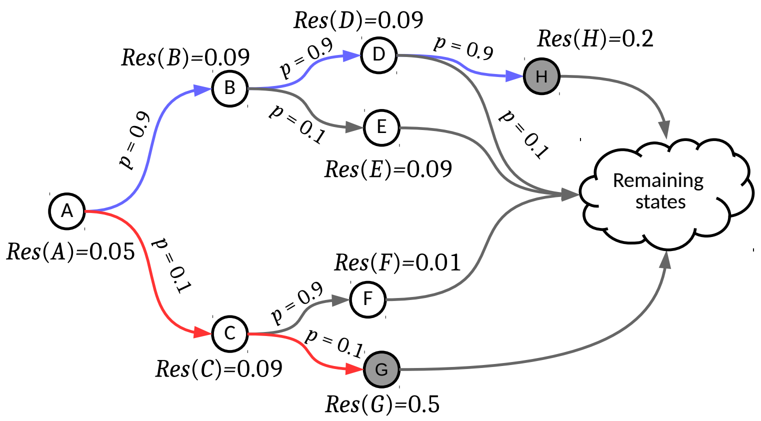

Figure 5.3: Illustration of the ϵ-distance of a state under a given policy, for two different distance functions.

Both of these labeling criteria consider states "solved" once they are labeled. In practice, this means that trial-based algorithms using labeling (e.g., LRTDP or FLARES) stop trials as soon as they encounter labeled states. We can describe this process via a *sampling function*, $\sigma : \mathcal{S} \times \mathcal{A} \times \mathcal{S}^+ \to [0, 1]$, such that $\sigma(s, a, s')$ represents the probability that a trial continues in state s' if action a is chosen when visiting state s. We use the notation $\mathcal{S}^+ \triangleq \mathcal{S} \cup \{\hat{s}\}$, where $\hat{s}$ is a dummy state that represents that the current trial stops. Note that σ only affects the algorithm's choice of explored states, not the computation of values using (2.8).

Building on this notation, we can represent labeling-based sampling via

$$\sigma(s, a, s') = \begin{cases} \big(1 - \mathcal{L}(s')\big) \cdot \mathcal{T}(s, a, s') & \text{if } s' \in \mathcal{S} \\ \sum_{x \in \mathcal{S}} \mathcal{L}(x) \cdot \mathcal{T}(s, a, x) & \text{if } s' = \hat{s} \end{cases} \tag{5.4}$$

where the label, $\mathcal{L}$, is a factor that alters the probability of a trial continuing at a given successor state.

This definition of labeling generalizes existing forms of labeling, which can be recovered from (5.4) with appropriate definitions of $\mathcal{L}$. For example, to recover the labeling used in LRTDP, $\mathcal{L}$ should be defined as

$$\mathcal{L}(s') \triangleq [\mathfrak{d}_\epsilon(s') = \infty] \tag{5.5}$$

where $\mathfrak{d}_\epsilon(s')$ is conditioned on the current value function V, the greedy policy over V, and an arbitrary weighting function w; $[\cdot]$ denotes an Iverson bracket. For the depth-based short-sighted labeling used in FLARES, the label is

$$\mathcal{L}(s') \triangleq [\forall s'' \in \mathcal{S}_{s',\pi} \cap \{x : \delta(\langle s', s''\rangle) \leq t)\}, \quad \mathfrak{d}_\epsilon(s'') \geq t] \tag{5.6}$$

where $\mathfrak{d}_\epsilon(s'')$ is conditioned on V, its associated greedy policy π, and the weight function $w(\langle s, s'\rangle) = 1$. While this labeling function might seem overly complicated, in later sections we show that we can generalize this behavior by means of a generic procedure for estimating ϵ-distances, while letting the labeling functions be directly dependent only on the estimated value of $\mathfrak{d}_\epsilon(s')$.

5.2.2 Soft Labeling

Algorithm 17 presents a generic trial-based solver based on the soft labeling framework described above. The algorithm receives a labeling function, $\mathcal{L}$, a weight function used to compute distances, w, the residual tolerance to be used, ϵ, the number of trials to perform, n, and a vector with additional parameters, θ (e.g., the horizon t in FLARES).

The algorithm starts by initializing ϵ-distances of all states (line 1)[1]. Typically, ϵ-distances should be initialized so that $\mathcal{L}(s) = 0$ (e.g., by setting $\mathfrak{d}_\epsilon(s)$ to $-\infty$), but we allow room for other possibilities, such as keeping ϵ-distances computed during previous calls to the solver on the same input problem. The algorithm functions in a manner much similar to LRTDP, with the following differences:

[1]In practice this should be done lazily, i.e., whenever a state's ϵ-distance needs to be used the first time. We explicitly include it here to highlight the prominent role ϵ-distances play on the algorithm.

- The label s.SOLVED is replaced by a sample $[x \sim \mathit{Bernoulli}(\mathcal{L}(s)) = 1]$ (see lines 13 and 18). This choice is consistent with the probabilistic interpretation of $\mathcal{L}$, and will become more relevant when we introduce soft versions of $\mathcal{L}$.

- The states sampled during the trials (line 12) are sampled according to (5.4) (function SAMPLE-FROM-SIGMA).

- The call to CHECK-SOLVED(s) is replaced with ESTIMATE-ϵ-distance(s) (line 17) . The objective of this function is to explore states in $\mathcal{S}_{s,\pi}$, where π is the greedy policy on the current value estimates, and estimate ϵ-distances for s (and possibly for other states in the graph). The function receives the distance function to be used, w, and any additional parameters necessary, θ.

Algorithm 17: A generic soft labeling trial-based algorithm based on RTDP.

```
SOFT-LABELED-RTDP
    input  : $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{C}, s_0, \mathcal{G} \rangle, \mathcal{L}, w, \epsilon, n, \theta$
    output : an action to execute
1   $\forall s \in \mathcal{S}$, $\mathfrak{d}_\epsilon(s) \leftarrow$ INITIALIZE-$\epsilon$-DISTANCE$(s)$
2   $i \leftarrow 0$
3   while $i < n$ do
4       $i \leftarrow i + 1$
5       $s = s_0$
6       $visited \leftarrow$ EMPTY-STACK
7       while true do
8           $visited$.PUSH$(s)$
9           if $s \in \mathcal{G}$ then break
10          BELLMAN-UPDATE$(s)$
11          $a \leftarrow$ GREEDY-ACTION$(s)$
12          $s \leftarrow$ SAMPLE-FROM-SIGMA$(s, a, \mathcal{T}, \mathcal{L}, \mathfrak{d})$
13          if $[x \sim \mathit{Bernoulli}(\mathcal{L}(s)) = 1]$ then
14              break
15      while $visited \neq$ EMPTY-STACK do
16          $s \leftarrow visited$.POP()
17          $\mathfrak{d} \leftarrow$ ESTIMATE-$\epsilon$-distance$(\mathcal{M}, s, w, \theta)$
18          if $[x \sim \mathit{Bernoulli}(\mathcal{L}(s)) = 0]$ then
19              break
20  return GREEDY-ACTION$(s_0)$
```

Algorithm 18: A depth limited procedure to compute ϵ-distances.

```
ESTIMATE-$\epsilon$-distance
    input : $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{C}, s_0, \mathcal{G} \rangle, s, w, \epsilon, \psi, t$
1   $\textit{no-high-res} \leftarrow \textit{true}$
2   $\textit{open} \leftarrow$ EMPTY-STACK
3   $\textit{closed} \leftarrow$ EMPTY-STACK
4   $\textit{all} \leftarrow \textit{true}$
5   $z \sim \textit{Bernoulli}(\psi)$
6   $h \leftarrow [z = 0] \cdot t + [z = 1] \cdot \infty$
7   if $[x \sim \textit{Bernoulli}(\mathcal{L}(s)) = 1]$ then
8       $\textit{open}.$PUSH$(\langle s, 0 \rangle)$
9   while $\textit{open} \neq$ EMPTY-STACK do
10      $\langle s, d \rangle \leftarrow \textit{open}.$POP()
11      if $d > 2h$ then
12          $\textit{all} \leftarrow \textit{false}$
13          continue
14      $\textit{closed}.$PUSH$(\langle s, d \rangle)$
15      if $s.$RESIDUAL$() > \epsilon$ then
16          $\textit{no-high-res} \leftarrow \textit{false}$
17      $a \leftarrow$ GREEDY-ACTION$(s)$
18      for $s' \in \{s' \in \mathcal{S} | \ \mathcal{T}(s, a, s') > 0\}$ do
19          if $\big([x \sim \textit{Bernoulli}(\mathcal{L}(s)) = 0]$
             $\vee h = \infty\big) \wedge s' \notin \textit{closed}$ then
20              $\textit{open}.$PUSH$\big(\langle s', d + w(\langle s, s' \rangle) \rangle\big)$
21          else if $\mathfrak{d}_\epsilon(s') \neq \infty \wedge s' \notin \textit{closed}$ then
22              $\textit{all} = \textit{false}$
23  if $\textit{no-high-res}$ then
24      for $\langle s', d \rangle \in \textit{closed}$ do
25          if $\textit{all}$ then
26              $\mathfrak{d}_\epsilon(s') = \infty$
27          else if $d \leq t$ then
28              $\mathfrak{d}_\epsilon(s') = t - d$
29  else
30      while $\textit{closed} \neq$ EMPTY-STACK do
31          $\langle s', d \rangle = \textit{closed}.$POP()
32          BELLMAN-UPDATE$(s)$
```

This generic SOFT-LRTDP algorithm generalizes LRTDP and FLARES, as long as ESTIMATE-ϵ-distance(s) is instantiated appropriately. For example, for obtaining LRTDP, it needs to explore all states in $s' \in S_{s,\pi}$, and set $\mathfrak{d}_\epsilon(s') \leftarrow \infty$ iff $\text{Res}^V(s') \leq \epsilon$

for all s'. For obtaining FLARES, we can implement ESTIMATE-ϵ-distanceas a short-sighted version of CHECK-SOLVED(s) that: i) limits the search to depth $2t$, and ii) sets $\mathfrak{d}_\epsilon(s') = t$ for any s' found up to depth t iff all states explored satisfy $\text{Res}^V(s') \leq \epsilon$. Moreover, this framework allows extensions of FLARES that use other distance measures, such as trajectory probabilities.

Although reformulating existing algorithms in this light is somewhat interesting, it does not immediately result in drastically different solution methods for SSPs. However, as we show next, the real power of this framework is that it directly implies a family of labeling mechanisms that achieve the computational efficiency of FLARES, while still maintaining theoretical guarantees of performance. The main insight is to realize that there is nothing forcing us to use an indicator function for $\mathcal{L}$; in fact, we can use any arbitrary function $\mathcal{L} : \mathcal{S} \to [0, 1]$. We refer to the resulting outcome selection approach as *soft labeling* because it allows the labeling function, $\mathcal{L}$, to deter—but not prevent—a state from being explored.

The use of soft labeling has important theoretical advantages over FLARES. In particular, Theorem 3 shows conditions under which SOFT-LRTDP can produce optimal policies, by operating similarly to RTDP. Further, Theorem 4 shows conditions on ESTIMATE-ϵ-distance under which the algorithm converges to ϵ-consistent values with high probability, thus operating similarly to LRTDP. Note that, crucially, the use of ψ in Theorem 4 implies the existence of a wide spectrum of short-sighted labeling strategies that allow SOFT-LRTDP to bridge the gap between RTDP and LRTDP.

Theorem 3. *Given, i) a labeling function $\mathcal{L}$ such that $\forall s' \in \mathcal{S}, \mathcal{L}(s') < \eta < 1$, for some fixed η, and ii) an implementation of ESTIMATE-ϵ-distance(s) that only changes state values through Bellman backups, and iii) an admissible initial value function, then repeated trials of Algorithm 17 eventually yield optimal values over all states reachable by a greedy policy on the states values.*

Proof. This follows from the optimality of asynchronous value iteration and RTDP (Barto et al., 1995). Conditions i-iii) ensure that SOFT-LRTDP operates like RTDP, with the only difference being the sampling probabilities used during the trials. Restricting $\mathcal{L}(s') < \eta < 1$ guarantees that repeated trials can visit states in any optimal policy an infinitely often. □

Theorem 4. *Consider, i) a labeling function $\mathcal{L}$ such that $\mathcal{L}(s') = 1$ iff $\mathfrak{d}_\epsilon(s') = \infty$; ii) an implementation of ESTIMATE-ϵ-distance(s) that sets $\mathfrak{d}_\epsilon(s') = \infty$ iff $\mathcal{S}_{s'}^{\epsilon+} = \emptyset$, only changes state values through Bellman backups, and with probability $\psi > 0$ it explores all states in $\mathcal{S}_{s,\pi}$; and iii) an admissible initial value function. Then, under conditions i-iii) and for any $0 < p < 1$, there exists a value $N_p > 0$ s.t. the probability that N_p trials of Algorithm 17 yield ϵ-consistent values over all states reachable by the greedy policy is higher than p.*

Proof. Under conditions i-iii) there is a probability ψ that ESTIMATE-ϵ-distance operates exactly like the CHECK-SOLVED function of LRTDP. Suppose SOFT-FLARES never terminates under these conditions when $n \to \infty$. Then, there must be a state s_i such that ESTIMATE-ϵ-distance(s_i) is called an infinite number of times. Also, following Bonet and Geffner (Bonet and Geffner, 2003a), there is a finite number of calls to CHECK-SOLVED(s_i) after which $\mathcal{S}_{s_i}^{\epsilon+} = \emptyset$. This maximum number of calls to CHECK-SOLVED(s_i) is bounded by

$$C = \epsilon^{-1} \sum_{s \in S} V^*(s) - h(s) \tag{5.7}$$

where $h(s)$ is the initial admissible value function. Because $\psi > 0$, we can then bound the probability that after n calls to ESTIMATE-ϵ-distance(s_i), $\mathcal{S}_{s_i}^{\epsilon+} \neq \emptyset$. Let X be the random variable representing the total number of calls to ESTIMATE-ϵ-distance(s_i) that are equivalent to CHECK-SOLVED(s_i). Then, by applying Chernoff bound,

$$Pr(X \leq C) \leq \exp\big\{ -(n\psi - C)^2 / 2n\psi \big\} \tag{5.8}$$

Thus, for any $0 < q < 1$, we have that, as long as $N_q - \sqrt{2N_q\psi \log 1/q} > C$, then N_q calls ensure $Pr(X \leq C) < q$. Together with conditions i-ii), this implies that after N_q calls to ESTIMATE-ϵ-distance, s_i will be labeled $\mathcal{L}(s_i) = 1$ with probability higher than q. Moreover, since q can be arbitrarily small and the number of states is finite, this implies that for any probability $p < q$ there is a number of SOFT-FLARES trials, N_p, after which $\mathcal{L}(s_0) = 1$. □

Next, we provide a short-sighted implementation of ESTIMATE-ϵ-distance(s), which, coupled with appropriate labeling functions, satisfies the conditions of Theorem 4. This implementation is outlined in Algorithm 18.

Algorithm 18 closely follows the labeling procedure of FLARES, with some major differences. First, as in Algorithm 17, all boolean label checks have been replaced by Bernoulli trials with probability $\mathcal{L}(s)$ (lines 7 an 19). Second, labeling is done by modifying the ϵ-distances of states, instead of assigning hard labels (lines 26 and 28). Third, the short-sighted horizon, h, is set to infinity with probability ψ, allowing $\mathcal{S}_{s,\pi}$ to be explored fully.

In more detail, the algorithm works as follows. Given a state s, Algorithm 18 expands all states in $\mathcal{S}_{s,\pi}$ up to distance $2h$, and checks if all of these are ϵ-consistent (lines 15-16); the notion of distance to use is specified by the function w (see line 20). If all of the states found are ϵ-consistent, the algorithm then modifies ϵ-distance estimates according to the distance—from s—at which the state was first found (lines 23-28). If it turns out that $\mathcal{S}_{s,\pi}$ lies completely within the horizon $2h$ (when variable *all* is true), then $\mathfrak{d}_\epsilon(s')$ is set to ∞ for all states found, since this condition is the usual requirement for correctly hard-labeling states.

Note that, in the case where only finite ϵ-distances can be assigned (line 28), our distance estimate slightly departs from Definition 11. A more accurate estimate would be $\mathfrak{d}_\epsilon(s') \leftarrow 2t - d$, considering that there are ϵ-consistent states at distances t to $2t$ from the initial state s. Similarly, we could also have assigned ϵ-distances for all states found up to distance $2t$, instead of only for those at distance $\leq t$ (line 27). However, we took the more conservative approach shown here because it is equivalent to a more robust version of FLARES; one that labels the same set of states, but that uses soft labels instead. But, in contrast to FLARES, the use of soft-labeling allows the planner to explore, with some probability, states that it has previously labeled as "probably solved".

Finally, what are good labeling functions to use? Intuitively, we want increasing functions of the ϵ-distance, to encourage sampling towards states that are "more likely" to be far away from convergence. In our experiments we consider the following labeling functions, for the case when $t > 0$:

- **Linear**: $\mathcal{L}(s) \triangleq \frac{\beta-\alpha}{t}\mathfrak{d}_\epsilon(s) + \alpha$
- **Logistic**: $\mathcal{L}(s) \triangleq \frac{1}{1+\frac{1-\alpha}{\alpha}\exp\{-\frac{1}{t}\ln\frac{(1-\alpha)\beta}{\alpha(1-\beta)}\cdot\mathfrak{d}_\epsilon(s)\}}$
- **Exponential**: $\mathcal{L}(s) \triangleq \alpha\exp\{\frac{1}{t}\ln\frac{\beta}{\alpha}\cdot\mathfrak{d}_\epsilon(s)\}$

where α and β are parameters that represent the desired labeling probability for $\mathfrak{d}_\epsilon(s) = 0$ and $\mathfrak{d}_\epsilon(t)$, respectively. In all cases, we assume that $\mathcal{L}(s) = 0$ if $\mathfrak{d}_\epsilon(s) < 0$ and $\mathcal{L}(s) = \beta$ if $\mathfrak{d}_\epsilon(s) \geq t$.

5.3 Experiments

In this section we empirically evaluate the use of soft labeling for approximately solving SSPs, denoting the combination of Algorithms 17 and 18 as SOFT-FLARES. The goal of these experiments was to demonstrate that the general soft labeling framework can produce algorithms competitive with state-of-the-art methods, both in terms of expected cost and total planning time. We compared different variants of SOFT-FLARES to several SSP solvers: RFF (Teichteil-Königsbuch et al., 2010), LRTDP (Bonet and Geffner, 2003a), FLARES, HDP (Bonet and Geffner, 2003b), BRTDP (McMahan et al., 2005), and SSIPP (Trevizan and Veloso, 2014). We did not perform extensive experiments with LABELED-SSIPP, since preliminary experiments indicate that run time was never better than that of LRTDP. Similarly, we did not compare with VPI-RTDP (Sanner et al., 2009) as it seems to be easily affected by the quality of initial upper bounds; in our experiments we have been unable to reproduce the results in (Sanner et al., 2009).

Algorithm	Exp. cost	Time (seconds)
SOFT-FLARES-T-EXP(4)	89.44 ± 1.62	5.50
LRTDP	91.63 ± 1.71	12.56
FLARES(4)	91.70 ± 1.55	5.72
HDP(0)	92.16 ± 1.65	8.83
SSIPP(8)	92.29 ± 1.68	19.15
BRTDP	93.73 ± 1.83	12.97

Table 5.2: Expected cost and total planning of several planning algorithms on the sailing domain (middle-goal).

Algorithm	Exp. cost	Time (seconds)
SOFT-FLARES-T-EXP(2)	177.32 ± 2.32	9.34
LRTDP	177.81 ± 2.50	16.93
BRTDP	178.70 ± 2.40	45.21
HDP(0)	179.63 ± 2.42	13.81
FLARES(3)	179.72 ± 2.37	11.67
SSIPP(4)	191.03 ± 2.57	25.64

Table 5.3: Expected cost and total planning of several planning algorithms on an instance of the sailing domain (corner-goal).

We use the notation ALGORITHM(X) to denote the short-sighted horizon, X, used by the algorithm. In the case of SOFT-FLARES and SSIPP, the notation ALGORITHM-DIST-LABEL; refers to a distance function, DIST, (D for depth, T for trajectory probability, or P for plausibility), and a label function, LABEL, (LINear, LOGistic, or EXPonential). In all cases we used $\alpha = 0.1$ and $\beta = 0.9$ for SOFT-FLARES.

For SOFT-FLARES, we also set $\psi = 0$, since we will evaluate the quality of the resulting policies empirically. We used the h_{min} heuristic (Bonet and Geffner, 2003a), pre-computed for all states before planning started. We evaluated different parameterizations of the algorithms with distances from 0 to 4 for HDP, FLARES and SOFT-FLARES, distances in $\{1, 2, 4, 8\}$ for SSIPP, and values of $\rho = 2^{-5}$ and $\rho = 2^{-4}$ for TRAJECTORY-BASED-SSIPP (Trevizan and Veloso, 2012); we report the results of the parameterizations with the best expected cost in the problems considered.

All experiments were performed on Xeon E5-2680 v4 @ 2.40GHz computers. The performance of a planner is evaluated by running simulations of the partial policy implied by the algorithm's action selection, and computing the resulting expected cost and total time spent on planning. We reset any internal state of the algorithms before each simulation starts, to evaluate their performance in a one-shot planning task. Note that this is harder than typical competition settings, where planners are allowed to reuse computation from previous simulations. Actions are selected greedily on the current value estimates, and we allow the algorithm to re-plan if necessary, adding the accrued time to the total. For SSIPP, the algorithm re-plans before each action, for HDP re-planning is done as described by (Bonet and Geffner, 2003b), and for SOFT-FLARES it is done whenever a soft-label check fails.[2]

5.3.1 Sailing Domain

Our first evaluation benchmark is the sailing problem (Kocsis and Szepesvári, 2006). We evaluated on two instances of this domain, both with size 40×40 (12,801 states), differing in the goal location (corner or middle of the grid). We consider the performance of the algorithms when there is no time limit per action, considering the following stopping criteria (whichever happens first): successful label checks for FLARES, SOFT-FLARES, and HDP; 1000 trials for LRTDP, BRTDP, FLARES, and

[2]The code to reproduce these experiments will be available once anonymity is no longer needed.

SOFT-FLARES; a single simulated trial reaching a goal for SSIPP. Tables 5.2 and 5.3 show the expected costs and total planning times of these experiments, averaged over 250 simulations; standard errors are also shown for the expected cost (the variance for the planning time was negligible). In both of the problem instances considered, a parameterization of SOFT-FLARES was able to achieve the policy with the best expected cost, while requiring less planning time than the remaining algorithms. Incidentally, the best parameterization of SOFT-FLARES involved the trajectory probability distance and exponential labeling functions, although other parameterizations achieved similar performance. These results suggest that our soft labeling framework is flexible enough to produce a diverse set of planners having different trade-offs between quality and computational time.

Algorithm	Exp. cost	Time (seconds)
FLARES(4)	26.66 ± 0.35	8.37
SOFT-FLARES-T-EXP(3)	26.88 ± 0.36	4.06
HDP(3,0)	26.93 ± 0.39	4.07
BRTDP	27.24 ± 0.37	9.25
SSIPP(8)	27.51 ± 0.39	28.40
LRTDP	27.62 ± 0.41	12.13

Table 5.4: Expected cost and total planning of several planning algorithms on the racetrack domain (ring-5).

Algorithm	Exp. cost	Time (seconds)
SOFT-FLARES-D-LOG(2)	11.46 ± 0.08	0.82
LRTDP	11.52 ± 0.11	35.39
SSIPP(8)	11.57 ± 0.11	32.40
BRTDP	11.64 ± 0.07	2.70
FLARES(3)	11.65 ± 0.11	2.67
HDP(2,0)	11.66 ± 0.11	0.21

Table 5.5: Expected cost and total planning of several planning algorithms on the racetrack domain (square-4).

Algorithm	p01	p02	p03	p04	p05	p06	p07	p08	p09	p10
TRIANGLE-TIREWORLD										
SOFT-FLARES	50	50	50	50	50	50	50	49	49	41
RFF	50	50	50	50	50	50	50	0	0	0
TRAJECTORY-BASED-SSIPP	50	50	48	46	46	38	40	33	42	31
EX-BLOCKSWORLD										
SOFT-FLARES	45	15	17	21	50	48	50	27	23	1
RFF	31	7	25	10	50	12	41	6	5	0
TRAJECTORY-BASED-SSIPP	0	0	31	26	50	46	0	0	0	0

Table 5.6: Number of runs in which planners were able to successfully reach the goal for two IPPC'08 domains.

5.3.2 Racetrack Domain

Our second evaluation benchmark is the racetrack domain, first proposed in (Barto et al., 1995), with the modifications described in Chapter 3. We used a probability of 0.20 for slipping, and a probability of 0.10 for randomly changing accelerations. We experimented with two problems instances, one with 92,909 states (ring-5) and one with 400,270 states (square-4). The results are shown in Tables 5.4 and 5.5, respectively. In the two problem instances considered a parameterization of SOFT-FLARES was among the best two planners, both in terms of expected cost and total planning time. These results offer additional evidence in favor of the utility of soft labeling as a framework for probabilistic planning.

5.3.3 International Planning Competition Domains

We assessed the scalability of SOFT-FLARES to problems with very large state spaces, using two domains from the International Planning Competition held in 2008 (the last competition involving goal-based MDPs) (Bryce and Buffet, 2008). We used problems 1-10 from domains TRIANGLE-TIREWORLD and EX-BLOCKSWORLD. As typical in competition settings, we gave planners 20 minutes to successfully complete 50 runs of each problem. For SOFT-FLARES, we used $t = 2$, an exponential labeling function and the inadmissible FF heuristic (Bonet and Geffner, 2005). We compare the performance of SOFT-FLARES with our implementation of RFF (Teichteil-Königsbuch et al., 2010), the winner of IPPC'08, and with trajectory-based SSIPP (Trevizan and Veloso, 2012), using $\rho = 0.25$ and the h_{add} heuristic (SSIPP code provided by the original author); all experiments were run on the same machine. The results, shown in Table 5.6, demonstrate that SOFT-FLARES is able scale to very large problems, outperforming state-of-the-art planners in two probabilistically interesting domains (Little and Thiebaux, 2007). Crucially, it outperforms RFF, *without relying on a classical planner to speed up computation*, providing convincing evidence that soft labeling is a promising framework for scalable and performant probabilistic planning. Note that for TRAJECTORY-BASED-SSIPP, the original work reports 50 in every triangle-tireworld instance (Trevizan and Veloso, 2012), but we could not reproduce these results using the original code.

CHAPTER 6
CONCLUSION

This thesis studied scalable algorithms for solving Markov Decision Processes, based on novel paradigms for efficient state-space exploration. Our contributions push the state-of-the-art in MDP solution methods, and offer flexible frameworks upon which other efficient and high-performing algorithms can be built. In this chapter we briefly discuss the main results of this thesis, and then offer directions for future research.

6.1 Summary of Contributions

6.1.1 $\mathcal{M}_l^k$-reductions

In Chapter 3, we introduced the $\mathcal{M}_l^k$-reduction for MDPs, a model reduction framework that generalizes the popular single-outcome determinization approach, resulting in a more robust, yet computationally efficient, planning paradigm. The main idea behind the $\mathcal{M}_l^k$-reduction is to prune sections of the state space by classifying transition outcomes into one of two types, primary outcomes or exceptional outcomes. Given an outcome classification, planning is done on a modified MDP in which any trajectory can have at most k exceptions: that is, after k exceptions have occurred on a trajectory, the model assumes only primary outcomes are possible. The parameter l represents how many outcomes, at most, are considered primary in the transition function of any state-action pair. Thus, for example, a single-outcome determinization is an instance of an $\mathcal{M}_1^0$-reduction.

By allowing the agent to plan for more than just a single primary outcome, a planner using a $\mathcal{M}_l^k$-reduction is able to compute, as we show in our experiments, near-optimal plans orders of magnitude faster than when planning using the original, non-reduced, model. Moreover, we show how the parameters k and l allow the agent to trade-off computational cost for policy quality.

Importantly, a key result from the research presented in Chapter 3 is that the choice of reduction (i.e., the choice of primary outcomes) is crucial. Indeed, it can make the different between optimal behavior and catastrophic performance, even when using a deterministic reduction. Building on this insight, we presented a greedy approach for learning $\mathcal{M}_l^k$-reductions for a given planning domain, which was used to learn the reductions used in our experiments.

In Chapter 4 we introduced FF-LAO*, an algorithm that directly leverages the structure of $\mathcal{M}_1^k$-reductions to gain computational efficiency. In particular, for MDPs described using factored domain languages like PPDDL, it is possible to create a deterministic version of the original problem in the same language, which can then be solved efficiently using out-of-the-box classical planners (e.g., FF). Thus, since a $\mathcal{M}_1^k$-reduction becomes deterministic in states with exception counters equal to zero, the planner can replace a full search in the deterministic state space with a call to a highly optimized classical planner, which is typically orders of magnitude faster.

We also introduce an approach to learn the best determinization to use, by doing an exhaustive search in the space of action schema representations (i.e., before actions are grounded with the predicates of a particular problem). Our experiments show that the combination of an appropriate determinization choice, and the use of a fast classical planner results in state-of-the-art performance in challenging planning domains with billions of states. Moreover, the ability to use the full model by increasing k allows FF-LAO* to produce better policies when determinization alone is not robust enough.

6.1.2 Soft Labeling

In Chapter 5 we introduced soft labeling, a general algorithmic paradigm for solving SSPs that results in more efficient state-space exploration, while achieving near-optimal performance. In contrast to the $\mathcal{M}_l^k$-reduction, soft labeling works by altering the sampling strategy of a search algorithm, rather than changing the underlying model used for value computation. We started our discussion of soft-labeling by first proposing a variant of RTDP, called FLARES, that uses a short-sighted form of labeling for terminating trials early, potentially pruning large sections of the state space. Since a near-equivalent version of this algorithm can instead sample from the distribution of unlabeled states, we can see this approach as a form of outcome selection.

Building on this insight, we then generalize the behavior of FLARES, by grounding it as a specific instance of our new soft labeling framework. In contrast to deterministic labels, soft labels work by decreasing the probability of sampling labeled states during exploration, rather than completely preventing the planner from visiting these states again. Since this prevents the possibility of starving states during the search, the soft labeling technique can be leveraged to produce algorithms with guarantees of optimality in the limit.

Our soft-labeling framework characterizes different instances according to the choice of short-sighted measure and the labeling function to use. To formalize the notion of short-sighted measure, we introduced the concept of ϵ-distance, a heuristic measure for how close the value of a state is from convergence to ϵ-optimality. The labeling function determines how much the probability of sampling a state decreases as a function of ϵ-distance. Using different combinations of ϵ-distances and labeling functions, we can devise many different planners based on soft-labeling. Our experiments shows that instances of this framework can outperform state-of-the-art SSP solvers, illustrating its power.

6.2 Future Work

The results reported in this thesis suggest that both $\mathcal{M}_l^k$-reductions and soft labeling have significant potential as frameworks upon which efficient and robust MDP solvers can be built. On the other hand, there are several directions for improvement and future work.

The performance of algorithms like $\mathcal{M}_l^k$-REPLAN and FF-LAO* hinges on access to good $\mathcal{M}_l^k$-reductions for the planning problems to be solved. While we have provided two methods that are good initial steps in these direction, both of these build on the assumption that a reduction learned on a small problem instance of a domain will carry over to other instances. The conditions under which this assumption is true are not well understood, and it will be beneficial to formally characterize domains in which it is guaranteed to hold.

Another interesting direction would be to develop methods for learning contextual reductions, in which the choice of primary outcomes for a given action is a function of state features, rather than a single reduced schema universally applied to all states. Although using universal reductions resulted in highly effective planner performance in our experiments, it is possible that many other planning domains would require a more refined choice of reduction.

Conversely, one of the benefits of the soft labeling framework is that it does not depend on learning a reduced model ahead of time. On the other hand, the choice of labeling function and ϵ-distance can have significant impact in the performance of the algorithm. Thus, a deeper study of the theoretical and empirical properties of different combinations of distance and labeling functions, as well as developing additional—more principled—functions, is an immediate direction for future research.

BIBLIOGRAPHY

A. Anand, A. Grover, and P. Singla. ASAP-UCT: Abstraction of state-action pairs in UCT. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

A. Anand, R. Noothigattu, P. Singla, et al. OGA-UCT: On-the-go abstractions in UCT. In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling*, 2016.

P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

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

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

D. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and distributed computation: Numerical methods*. Prentice Hall Englewood Cliffs, NJ, 1989.

D. P. Bertsekas and J. N. Tsitsiklis. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16(3):580–595, 1991.

B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence*, 129 (1-2):5–33, 2001. ISSN 0004-3702.

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

B. Bonet and H. Geffner. Faster heuristic search algorithms for planning with uncertainty and full feedback. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 1233–1238, Acapulco, Mexico, 2003b.

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

B. Bonet and H. Geffner. Action selection for MDPs: Anytime AO* versus UCT. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, pages 1749–1755, 2012.

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

C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

D. Bryce and O. Buffet. Sixth international planning competition: Uncertainty part. In *Proceedings of the Sixth International Planning Competition*, 2008.

O. Buffet and D. Aberdeen. The factored policy-gradient planner. *Artificial Intelligence*, 173(5-6):722–747, 2009.

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

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

G. Comanici, P. Panangaden, and D. Precup. On-the-fly algorithms for bisimulation metrics. In *Proceedings of the Ninth International Conference on Quantitative Evaluation of Systems*, pages 94–103, 2012.

P. Dai and J. Goldsmith. Topological value iteration algorithm for Markov decision processes. In *Proceedings of the Twentieth International Joint Conference on Artifical Intelligence*, pages 1860–1865, San Francisco, CA, USA, 2007.

D. P. de Farias and B. Van Roy. The linear programming approach to approximate dynamic programming. *Operations Research*, 51(6):850–865, Nov. 2003.

R. Dearden, N. Friedman, and S. Russell. Bayesian Q-learning. In *Proceedings of the Tenth Conference on Innovative Applications of Artificial Intelligence*, pages 761–768. American Association for Artificial Intelligence, 1998.

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

Z. Feng and E. A. Hansen. Symbolic heuristic search for factored Markov decision processes. In *Eighteenth National Conference on Artificial Intelligence*, pages 455–460, Menlo Park, CA, USA, 2002.

N. Ferns, P. Panangaden, and D. Precup. Metrics for finite Markov decision processes. In *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence*, pages 162–169, 2004.

N. Ferns, P. S. Castro, D. Precup, and P. Panangaden. Methods for computing state similarity in Markov decision processes. In *Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence*, pages 174–181, Arlington, Virginia, United States, 2006.

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

R. Givan and T. Dean. Model minimization, regression, and propositional STRIPS planning. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 1163–1168, 1997.

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

C. Guestrin, D. Koller, R. Parr, and S. Venkataraman. Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research*, 19:399–468, 2003.

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

J. Hoey, R. St-Aubin, A. Hu, and C. Boutilier. SPUDD: Stochastic planning using decision diagrams. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 279–288, San Francisco, CA, USA, 1999.

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

J. Hostetler, A. Fern, and T. Dietterich. Progressive abstraction refinement for sparse sampling. In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence*, pages 365–374. AUAI Press, 2015.

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

R. A. Howard. Information value theory. *IEEE Transactions on systems science and cybernetics*, 2(1):22–26, 1966.

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

M. Issakkimuthu, A. Fern, R. Khardon, P. Tadepalli, and S. Xue. Hindsight optimization for probabilistic planning with factored actions. In *Proceedings of the Twenty-Fifth International Conference on International Conference on Automated Planning and Scheduling*, pages 120–128, 2015.

N. Jiang, S. Singh, and R. Lewis. Improving UCT planning via approximate homomorphisms. In *Proceedings of the Fourteenth International Conference on Autonomous Agents and Multi-Agent Systems*, pages 1289–1296, 2014.

N. K. Jong and P. Stone. State abstraction discovery from irrelevant state variables. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, volume 8, pages 752–757, 2005.

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

M. Kearns, Y. Mansour, and A. Y. Ng. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *Machine learning*, 49(2):193–208, 2002.

T. Keller. *Anytime Optimal MDP Planning with Trial-based Heuristic Tree Search*. PhD thesis, University of Freiburg, Freiburg im Breisgau, Germany, 2015.

T. Keller and P. Eyerich. PROST: Probabilistic planning based on UCT. In *Twenty-Second International Conference on Automated Planning and Scheduling*, 2012.

T. Keller and M. Helmert. Trial-based heuristic tree search for finite horizon MDPs. In *Proceedings of the Twenty-Third International Conference on International Conference on Automated Planning and Scheduling*, pages 135–143, 2013.

E. Keyder and H. Geffner. Heuristics for planning with action costs revisited. In *Proceedings of the Eighteenth European Conference on Artificial Intelligence*, pages 588–592, 2008a.

E. Keyder and H. Geffner. The HMDPP planner for planning with probabilities. In D. Bryce and O. Buffet, editors, *ICAPS Third International Probabilistic Planning Competition*. 2008b.

L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning. In *Proceedings of the Seventeenth European conference on Machine Learning*, pages 282–293, 2006.

S. Koenig, R. Goodwin, and R. G. Simmons. Robot navigation with Markov models: A framework for path planning and learning with limited computational resources. In L. Dorst, M. Lambalgen, and F. Voorbraak, editors, *Reasoning with Uncertainty in Robotics*, volume 1093 of *Lecture Notes in Computer Science*, pages 322–337. 1996.

A. Kolobov and Mausam. Planning with Markov decision processes: An AI perspective. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1): 1–210, 2012.

A. Kolobov, Mausam, and D. S. Weld. ReTrASE: Integrating paradigms for approximate probabilistic planning. In *Proceedings of the Twenty-First International Joint Conference on Artifical Intelligence*, pages 1746–1753, San Francisco, CA, USA, 2009.

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

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

L. Li, T. J. Walsh, and M. L. Littman. Towards a unified theory of state abstraction for MDPs. In *Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics*, 2006.

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

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

R. A. Mccallum. Overcoming incomplete perception with utile distinction memory. In *Proceedings of the Tenth International Conference on Machine Learning*, 1993.

H. B. McMahan, M. Likhachev, and G. J. Gordon. Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *Proceedings of the Twenty-Second International Conference on Machine Learning*, pages 569–576. ACM, 2005.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

A. W. Moore and C. G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine learning*, 13(1):103–130, 1993.

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

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

L. Pineda and S. Zilberstein. Planning under uncertainty using reduced models: Revisiting determinization. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling*, pages 217–225, Portsmouth, New Hampshire, 2014.

L. Pineda, Y. Lu, S. Zilberstein, and C. V. Goldman. Fault-tolerant planning under uncertainty. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, pages 2350–2356, Beijing, China, 2013.

L. Pineda, T. Takahashi, H.-T. Jung, S. Zilberstein, and R. Grupen. Continual planning for search and rescue robots. In *Proceedings of the IEEE-RAS 15th International Conference on Humanoid Robots*, pages 243–248, Seoul, Korea, 2015.

L. Pineda, K. H. Wray, and S. Zilberstein. Fast SSP solvers using short-sighted labeling. In *Proceedings of the Thirty-First Conference on Artificial Intelligence*, pages 3629–3635, San Francisco, California, 2017.

W. B. Powell. *Approximate Dynamic Programming: Solving the curses of dimensionality*, volume 703. John Wiley & Sons, 2007.

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

B. Ravindran and A. G. Barto. Model minimization in hierarchical reinforcement learning. In *International Symposium on Abstraction, Reformulation, and Approximation*, pages 196–211, 2002.

S. Sanner, R. Goetschalckx, K. Driessens, and G. Shani. Bayesian real-time dynamic programming. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence*, Pasadena, California, 2009.

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

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587): 484–489, 2016.

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

T. Smith and R. Simmons. Focused real-time dynamic programming for MDPs: squeezing more out of a heuristic. In *Proceedings of the Twenty-First National Conference on Artificial intelligence*, pages 1227–1232, Boston, Massachusetts, 2006.

M. Steinmetz, J. Hoffmann, and O. Buffet. Revisiting goal probability analysis in probabilistic planning. In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling*, London, UK, 2016.

R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1998a.

R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998b.

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

A. E. Tate and J. E. Hendler. *Readings in Planning.* Morgan Kaufmann Publishers Inc., 1994.

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

S. Temizer et al. Collision avoidance for unmanned aircraft using Markov decision processes. In *AIAA Guidance, Navigation, and Control Conference, Toronto, Ontario.* American Institute of Aeronautics and Astronautics, 2010.

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

F. W. Trevizan and M. M. Veloso. Trajectory-based short-sighted probabilistic planning. In *Proceedings of Neural Information Processing Systems*, pages 3257–3265, Lake Tahoe, Nevada, 2012.

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

T. J. Walsh, S. Goschin, and M. L. Littman. Integrating sample-based planning and model-based reinforcement learning. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, pages 612–617, Atlanta, Georgia, 2010.

D. S. Weld. Recent advances in AI planning. *AI Magazine*, 20:93–123, 1999.

D. Wingate and K. D. Seppi. Prioritization methods for accelerating MDP solvers. *Journal of Machine Learning Research*, 6(May):851–881, 2005.

K. H. Wray, L. Pineda, and S. Zilberstein. Hierarchical approach to transfer of control in semi-autonomous systems. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pages 517–523, 2016.

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

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

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

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