Department of Mathematics and Computer Science
Institute of Computer Science
Knowledge-Based Systems Group

Relevance-based Online Planning in Complex POMDPs

by

Juan Carlos Saborío Morales

Submitted in partial fulfillment
of the requirements for the degree of
Doctor rerum naturalium (Dr.rer.nat)
in Computer Science

Osnabrück
September, 2019

First supervisor:	Prof. Dr. Joachim Hertzberg
Second supervisor:	Prof. Dr. Marc Toussaint

Abstract

Planning under uncertainty is a central topic at the intersection of disciplines such as artificial intelligence, cognitive science and robotics, and its aim is to enable artificial agents to solve challenging problems through a systematic approach to decision-making. Some of these challenges include generating expectations about different outcomes governed by a probability distribution and estimating the utility of actions based only on partial information. In addition, an agent must incorporate observations or information from the environment into its deliberation process and produce the next best action to execute, based on an updated understanding of the world. This process is commonly modeled as a POMDP, a discrete stochastic system that becomes intractable very quickly. Many real-world problems, however, can be simplified following cues derived from contextual information about the relative expected value of actions. Based on an intuitive approach to problem solving, and relying on ideas related to attention and relevance estimation, we propose a new approach to planning supported by our two main contributions: PGS grants an agent the ability to generate internal preferences and biases to guide action selection, and IRE allows the agent to reduce the dimensionality of complex problems while planning online. Unlike existing work that improves the performance of planning on POMDPs, PGS and IRE do not rely on detailed heuristics or domain knowledge, explicit action hierarchies or manually designed dependencies for state factoring. Our results show that this level of autonomy is important to solve increasingly more challenging problems, where manually designed simplifications scale poorly.

Kurzfassung

Planen unter Unsicherheit ist ein zentrales Thema am Schnittpunkt der künstliche Intelligenz, der Kognitionswissenschaft und der Robotik. Ziel der Planung unter Unsicherheit ist es, künstliche Agenten zubefähigen herausfordernde Probleme durch einen systematischen Ansatz der Entscheidungsfindung zu lösen. Einige dieser Herausforderungen beinhalten die Generierung von Erwartungen unterschiedlicher Ausgänge, welche durch eine Wahrscheinlichkeitsverteilung gesteuert sind, sowie eine Abschätzung der Nützlichkeit von Aktionen basierend auf unvollständigen Informationen. Desweiteren muss ein Agent Beobachtungen und Information aus der Umgebung in seinen Überlegungsprozess integrieren, um die nächstbeste Aktion auszuwählen, basierend auf einem geänderten Verständnis der Welt. Dieser Vorgang ist im Allgemeinen als POMDP modeliert: ein diskretes stochastisches System, welches schnell unlösbar wird. In der Praxis ist es aber möglich viele Probleme durch intuitive Hinweise zu vereinfachen, z.B. mit Kontextinformation über die Erwartungswerte der Aktionen. Inspiriert durch intuitive Annäherungen zur Problemlösung, die auf Aufmerksamkeit und Relevanzschätzung zurückzuführen sind, schlagen wir einen neuen Ansatz für die Planung vor, der auf unseren Hauptbeiträgen basiert ist: PGS erweitert einen Agenten um die Fähigkeit Aktionsauswahlpräferenzen intern zu generieren, und IRE ermöglicht es, Dimensionalitätsreduktionkriterien für komplexe Probleme bei der on-line Planung zu konstruieren. Im Vergleich zu bestehenden Methoden für die Verbesserung der Leistung des POMDP-Planens sind PGS und IRE unabhängig von detaillierten Heuristiken, expliziten Aktionshierarchien und manuell konstruierten Abhängigkeiten für Zustandfaktorisierung. Unsere Ergebnisse zeigen, dass der Grad der Autonomie überaus wichtig für das Lösen immer komplexer werdender Probleme ist, insbesondere dort, wo manuell gestaltete Vereinfachungen mangelhaft skalieren.

Acknowledgments

I am immensely grateful to my supervisor Joachim Hertzberg for his sincere commitment to his role as an academic guide and as a colleague, which made life as a foreign student in Germany much easier. I greatly enjoyed our conversations about life, culture and research. This thesis as well as its related publications are the result of his diligent mentorship.

Many thanks to Marc Toussaint, my second supervisor, for his valuable feedback and warm welcome in Stuttgart. A heartfelt thank you to everybody at the Knowledge-Based Systems group, as well as our colleagues at the DFKI Robotics Innovation Center in Osnabrück (now DFKI Niedersachsen) for contributing to such a comfortable working environment. I am particularly grateful to my former colleague Sven Albrecht for his help with all sorts of issues during my first couple of years in Germany, to Thomas Wiemann for his support in both academic and bureaucratic matters as well as his friendly demeanor at and outside of work, to Sebastian Pütz and Felix Igelbrink whose feedback helped me improve the planning domains, and with whom I shared many lunches and drinks as well as many amusing discussions, and to Lena Herrmann who helped me write the German abstract. Many others at the Institute of Computer Science were always helpful throughout the years, and to them I am also thankful.

For the entirety of my doctoral work I was financially supported by the DAAD. I'd like to thank all of the different people I was in contact with from Section ST31, who always answered my many e-mails cordially and promptly.

My family, in particular my parents, also played a significant role with their enduring support throughout the years. I am however especially thankful to my wife Katarina who accompanied me through this process. I brought home with me many moments of joy but also of frustration, all of which she handled patiently and in return offered encouragement and motivation.

Contents

Abstract **iii**

Kurzfassung **v**

Acknowledgments **vii**

List of Publications **xiii**

List of Tables **xv**

List of Figures **xvii**

List of Algorithms **xix**

List of Acronyms **xxi**

Notation **xxiii**

1 Introduction **1**
1.1 Planning Under Uncertainty: From Principles to Applications 1
1.2 Summary of Contributions 3
1.3 Document Structure 4

2 Background **5**
2.1 Probabilistic Planning in Context 5
2.2 Sequential Decision Problems 6
2.2.1 Markov Decision Processes 9
2.2.2 Dynamic Programming on MDPs 11
2.2.3 Monte Carlo Search 13

2.3 Decision Problems with Partial Observability 17
2.3.1 Partially Observable Markov Decision Processes 18
2.3.2 POMDP Algorithms 20
2.3.3 Partially Observable Monte Carlo Search 21
2.4 Planning and Learning 24
2.5 On Performance and Scalability 25
2.6 Applications in Robot Control 27
2.7 Summary 29

3 Framing the problem **31**
3.1 Challenges and Expectations 31
3.2 Problem Description 32
3.3 Overview of the Solution 33
3.4 Methodology 35
3.5 Planning Domains 36
3.5.1 Rocksample 37
3.5.2 Cellar 38
3.5.3 Big Foot 40
3.6 Summary 42

4 Relevance in Action Selection **43**
4.1 Goal Proximity 43
4.2 PGS in Reward Shaping 48
4.3 PGS in Rollouts and Simulation 50
4.4 Results 54
4.4.1 Taxi 54
4.4.2 Rocksample 57
4.4.3 Cellar 58
4.4.4 Discussion 65
4.5 Contributions, Related Work and Outlook 66

5 Relevance in Dimensionality Reduction **69**
5.1 Domain Features and Complexity 70
5.2 Relevance Estimation 71
5.2.1 Feature Values 72
5.2.2 Feature Relevance 74
5.2.3 Value Approximation 77
5.3 Dimensionality Reduction 78
5.3.1 Bounds for Feature Activation 80
5.3.2 Estimation Errors 81
5.4 Relevance-based Online Planning 82

5.5 Results 84
5.5.1 Cellar 86
5.5.2 Big Foot 89
5.5.3 Runtimes 89
5.5.4 Discussion 92
5.6 Contributions, Related Work and Outlook 94

6 Planning and Acting Under Uncertainty **97**
6.1 Interleaving Planning and Acting 97
6.2 Planning with Delayed Action Results 100
6.3 Challenges and Outlook 103

7 Conclusions **107**
7.1 The Role of Relevance in Planning and Acting 107
7.2 POMDP Planning Onboard Robots 109
7.3 Future Directions 110

Bibliography **113**

List of Publications

Saborío, J. C. and Hertzberg, J. (2019a). Efficient planning under uncertainty with incremental refinement. In *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019, Tel Aviv, Israel, July 22-25, 2019*, page 112

Saborío, J. C. and Hertzberg, J. (2019b). Planning under uncertainty through goal-driven action selection. In van den Herik, J. and Rocha, A. P., editors, *Agents and Artificial Intelligence*, pages 182–201, Cham. Springer International Publishing

Saborío, J. C. and Hertzberg, J. (2018). Towards domain-independent biases for action selection in robotic task-planning under uncertainty. In *Proceedings of the 10th International Conference on Agents and Artificial Intelligence - Volume 2: ICAART,*, pages 85–93. INSTICC, SciTePress

Saborío, J. C. and Hertzberg, J. (2017). Practical assumptions for planning under uncertainty. In *Proceedings of the 9th International Conference on Agents and Artificial Intelligence - Volume 2: ICAART,*, pages 497–502. INSTICC, SciTePress

List of Tables

4.1 PGS in the coffee cup domain 47
4.2 Reward shaping in the coffee cup domain 49
4.3 Reward shaping in the coffee cup domain, part 2 50
4.4 Action selection with PGS in the coffee cup domain 52
4.5 Action selection with PGS in the coffee cup domain, part 2 . . 53
4.6 Performance in the Taxi domain 55
4.7 PGS – Runtime comparison in Rocksample 58
4.8 PGS – Runtimes in cellar[5,1,0,4] 63

5.1 Feature-action values in the coffee cup domain 76
5.2 Relevance values in the coffee cup domain 76
5.3 Feature-action values in the coffee cup domain, part 2 76
5.4 Relevance values in the coffee cup domain, part 2 77
5.5 Relevance values in the coffee cup domain (revisited) 79
5.6 Relevance values in the coffee cup domain (revisited), part 2 . 79
5.7 IRE – Returns and runtime Source: (Saborío and Hertzberg, 2019a) 91
5.8 IRE – Runtime comparison with random MCTS 92

List of Figures

2.1 Person-tracking robot 8
2.2 The four stages of MCTS: 1) Selection, 2) Expansion, 3) Simulation/rollout and 4) Backpropagation. 14
2.3 POMCP first performs regular MCTS over a tree of histories, then selects an action a with UCB1 and receives an observation o from the "real world", finally making the resulting history hao the new tree root. 22
2.4 Reinforcement Learning (left) and POMDP planning (right) . 24
2.5 Qualitative comparison of POMDP planning methods 30

3.1 Relevance-based planning 34
3.2 Rocksample[7,8] 38
3.3 Cellar[7, 8, 7, 8] 39
3.4 BigFoot[4,3,2] 41

4.1 The Taxi domain 55
4.2 PGS in the Taxi domain 56
4.3 PGS in Rocksample (medium) 59
4.4 PGS in Rocksample (large) 60
4.5 Minimal Cellar example 62
4.6 PGS in cellar[5,1,0,4] 63
4.7 PGS in Cellar 64

5.1 IRE in Cellar Source: (Saborío and Hertzberg, 2019a) 87
5.2 Cellar[5,2,6,4] 88
5.3 IRE in Big Foot Source: (Saborío and Hertzberg, 2019a) 90

6.1 Overlapped action planning and execution 99

List of Algorithms

2.1 Policy Iteration 12
2.2 Value Iteration 13
2.3 UCT 16
2.4 Partially Observable Monte-Carlo Planning 23
5.1 Online planning & acting with feature relevance 83
5.2 Feature activation in the belief state 84
5.3 Relevance-aware Partially Observable MCTS 85
6.1 Status dependent search 101
6.2 Planning with delayed results: Dual Planners 102
6.3 Planning with delayed results: Multiple Planners 104

List of Acronyms

IRE *Incremental Refinement.*

MCTS *Monte-Carlo Tree Search.*

MDP *Markov Decision Process.*

PBRS *Potential-Based Reward Shaping.*

PGS *Partial Goal Satisfaction.*

POMCP *Partially-Observable Monte-Carlo Planning.*

POMDP *Partially-Observable Markov Decision Process.*

RL *Reinforcement Learning.*

SSP *Stochastic Shortest-Path Problems.*

UCB1 *Upper-Confidence Bound.*

UCT *Upper-Confidence Bound applied to Trees.*

Notation

In addition to the standard MDP and POMDP notation:

- $s \approx \mathcal{B}(h)$: an observable state sampled from the approximate belief state
- $\mathcal{G}$: a generative model (e.g.: POMDP simulator)
- $\mathcal{F}$: the set of features
- $f \in \mathcal{F}$: a feature

- G_+: the set of observable goal-related features
- G_-: the set of observable restrictions
- G_p: the set of partially observable features
- $\mathfrak{p}(s)$: the PGS scoring function
- T_H: the PGS entropy threshold
- $\gamma_{\mathfrak{p}}$: the discount in the (PGS) PBRS function

- $v(f, a)$: the feature-action value
- $V(f)$: the feature relevance function
- γ_f: the discount factor in the feature-action value estimate
- A_t: the set of relevant actions at time t

Chapter 1

Introduction

In this chapter we review the general motivation behind this thesis and provide an intuitive introduction to relevance-based online planning, which constitutes our main contribution. We also list a summary of more specific theoretical and practical contributions, and provide a brief description of the structure of the dissertation.

1.1 Planning Under Uncertainty: From Principles to Applications

Decision making, planning and reasoning about actions and their outcomes is a key area of research across many disciplines, with many different application domains. At the core of planning, however, there are important theoretical principles that ultimately determine the extent and scope, as well as the practical benefit, of these interesting algorithms. Throughout the last couple of decades, much attention has been devoted to the improvement of planning methods particularly in probabilistic scenarios.

As of the writing of this thesis, state of the art algorithms can solve somewhat large stochastic problems with relative efficiency. These problems are, in fact, orders of magnitude more complex than what the previous state of the art could handle. Thanks to several clever approximations and simplifications, automated planners can now solve difficult problems with combinatorial complexity similar to challenging games of chance.

These game-inspired problems, however, are relatively simple compared to the types of problems we, as humans, have to face every single day. In

a home or office environment, for instance, a task as "simple" as pouring coffee in the morning requires the combination of visual information (such as object recognition), localization, semantic information (coffee can be poured into cups), as well as an abstraction derived from all of these interaction opportunities that induces potentially very large sets of actions and their outcomes, not to mention all the possible world configurations. Accounting for all the possible outcomes of a series of actions and identifying those that are not only useful, but most likely to succeed, quickly becomes computationally intractable when we switch from limited game scenarios to practical problem solving. This is perhaps the most important factor limiting the widespread use of state of the art probabilistic planning.

We can then ask a series of questions: "Why is probabilistic planning important for artificial agents and robots?", "What are the limitations of algorithms for planning under uncertainty?" and most importantly "How can planning under uncertainty for agents such as robots be improved?". The answer to the first question is simple: not only is robotics inherently probabilistic (Thrun, 2000), but most problems of interest cannot be represented exactly and fully-observable deterministic models require strict assumptions. If we intend to continue to advance the state of artificial intelligence for autonomous agents, this is a key area to develop. This leads us to the second issue: limitations. In principle, even the most basic algorithms such as value iteration can solve complex problems given enough time. The challenge, if we intend to solve practical problems, is doing so with severe time constraints. For this reason, much effort has been devoted to improving the state of planning under uncertainty but most solutions in the literature rely on human experts and the construction of a partial solution, often provided to the agent in the form of a knowledge representation, heuristics, explicit action abstractions, explicit state dependencies, and more. Ultimately the planning agent performs only a fraction of the work, and will continue to depend on an external party if the domain or other conditions change. This thesis was motivated by attempting to address the open issue posed by the third question: in order to reduce the aforementioned limitations, we adopted a relevance-based or relevance-aware approach to online planning that grants the agent a degree of autonomy to derive its own internal simplification criteria, instead of depending on prior information.

Due to its widespread use and expressiveness we use the Partially Observable Markov Decision Process (POMDP) as a model of the domain dynamics, which will be explained in detail in the next chapter. In order to facilitate the integration of action outcomes (or observations in POMDPs), we focus on planning online, meaning action planning and execution are interleaved. Relevance-based online planning, therefore, attempts to simplify a POMDP

online by exploiting the problem structure and information obtained from actions and their outcomes.

The formal understanding of *relevance* was originally inspired by how we, as humans, approach practical problem solving. POMDP planning algorithms have mathematical properties that guarantee an optimal or semi-optimal solution by balancing what is commonly called *exploration* and *exploitation*. This systematic approach to problem solving is necessary when there is little or no information available, or when a complex problem has already been reduced to a minimal representation comprised of only relevant elements (as is the case in games, regardless of their complexity). In practice, however, agents in a problem solving setting have numerous interactions and objects at their disposal and the balanced systematic approach becomes extremely slow. Despite these complications, a human problem solver can easily find practical solutions such as standing on an object to reach another, or committing to a difficult action in order to receive an even larger (but delayed) reward. Such an informed problem solver can also quickly identify and focus on useful actions or objects and disregard the rest, effectively simplifying the underlying decision problem. We borrowed the operating principles behind deliberation in problem solving scenarios and designed a formal approach to relevance estimation for online POMDP planning.

1.2 Summary of Contributions

Throughout the thesis we discuss the theoretical principles and the practical implications of our proposal. In summary, relevance-based planning provides:

1. A rollout policy with implicit preferred actions
2. Sample efficient, goal-driven action selection criteria
3. Implicit subtask decomposition through a goal-driven reward bonus
4. Domain-independent dimensionality reduction
5. Attention to relevant features with numerical guarantees

which have mostly a theoretical background. Our experiments allow us to conclude that relevance-based planning also offers:

1. Better scalability than heuristics and random selection in large and complex domains

2. Runtime performance competitive with a random policy
3. Minimal dependence on prior problem analysis
4. Independence from detailed domain knowledge such as heuristics, action hierarchies or state factoring dependencies
5. Increased autonomy in decision-making, since the simplification criteria is developed internally by the planning agent

1.3 Document Structure

This thesis can be subdivided into three parts. The first part is comprised by Chapter 2, which introduces the theoretical background and state of the art, and Chapter 3 which addresses the methodological approach and describes the domains used in the experimental validation. The second part contains Chapters 4 and 5; the former develops the proposed approach for goal-based action selection, while the latter explains relevance-based planning and dimensionality reduction in detail. The last part consists of Chapter 6, in which we offer an analysis and algorithmic proposal for the integration of POMDP planning and acting, and the final chapter in which we conclude the thesis with a discussion of relevance as a mechanism for efficient planning as well as its potential applications.

Chapter 2

Background

Bibliographic review and formal definitions

Planning and reasoning under uncertainty is a fundamental area of research at the intersection of fields such as artificial intelligence, cognitive science and robotics. It is no surprise that numerous approaches and different directions exist to model this process and replicate it in artificial agents. From a "classical" planning perspective (where a number of simplifying assumptions are valid), there are for example logical models that rely on possible world semantics (Thiébaux and Hertzberg, 1992; Kushmerick et al., 1994). Other approaches, such as Markov models, represent the environment using discrete dynamical systems with transitions between information states that follow probability distributions. This background review focuses on algorithms for planning on Markov models, which are widely used to represent planning problems across a variety of domains.

This chapter summarizes the state of the art in probabilistic planning with Markov processes, reviews the relevant literature and provides the mathematical definitions that support the core concepts and algorithms used throughout the dissertation. Analysis of relevant related work is included, as are examples of successful systems that rely on similar principles.

2.1 Probabilistic Planning in Context

Within the field of artificial intelligence, we understand planning as the process of deliberately reasoning about and choosing actions that help an agent

achieve its goals (Ghallab et al., 2016). This behavior is therefore goal-directed, instead of arbitrary or instinctual, and these actions may modify the environment or the agent's perception of it (for instance, by gathering information). Actions are an abstraction of an agent's abilities modulated or determined by its representation of the world. As such, actions can vary depending on what level of abstraction is required: "lower level" actions may be related to operating sensors and motor control, as is the case in navigational planning, whereas "higher level" actions may refer to complex tasks, behavior and indirect interactions with the environment. For example, in the first case a robot might continuously receive data from a sensor (such as a laser range scanner) and choose one of several control options, such as moving forward at a specific linear and angular velocity. In the second case the robot also receives information from sensors, but its available actions might include interacting with specific objects or leaving the room. For the rest of this document we assume "higher level" actions, given in terms of *commands* independent of their implementation. These commands or tasks are the object of deliberation and provide a modeling framework rich enough to represent and solve practical problems, such as serving a cup of coffee or finding a set of keys.

Probabilities come into play when we lift the assumption that a robot or planning agent has complete knowledge of its environment. This lack of information constitutes degrees of uncertainty that induce a probabilistic model. Sources of uncertainty include the outcome of actions, in the case of non-deterministic problems, and noisy or erroneous information from sensors, in the case of partial observability. Despite the many assumptions one could make in order to model a robot's planning domain as a fully observable, deterministic domain, the underlying problem is probabilistic in nature. Well understood problems can of course be solved very efficiently with a variety of *classical* planning methods, but in most (if not all) non-controlled scenarios there will be moments when exploration is necessary to solve the most basic question in planning: "what do I do next?", making reasoning under uncertainty an unavoidable necessity.

2.2 Sequential Decision Problems

Research in sequential decision problems attempts to understand the process of making choices under uncertainty, when each choice and its outcome may lead to another decision problem. These problems are commonly modeled as a *Markov Decision Process* (MDP), with roots in statistics and optimal control (see (Bertsekas, 2005)). This section introduces MDPs from the per-

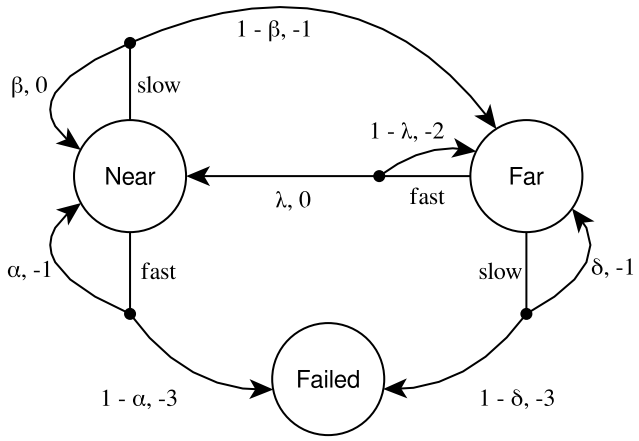
spective and interests of planning, although they are closely related to autonomous learning. For an in-depth review see (Sutton and Barto, 2018) and (Ghallab et al., 2016).

MDPs model not only the problem but also the results of an agent's interactions. The domain is modeled using states and actions, which implicitly create a graph-like structure that represents the environment dynamics as state-action transitions. States are the information available to an agent at any given point in time, and they represent everything and anything required by an agent to achieve its goals. In a given problem or planning domain, all available configurations constitute the state space, which in some cases can grow quickly and easily become intractable. Actions are opportunities available to an agent that allow it to interact with its surroundings, either directly (eg.: by making small changes) or indirectly (eg.: by gathering information). Additionally, actions are non-deterministic and incur in some form of cost (or conversely yield some form of reward). This allows an agent to numerically evaluate the outcome and benefit of actions at some point in time and construct an appropriate plan, or sequence of actions that achieves the specified goals while meeting some given constraints, such as length, cost or time. We can define an MDP as the tuple $\langle S, A, T, R \rangle$, where:

- S is a finite set of states
- A is a finite set of actions
- $T(s, a, s') = p(s'|s, a)$ is the state transition function
- $R(s, a, s')$ is the real-valued reward function

Markov processes assume the Markov property: functions are defined in terms of the current state, s, and a resulting state s' upon executing an action a. This means all decisions are made exclusively with the information provided by s, regardless (in principle) of the prior states. States are however not restricted to exclusively representing *immediate* information, and can instead synthesize any form of previous information that is relevant for decision-making. We can therefore understand *state* as the information required by an agent to make a decision.

Example 2.1. *Consider the MDP in figure 2.1, where a robot must follow a person closely from a emphsafe distance. The only two actions are driving fast and slowly, and there are three possible states: the robot can be near the person (within a safe distance), far from the person, or it can fail its task. Failure occurs when the robot gets too close or too far from the person, in*



(a) MDP

s	a	s'	$p(s, a, s')$	$r(s, a, s')$
Near	fast	Near	α	-1
Near	fast	Failed	$1 - \alpha$	-3
Near	slow	Near	β	0
Near	slow	Far	$1 - \beta$	-1
Far	fast	Near	λ	-1
Far	fast	Far	$1 - \lambda$	-2
Far	slow	Far	δ	-1
Far	slow	Far	$1 - \delta$	-3

(b) Transition table

Figure 2.1: Person-tracking robot

which case the task is finalized or restarted. For each transition, the two values represent the probability (left) and the immediate reward (right).

The reward distribution implicitly defines the robot's goal. Being so small, it is relatively intuitive to see that this particular a solution for MDP achieves maximal reward when the robot remains on state Near *as long as possible. The rewards per action also represent an aspect of the domain dynamics: in this case, driving fast incurs in a negative reward in every state. In the case of failure, the robot receives an even heavier punishment. Uncertainty in this problem is the result of transition probabilities (see fig. 2.1b): for example, driving fast when on state* Near *succeeds with probability* α *(robot remains in safe proximity), but with probability* $1 - \alpha$ *it gets too close and the task*

fails. Likewise, driving slowly can help the robot remain in the safe area with probability β, but it can make the robot fall behind and into the Far *state with probability $1-\beta$. Driving slowly when the robot is far from the person is an inexpensive action but it can lead to failure with probability $1-\delta$, and with probability λ the robot can return to the safe distance if it drives fast.*

Given information about the transition probabilities and the immediate rewards, a solution to an MDP is given by solving an optimization problem: at any given state, select the action with the highest expected value, given as a combination of its immediate probability and reward as well as the probability and reward of the best possible outcome thereafter. This is formally discussed in the following subsection, which provides a brief formal definition of finite MDPs, their value functions and their properties.

2.2.1 Markov Decision Processes

For the following we adopt the notation used in (Sutton and Barto, 2018). We'll summarize their MDP definition using the most relevant equations, but we'll skip their full derivation. We begin by defining the transition and reward functions. The transition probability is defined as:

$$\begin{aligned} p(s'|s,a) &= P\{S_{t+1}=s'|S_t=s,A_t=a\} \\ &= \sum_{r\in R} p(s',r|s,a) \end{aligned} \tag{2.2.1}$$

and the immediate reward of this transition:

$$\begin{aligned} r(s,a,s') &= \mathbb{E}\left[R_{t+1}|S_t=s,A_t=a,S_{t+1}=s'\right] \\ &= \frac{\sum_{r\in R} rp(s',r|s,a)}{p(s'|s,a)} \end{aligned} \tag{2.2.2}$$

In order to make informed choices about the long-term benefits of actions, it is common to use long planning horizons instead of maximizing immediate rewards. The return is the sum of rewards after a number of steps, $G_t = R_{t+1} + R_{t+2} + \ldots + R_T$. Often future rewards are discounted to better reflect the impact of near and distant rewards. The expected discounted return is therefore:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T \\ &= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \end{aligned} \tag{2.2.3}$$

where $0 \leqslant \gamma \leqslant 1$ is the discount factor.

Planning on MDPs involves estimating values of states and actions, that represent the benefit of reaching a state or executing some given action. This benefit is defined in terms of the expected return and policy. A policy is a function mapping each state $s \in S$ and action $a \in A$ to the probability of executing a when the agent is in state s. The value function $v_\pi(s)$ denotes the expected return from initial state s following policy π, and is defined as:

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t | S_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s \right] \tag{2.2.4}$$

which corresponds to the state-value function. The expected return of executing action a in state s and then following policy π is denoted by the value function $q_\pi(s, a)$, defined as:

$$q_\pi(s, a) = \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a \right] \tag{2.2.5}$$

and is called the action-value function. Given an initial state, an optimal plan is one that achieves the largest possible return. That is, for every state, it satisfies:

$$v_*(s) = \max_\pi v_\pi(s) \tag{2.2.6}$$

where v_* is the optimal value function. Similarly the optimal plan also satisfies:

$$q_*(s, a) = \max_\pi q_\pi(s, a) \tag{2.2.7}$$

where q_* is the optimal action-value function. These two functions satisfy a number of consistency conditions summarized in the form of a Bellman optimality equation, that expresses the relationship between the value function and the optimal action-value function:

$$\begin{aligned} v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s, a) \\ &= \max_{a \in \mathcal{A}(s)} \sum_{s', r} p(s', r | s, a) \left[r + \gamma v_*(s') \right] \end{aligned} \tag{2.2.8}$$

and in the case of of q_*:

$$\begin{aligned} q_*(s,a) &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') | S_t = s, A_t = a\right] \\ &= \sum_{s',r} p(s', r|s, a)\left[r + \gamma \max_{a'} q_*(s', a')\right] \end{aligned} \tag{2.2.9}$$

For each state in an MDP there is one such equation. If the probabilities are known, solving an MDP reduces to obtaining the corresponding values and recommending the optimal action for each state. In practice, however, it is only necessary to find optimal actions for states reachable by the agent, not for every single state. This is the purpose of a planner: mapping states in $\mathcal{S} \subset S$ to actions in A through the optimal partial policy (or plan) $\pi_*(s, a)$ starting from initial state s_0. The following subsections summarize MDP planning algorithms with special attention to MCTS.

2.2.2 Dynamic Programming on MDPs

There are a number of algorithms that either directly or indirectly compute the aforementioned value functions, and converge to the optimal policy that, in essence, must simply satisfy $\forall s \in S\ \pi_*(s) = \arg\max_a q(s, a)$. The two most basic algorithms, policy iteration and value iteration, are dynamic programming (DP) techniques that exploit the perfect information available in an MDP. From now on we adopt the notation $V(s)$ to refer to the approximation of the true state-value function $v(s)$, and $Q(s, a)$ to refer to $q(s, a)$, the true action-value function.

Policy iteration is a powerful method that consists of two stages: policy evaluation and policy improvement. The evaluation stage updates the current $V(s)$ estimate in small increments, using the estimates of $V(s')$ for all states immediately reachable from actions in s, in what we will call the DP-update:

$$V(s) = \sum_{s',r} p(s', r|s.a)\left[r + \gamma V(s')\right] \tag{2.2.10}$$

The policy improvement step updates π by assigning, for each state, the action with the current best value. Once no actions in π need to be changed, the policy is stable and $\pi \approx \pi_*$, the optimal policy. Algorithm 2.1 shows how iterative policy iteration approximates the optimal policy, with error ϵ.

These two steps, evaluation and improvement, can be combined in one algorithm using the Bellman equation as an update (eq. 2.2.8), resulting in value iteration. Like policy iteration it evaluates the entire state space but it takes the maximum value on each update instead. Algorithm 2.2 shows how a policy can be approximated with this method.

Algorithm 2.1 Policy Iteration

```
1: procedure EVALUATION
2:     repeat
3:         for s ∈ S do
4:             v(s) ← V(s)
5:             a ← π(s)
6:             V(s) ← DP-update
7:         end for
8:     until max_s |v(s) − V(s)| < ε
9:     IMPROVEMENT()
10: end procedure

11: procedure IMPROVEMENT
12:     stable ← true
13:     for s ∈ S do
14:         a_π ← π(s)
15:         a ← arg max_a DP-update
16:         π(s) ← a
17:         if a_π ≠ a then
18:             stable ← false
19:         end if
20:     end for
21:     if ¬stable then
22:         EVALUATION()
23:     end if
24:     return π
25: end procedure
```

In both cases, an ϵ-optimal policy can be found in a finite number of iterations, but these methods don't scale well to large domains due to their dependency on exhaustively traversing the entire state space. Their core element, namely the one-step value update, forms the basis of many planning algorithms.

Building on the convergence properties of DP, some methods introduce heuristics to improve problem tractability by performing more careful DP updates. Real-Time Dynamic Programming (RTDP) initializes some state values heuristically and through a number of trials, performs DP updates on a select sequence of states obtained by a greedy action selection policy (Barto et al., 1995). Labeled RTDP marks states as *solved* as soon as they meet the convergence criteria (value estimate below some error threshold) and only applies DP updates on unsolved states, improving response times (Bonet and Geffner, 2003). Bounded RTDP further improves performance by maintaining lower and upper bounds on the value functions (McMahan et al., 2005).

These methods share many similarities with heuristic search algorithms, some of which are used to solve MDPs by treating them as a graph traversal problems, or *Stochastic Shortest-Path Problems* (SSP). SSPs generalize MDPs as graphs and use probabilistic search algorithms based on extended

Algorithm 2.2 Value Iteration

repeat
  for $s \in S$ **do**
    $v(s) \leftarrow V(s)$
    $V(s) \leftarrow \max_a$ **DP-update**
  end for
until $\max_s |v(s) - V(s)| < \epsilon$
for $s \in S$ **do**
  $\pi(s) \leftarrow \arg\max_a$ **DP-update**
end for
return π

shortest-path graph methods. An SSP is defined as the tuple $\langle M, s_0, S_T \rangle$ where M is an MDP, $s_0 \in S$ an initial state and $S_T \subseteq S$ the set of terminal states. The solution to an SSP is a partial policy for M that guarantees a terminal state $s_T \in S_T$ can be reached from s_0. SSP algorithms include LAO* (Hansen and Zilberstein, 2001), an extension of AO* that allows loops and finds an optimal cyclic subgraph, and its generalization called Symbolic LAO* (Feng and Hansen, 2002).

2.2.3 Monte Carlo Search

Monte Carlo is a family of methods that rely on random sampling to obtain numerical approximations of target functions. A Monte Carlo approach to solving MDPs involves approximating state-value and action-value functions using random sampling, and obtaining averages of these samples. For example, after drawing N samples from the state transition function $T(s, a, s')$ we can approximate $v(s)$ as:

$$v(s) \approx V(s) = \frac{1}{N} \sum_{1}^{N} r + \gamma V(s') \tag{2.2.11}$$

Monte-Carlo Tree Search (MCTS) constructs a search tree with the initial state s_0 as root, where actions and states are progressively added as are their corresponding values. Once *sufficient* statistics are gathered, an action can be chosen following one or another action selection policy. Value estimation is therefore based on averaging the outcome of multiple possible scenarios, randomly obtained from a simulator. MCTS is popular in the game playing community, and has been successfully applied in games such as Tic-tac-toe, Othello and Chess (Abramson, 1987) as well as in the computer program

that defeated a professional Go player (Silver et al., 2016). The algorithm works in four stages commonly called selection, expansion, simulation and backpropagation (illustrated in fig. 2.2).

1. During **selection** states are visited and actions chosen following some policy until a new (unvisited) state is found.

2. The new state and its actions are added to the tree on the **expansion** stage.

3. The **simulation** stage initiates a *rollout*, consisting on a random walk starting from the newly discovered state until a terminal state is found or the maximum acting budget is exhausted.

4. Finally, the return obtained from the rollout is used to initialize the value of the new state, as a result of the **backpropagation** phase.

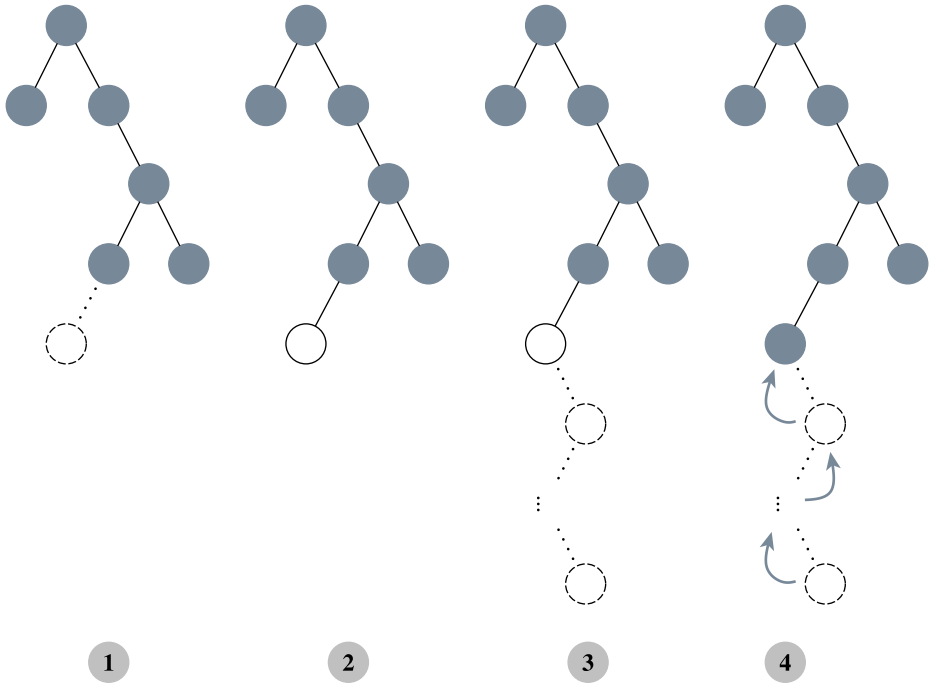


Figure 2.2: The four stages of MCTS: 1) Selection, 2) Expansion, 3) Simulation/rollout and 4) Backpropagation.

Upper Confidence Bound

There are many possible choices of action selection and rollout policies in Monte Carlo search, with different effects in performance. Within the context of bandit problems (non-sequential decision problems), (Auer et al., 2002) showed that it is possible to balance exploration and reward exploitation through a formula called *Upper-Confidence Bound* (UCB1). UCB1 is based on identifying an upper bound for action value estimates that minimizes regret, derived from Hoeffding's inequality (Hoeffding, 1963). Let X be a set of n independent random variables such that $\forall i.0 \leqslant X_i \leqslant 1$ and let $\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$ be the empirical mean. Then

$$P(\mathbb{E}(X) > \bar{X} + t) \leqslant e^{-2nt^2} \tag{2.2.12}$$

for $t > 0$. Now let $\mathcal{U}(s,a)$ be an upper bound on $Q(s,a)$ such that

$$q(s,a) \leqslant Q(s,a) + \mathcal{U}(s,a) \tag{2.2.13}$$

so we get the inequality:

$$P(q(s,a) > Q(s,a) + \mathcal{U}(s,a)) \leqslant \exp\{-2n(s,a)\mathcal{U}(s,a)^2\} \tag{2.2.14}$$

and for a given probability $p = \exp\{-2n(s,a)\mathcal{U}(s,a)^2\}$:

$$\begin{aligned} \ln p &= -2n(s,a)\mathcal{U}(s,a)^2 \\ \mathcal{U}(s,a) &= \sqrt{-\frac{\ln p}{2n(s,a)}} \end{aligned} \tag{2.2.15}$$

Using the bound found by (Auer et al., 2002), let $p = n^{-4}$ which yields the upper confidence bound:

$$\mathcal{U}(s,a) = \sqrt{\frac{2\ln n}{n(s,a)}} \tag{2.2.16}$$

so eq. 2.2.13 becomes:

$$q(s,a) \leqslant Q(s,a) + \sqrt{\frac{2\ln n}{n(s,a)}}$$

The resulting UCB1 action selection policy is:

$$a \leftarrow \arg\max_a Q(s,a) + c\sqrt{\frac{2\ln N(s)}{n(s,a)}} \tag{2.2.17}$$

where c is an exploration constant, $N(s)$ is the number of times s has been visited and $n(s, a)$ the number of times a has been chosen in s.

UCB in Tree Search

UCB1 forms the basis of *Upper-Confidence Bound applied to Trees* (UCT), a Monte-Carlo search algorithm that applies this formula sequentially as an action selection policy to solve MDPs (Kocsis and Szepesvári, 2006). The idea of using a generative model or MDP simulator to solve large MDPs had been explored before (eg.: (Kearns et al., 2002)), but UCT became the modern standard for MCTS possibly due to its simplicity and convergence properties. The basic outline is shown in algorithm 2.3, where $\mathcal{G}$ is a generative model (or MDP simulator).

Algorithm 2.3 UCT

```
1: function SEARCH(s)
2:     repeat
3:         SIMULATE(s, 0)
4:     until timeout
5:     return arg max_a Q(s, a)
6: end function

7: function ROLLOUT(s, d)
8:     if d > d_MAX ∨ s is terminal then
9:         return 0
10:    end if
11:    a ← π_rollout(s)
12:    s', r ← G(s, a)
13:    return r + γROLLOUT(s', d + 1)
14: end function

15: function SIMULATE(s, d)
16:    if d > d_MAX ∨ s is terminal then
17:        return 0
18:    end if
19:    if s ∉ Tree then
20:        Add and initialize s
21:        return ROLLOUT(s, d)
22:    end if
23:    a ← UCB1(s, a)
24:    s', r ← G(s, a)
25:    R ← r + γSIMULATE(s', d+1)
26:    Update visit counts N(s) and N(s, a)
27:    Update value with R
28:    return R
29: end function
```

Given enough time, UCT can approximate arbitrarily good policies for a given MDP. Using UCB1, it explores possible actions at first but converges to ϵ-optimal actions eventually, while performing multiple value backups over a potentially long planning horizon or search depth d_{MAX}. The sequential application of UCB1 results in the failure probability (at the root node)

converging to zero at a polynomial rate, in both discounted and undiscounted MDPs.

This algorithm is particularly useful due to it constituting an *anytime* and *online* MDP planner. Anytime planners can be interrupted when necessary and produce the best available answer subject to their (possibly limited) current estimation. Online planning refers to the ability to interleave value estimation with action execution rather than solving a problem in its entirety before acting. MCTS simulates possible transitions starting from the agent's current state so it can, in principle, respond to changes in the environment and the various outcomes of actions.

UCT works well for MDPs because simulating immediate state transitions is (computationally) inexpensive. The quality of Monte Carlo approximation, however, relies heavily on sampling so in the general case many simulations are required to produce reasonable behavior in planning agents. This complexity, like that of any other planning algorithm, grows with respect to the number of states and their (action) branching factor.

As stated above MDPs represent fully observable planning domains, which is often insufficient to accurately represent decision-making onboard agents with sensors. In the following section we will discuss the generalization of MDPs that removes this restriction, yielding an even richer family of models that more accurately represent complex planning and decision-making problems.

2.3 Decision Problems with Partial Observability

Despite their many fields of application, MDPs have limited representational power when it comes to modeling real-world problems. The underlying assumptions that the agent always knows its true state and that the outcome of actions is observed accurately (albeit *a posteriori*) don't translate well to how agents (artificial or not) interact with their environment. If we eliminate these restrictions, we introduce what is commonly known as "partial observability". This means state transitions are still governed by probability distributions, that may or may not be known in advance, but in addition to estimating value functions the agent must estimate its current state based on observations received upon executing actions. This generalization, modeled with a *Partially-Observable Markov Decision Process* (POMDP), provides a rich, formal framework to model planning under uncertainty when the agent must also consider the effect of information gathering actions.

In this section we will introduce the notation and basic concepts related to POMDPs, and review a number of POMDP planning algorithms.

2.3.1 Partially Observable Markov Decision Processes

Following the state and action nomenclature of MDPs, POMDPs can be defined as an extension such that:

- S is a finite set of states
- A is a finite set of actions
- $T(s, a, s') = p(s'|s, a)$ is the state transition probability
- $R(s, a, s')$ is the real-valued reward function
- Ω is a finite set of observations
- $O(s, a, \omega) = p(\omega|s, a)$ is the probability of receiving observation $\omega \in \Omega$ upon executing a in s

represented by the tuple $\langle S, A, T, R, O, \Omega \rangle$. Given than states are not directly observable in a POMDP, an agent must also maintain a probability distribution over the set of states, known as the belief state, and begins planning from an initial belief b_0 where $\forall s \in S\ b_0(s) = Pr(s_0 = s)$.

Action selection can be defined in terms of the sequence of previously executed actions and their observations, known as the *history* and defined as $h_t = \{a_0, \omega_1, \ldots, a_{t-1}, \omega_t, \}$ at time t. In practice this is cumbersome to maintain so instead it is represented as a belief distribution (Åström, 1965):

$$b_t(s) = Pr\{s_t = s|h_t\} \tag{2.3.1}$$

This allows us to express the new belief $b'(s)$ in terms of the previous belief, previous action and current observation:

$$\begin{aligned} b'(s) &= \tau(b, a, \omega) \\ &= \frac{\sum_{s'} O(s', a, \omega) T(s, a, s') b(s')}{\sum_s \sum_{s'} O(s', a, \omega) T(s, a, s') b(s')} \end{aligned} \tag{2.3.2}$$

which is equivalent to the Bayes filter and simplifies belief computation by using only the information available at the previous time step. The belief update can be computed over many states; finding an optimal policy, on the other hand, is the challenge of POMDP planning. Unlike in MDPs, policies are given in terms of beliefs, such that $\forall b \in \mathcal{B} \exists a \in A \pi(b, a)$, where $\mathcal{B}$ is the

belief space. The optimal policy is analogous to the MDP formulation in that it selects the action with maximal discounted return, but it is defined over a $|\mathcal{S}|$-dimensional space that makes most direct solutions intractable. This particular issue is known as the *curse of dimensionality*. Additionally, POMDPs also carry the *curse of history*, a product of the large number of action and observation sequences that arise especially in large domains.

Value functions in POMDPs are similar to those in MDPs, but they refer to beliefs and observations instead. The state-value function in a POMDP is analogous to eq. 2.2.4:

$$v_\pi(b) = \mathbb{E}_\pi \left[G_t | b_t = b \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| b_t = b \right] \tag{2.3.3}$$

with v_*, the optimal state-value function, also analogous to 2.2.6:

$$\begin{aligned} v_*(b) &= \max_a q_\pi(b, a) \\ &= \max_a \left[R_{t+1} + \gamma v_*(b') | b_t = b, A_t = a, \omega_t = \omega \right] \\ &= \max_a \left[\sum_s b(s) R(s, a, s') + \gamma \sum_{\omega \in \Omega} O(s, a, \omega) v_*(b') \right] \end{aligned} \tag{2.3.4}$$

One of the most well known value iteration methods for POMDPs approximates this value function by using a set of vectors $\Gamma_t = \{\alpha_0, \ldots, \alpha_m\}$, $\forall 0 \leqslant i \leqslant m . a_i \in A$, where each α-vector corresponds to a $|S|$-dimensional hyperplane (Smallwood and Sondik, 1973). The solution set Γ is computed by generating and then combining α-vectors for each action and observation, which contain the value estimate at horizon t. Finally the value function is obtained from the region defined by the α-vectors:

$$v_t(b) = \max_{\alpha \in \Gamma_t} \sum_{s \in S} \alpha(s) b(s) \tag{2.3.5}$$

This work also showed that the value function is piecewise-linear and convex (PWLC) and can thus be approximated by a PWLC function. Such a direct approach, however, is impractical for anything other than the smallest of examples. For this reason, most literature on POMDP planning focuses on methods that quickly approximate the value function, without introducing a significant error compared to value iteration. We will now review select POMDP planning methods from different families, with special emphasis on the class used throughout the dissertation.

2.3.2 POMDP Algorithms

Direct improvements over exact methods (VI) include the algorithms known as Witness and Incremental Pruning. Witness constructs an optimal solution by selecting optimal α-vectors and discarding those that are suboptimal, based on evidence provided by some *witness* point (Cassandra et al., 1994). Incremental Pruning is an efficient version of the direct VI approach of (Smallwood and Sondik, 1973), which prunes suboptimal vectors while computing the value function (Cassandra et al., 1997).

A POMDP can also be represented as an equivalent belief-MDP, and many MDP techniques can be used to solve it. However, it should be fairly obvious that the resulting MDP is exponentially larger, equivalent to a $|S|$-dimensional belief simplex. Grid-based methods approximate this large state space by constructing a finite grid over the simplex, so that only the values contained in the grid are updated and the rest are interpolated. Such approaches include splitting the simplex in equal parts through a fixed-sized grid (Lovejoy, 1991), uneven spacing of points through a non-regular grid (Hauskrecht, 2000), and combining the interpolation benefits of both using a variable resolution grid (Zhou and Hansen, 2001; Bonet, 2002).

Beliefs can also be compressed into a lower dimensional structure, speeding up the approximation of value functions. Efforts in this line of work include an optimization approach that finds linear lossy compressions (Poupart and Boutilier, 2002) and non-negative matrix factorization (Li et al., 2007; Theocharous and Mahadevan, 2010).

Point-based algorithms constitute a significant step towards the tractability of large POMDPs. These methods reduce the complexity of planning by updating only a few select belief states, and maintaining no more than one α-vector for each. For example, Heuristic Search Value Iteration (HSVI) utilizes observation heuristics derived from upper and lower bounds of the value function to avoid unreachable parts of the belief space (Smith and Simmons, 2004). Point Based Value Iteration (PBVI) constitutes a family of algorithms that interleave value backups with carefully chosen update points, following different approaches that range from random belief point selection to error bound estimation (Pineau et al., 2003b, 2006). Perseus expands this idea by introducing more efficient backups that improve the value of multiple points in the belief set and approximating the value function faster, with fewer vectors (Spaan and Vlassis, 2005). One of the fastest point-based algorithms is SARSOP, which uses heuristic exploration to approximate the set of reachable belief points, but avoids sampling in suboptimal regions (Kurniawati et al., 2008). Both PBVI and SARSOP were tested in robotic applications, but in practice they are restricted very small POMDPs that fail to represent

realistic planning scenarios. These limitations are due, at least in part, to both of them being full-width, offline algorithms.

Consequently, online algorithms have been designed to focus on the current belief state. The heuristic search approach to online planning first computes an approximation of the value function offline, in order to obtain a heuristic for planning from the current belief state. The POMDP is normally seen as an AND-OR graph where belief states are OR-nodes and actions are AND-nodes. For example, Anytime Error Minimization Search (AEMS) searches over reachable beliefs and expands the fringe nodes with the highest expected error with respect to the current belief state (Ross and Chaib-draa, 2007).

Other online POMDP planning algorithms include extensions of the aforementioned offline planning methods, but the most successful modern approach for large POMDPs is online planning based on MCTS. Partially observable MCTS has successfully scaled to relatively large POMDPs and has spawned various derived algorithms, and is addressed in more detail in the following subsection.

2.3.3 Partially Observable Monte Carlo Search

Similar to its immediate predecessors, Monte Carlo search for POMDPs constitutes an anytime, online planning algorithm that, to some extent, also exploits principles from fully observable MDPs. Just like UCT, an online POMDP planner relies on a generative model (in this case a POMDP simulator) and value functions are computed by averaging the value of states and actions sampled from their respective probability distributions. The main challenge is representing belief states and handling the large spaces generated when observations are added to the search tree. As mentioned before, POMDPs carry the *curse of dimensionality* and the *curse of history*. *Partially-Observable Monte-Carlo Planning* (POMCP) is a POMDP planning algorithm that attempts to solve these issues by expanding UCT in two meaningful ways (Silver and Veness, 2010):

1. Instead of a tree of states, the algorithm maintains a tree of histories. Once an action has been executed and an observation is received, it discards every other history (they are at this point impossible), pruning the rest of the search tree and maintaining only the subtree rooted at the current history.

2. The belief state is approximated by an unweighted particle filter. Particles (states) are added as necessary and states (with partially observable elements) are sampled and used as the starting point for planning.

Particle filtering is a popular technique to efficiently approximate probability distributions, commonly used in AI and robotics (Fox et al., 1999).

The result is a modified UCT algorithm that plans over histories with a belief state approximator. This process is illustrated in fig. 2.3.

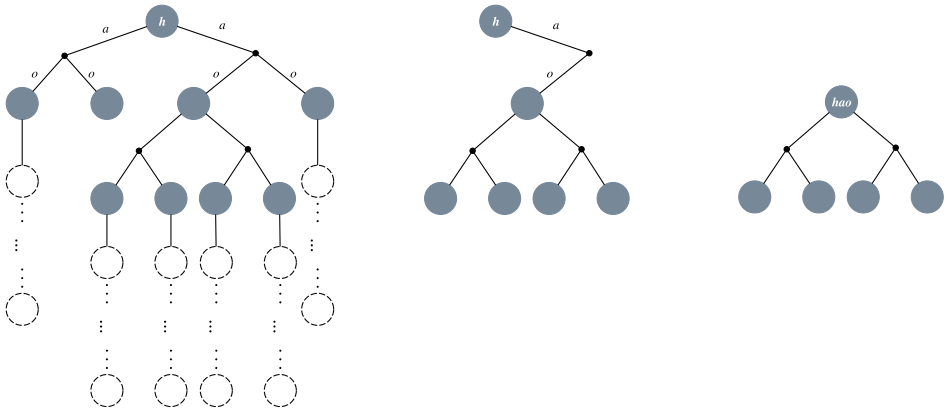


Figure 2.3: POMCP first performs regular MCTS over a tree of histories, then selects an action a with UCB1 and receives an observation o from the "real world", finally making the resulting history hao the new tree root.

Algorithm 2.4 shows the complete pseudocode, where $\mathcal{G}$ is a generative model or POMDP simulator. At the time of writing and to the best of our knowledge, POMCP is the most general method that is efficient enough to solve large POMDPs, and constitutes the foundation of what today is an entire family of sampling-based or generative POMDP algorithms (and by extension, PO-MCTS). Despite its potential in all levels of planning in mobile robotics, substantial improvements to POMCP is still an open area of research.

Key open areas in POMDP planning include better sampling efficiency and higher quality rollout policies, the latter of which have a large impact in overall performance. POMCP's best results, in games such as Battleship and Pac-Man, are dependent on heuristics encoded as preferred actions in the rollout policy. These heuristics are of course not transferable across domains and require a prior analysis of the problem. In addition, most applications and problems in practical problem-solving domains tend to be more complicated than games, even if they have similarly sized state-spaces. For example, games normally contain only the minimum and necessary objects and actions, whereas a mobile robot might encounter any number of objects in the world with little or no connection to its goal.

Algorithm 2.4 Partially Observable Monte-Carlo Planning

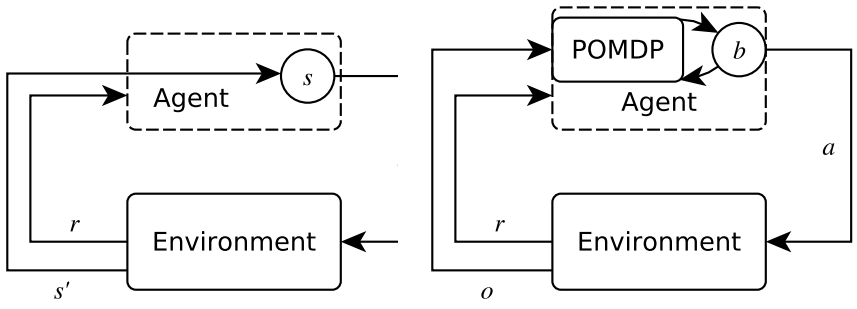
```
1: function SEARCH(h)
2:     repeat
3:         s ≈ B(h)
4:         SIMULATE(s, h, 0)
5:     until timeout
6:     return arg max_a Q(h, a)
7: end function

8: function ROLLOUT(s, h, d)
9:     if d > d_MAX ∨ s is terminal then
10:        return 0
11:    end if
12:    a ← π_rollout(h)
13:    s', o, r ← G(s, a)
14:    return r + γROLLOUT(s', hao, d + 1)
15: end function

16: function SIMULATE(s, h, d)
17:    if d > d_MAX ∨ s is terminal then
18:        return 0
19:    end if
20:    if h ∉ Tree then
21:        Add and initialize h
22:        return ROLLOUT(s, h, d)
23:    end if
24:    a ← UCB1(h, a)
25:    s', o, r ← G(s, a)
26:    R ← r + γSIMULATE(s', hao, d + 1)
27:    B(h) ← B(h) ∪ {s}
28:    Update visit counts N(h) and N(h, a)
29:    Update value with R
30:    return R
31: end function
```

DESPOT is another PO-MCTS algorithm which uses heuristics to expand a tree of beliefs and prefer those that are reachable, resulting in better worst-case performance than POMCP, but similar average performance (Somani et al., 2013). A recent modification called DESPOT-α attempts to address the issue of sampling from very large observation spaces (Garg et al., 2019), and like GPS-ABT (Seiler et al., 2015) and POMCPOW (Sunberg and Kochenderfer, 2018) (also PO-MCTS), it has applications in domains with continuous action and observation spaces such as motion and navigational planning. The ideas behind POMCP also have influence outside of MCTS: for example, POMHDP is a recent anytime POMDP planning algorithm based on RTDP that can use multiple heuristics and relies on a particle approximation of the belief state as well as a generative model (Kim et al., 2019). It should be noted that all of these algorithms rely on domain knowledge and heuristics.

In the following section we will discuss the connection between planning and learning methods, which differ mainly in whether the domain dynamics are known or available in advance or not. We will then address techniques



(a) (b)

Figure 2.4: Reinforcement Learning (left) and POMDP planning (right)

to improve the efficiency of planning and learning in large domains, many of which can be applied in both fully and partially observable MDPs.

2.4 Planning and Learning

The planning methods presented here assume the transition dynamics of the domain are already known, in the form of an MDP or a POMDP. For this reason, planning focuses on the deliberation stage of problem solving. A different family of problems have unknown domain dynamics so the agent also has to discover and learn a model, either before or during the deliberation stage. Such problems can be solved using one of many *Reinforcement Learning* (RL) techniques, in which the transition probabilities and value functions are learned through systematic interaction with the domain and the result is the learned (PO)MDP as well as its optimal policy. One of the most well regarded sources is (Sutton and Barto, 2018).

RL is generally defined with respect to the type of problems, and not with respect to specific methods. Intuitively it is very simple: RL represents an agent learning exclusively from its own experience, by trial and error (see fig. 2.4a): the decision-making agent knows only its current state, chooses the best available action, perceives a reward, and transitions to a new state.

In principle RL algorithms are not very different from MDP planning algorithms: both select an action that maximizes return and update the value function approximation based on the perceived reward (using a dynamic programming update). Unlike in planning problems (fig. 2.4b), a transition model is not (normally) available so values can only be approximated based on the immediate outcome of transitions. A key concept in RL is *temporal*

difference (TD) learning, which approximates state and action values by *bootstrapping* based on the difference between DP-updates for the current and previous states. TD(0), for example, uses the update rule:

$$V(s) \leftarrow V(s) + \alpha \left[r + \gamma V(s') - V(s) \right] \tag{2.4.1}$$

where $0 \leqslant \alpha \leqslant 1$ is the learning rate and γ the discount factor. More sophisticated TD algorithms include Q-Learning (Watkins, 1989), SARSA (Rummery and Niranjan, 1994), least-squares TD (Bradtke and Barto, 1996) and gradient-based TD (Sutton et al., 2008). TD(λ) is one of the most popular TD learning algorithms: it extends the range of DP updates to include multiple transitions, from one step for TD($\lambda = 0$) to a functional equivalent of Monte Carlo learning for TD($\lambda = 1$). The parameter λ refers to the use of an *eligibility trace*, a technique that improves performance by updating recently visited states, often combined with linear function approximation (Singh and Sutton, 1996).

It is possible to combine RL methods with an explicit planning stage, which results in better performance by exploiting the partial model acquired during the learning stage. For instance, Sutton's Dyna architecture introduces a value iteration phase once a number of state transitions is discovered, improving the value function estimate (Sutton, 1990). Randomly selecting states for any sufficiently large MDP is inadvisable however, so instead *prioritized sweeping* can be used to select, in order, which states participate in the planning stage (Moore and Atkeson, 1993; Peng and Williams, 1993).

Together these techniques grant an RL agent some of the advantages of MCTS algorithms, such as UCT. There is an interesting game-playing algorithm called TD-Search, that combines the simulation elements of MCTS with the function approximation and bootstrapping of temporal-difference methods (Silver et al., 2012). There are several well known techniques that attempt to improve the performance of both planning and learning in large domains, which will be discussed in the following section.

2.5 On Performance and Scalability

The performance of planning on MDPs and POMDPs is normally measured in terms of returns, or cumulative discounted reward. For a given problem, there is at least one optimal policy that is guaranteed to be found with exact methods, but this comes (as discussed earlier) at a high computational cost that makes large problems intractable. Larger problems can be solved through approximations and simplifications that speed up planning but also

introduce an approximation error. In this case, performance becomes a measure of how close these approximate solutions are to return the theoretical optimal policy. In this section we briefly review examples of techniques that improve the scalability of planning (and learning) methods, making complex problems more tractable.

The first group of techniques simplify very large problems by clustering or aggregating states following some given formal principle. This way, values are shared and computed for clusters instead of individual states. In large domains, cluster membership can be determined probabilistically (Singh et al., 1995) and the notion can be extended to include non-Markov processes (Hutter, 2016). An extension for POMDPs relies on principles from the Incremental Pruning algorithm (Feng and Hansen, 2004). In a similar direction, state spaces can also be converted to factored representations that use state descriptors with known dependencies (e.g.: using a Bayesian network). This can greatly simplify large problems with limitations: a factored representation is normally generated using specific domain knowledge, and not all problems can be factorized. Algorithms for planning on factored representations include (Strehl et al., 2007) and (Feng and Hansen, 2014). Factoring has also been used to improve the point-based POMDP algorithm SARSOP, resulting in an MDP with "mixed observability" (Ong et al., 2010).

The second group addresses the complexity of value estimation for many states by approximating the value function, often with neural networks, which provide a degree of generalization. There are many successful examples from the game-playing community, where RL and planning methods were combined to solve games such as Backgammon (Tesauro, 1995) and Go (Silver et al., 2016). In line with the current trends, deep and variational reinforcement learning techniques have also been applied in POMDPs (Karkus et al., 2017; Igl et al., 2018). Partial observability introduces a number of additional challenges however, since states cannot be perceived directly and must be approximated from observations. These techniques contribute to generalization and focus on learning the problem structure, but underneath they rely on unsophisticated planning algorithms that struggle in high dimensional domains.

The last group deals with actions and introduces the notion of subtasks or subgoals, either by modifying the reward distribution or by planning with an action hierarchy. Reward shaping allows a planning agent to focus on short-term subgoals by awarding reward bonuses to certain *preferred* actions. The new reward distribution implicitly generates a different MDP, so the policies might not transfer well to the original problem. *Potential-Based Reward Shaping* (PBRS) solves this issue by proposing a shaping function based on the weighted difference between the current and previous states, an impor-

tant result that preserves policy optimality (Ng et al., 1999). PBRS can easily be extended to POMDPs, with a variety of potential functions (Eck et al., 2016). Action hierarchies constitute an explicit abstraction of the action space, therefore inducing also a simpler, abstract state space. They are often represented with a Hierarchical Task Network (HTN), and hierarchical planning algorithms focus on using HTNs efficiently at different levels of abstraction, by planning in an abstract domain and then transferring these actions to the agent's actual action model. In RL and MDP planning, hierarchical planning methods include MAXQ (Dietterich, 2000) and Options (Sutton et al., 1999), and in POMDP planning PolCA (Pineau et al., 2003a), which addresses some considerations for planning onboard robots. In addition, the MAXQ approach has also been combined with POMCP (Vien and Toussaint, 2015). In all of these cases the action hierarchies must be provided in advance, but there are advances that propose how they might also be built automatically (Konidaris, 2016).

In the next section we present some examples of MDP- and POMDP-based robot-control systems, all of which rely on combinations of the planning algorithms and dimensionality management techniques previously reviewed.

2.6 Applications in Robot Control

MDP theory and RL are closely related to control theory, so it should be no surprise that many of their applications concentrate on motion planning and often lie within the field of engineering. Instead, this section focuses on task planning applications, where the robot must deliberate about the long term effect of its actions. Examples of autonomous robots controlled by (PO)MDP planning algorithms are normally the result of academic projects, and the ones presented below are only a handful considered particularly relevant.

Texplore, for instance, addressed multiple challenges involved in controlling an autonomous vehicle, such as a continuous state space and delayed action results, by combining an RL algorithm with random forest learning and anytime (UCT) planning (Hester and Stone, 2013). There are a number of assumptions that trade off optimality for generality, so while it solves multiple problems simultaneously it doesn't necessarily excel in all of them all the time. It is based on fully observable MDPs, so it also doesn't address partial observability.

A very interesting scenario for general POMDP planning was the robot Pearl used by both PBVI and PolCA (Pineau et al., 2003a, 2006). These methods were developed while searching for a robust robot controller that could deal with partial observability. The robot is a nursing assistant that

must find specific residents in an elder care facility and deliver messages promptly and correctly. The underlying POMDP was modeled according to the limitations of point-based algorithms, resulting in a space of 676 states. This is of course extremely small from our perspective, more than a decade later. With enough prior knowledge and human intervention, it might be possible to simplify many complex POMDPs and achieve manageable state spaces, but this eliminates the autonomy from the robot and restricts its use to only well understood problems. For this reason, one of the motivations behind this dissertation is to contribute to the development of planning algorithms that can handle very large, unfactored POMDPs without human intervention.

A more specialized example of POMDP planning is an object grasping robotic arm (Pajarinen and Kyrki, 2017). Here the task is relatively simple: there are cups on a table, some of which are dirty, and the robot must correctly identify and collect these dirty cups and transfer them to a dishwasher. Occlusion due to clutter and proximity is the main source of uncertainty, but the underlying POMDP is not particularly large. The innovation lies in estimating action success probabilities online, as the result of observations. For example, if the arm fails to grasp a cup, the grasp success probability of this particular cup is updated to reflect this event.

Finally, the results presented in (Hanheide et al., 2017) make a very compelling case for the potential integration of classical and probabilistic planning. In this project a robot finds itself in an unmapped location and it has to find a particular object (a magazine in this case), in addition to the well understood process of localization and mapping. For the most part the system behaves like a typical robot control architecture with a classical, deterministic planner. If at some point the outcome of an action is non-deterministic, a minimal POMDP is created from a set of *relevant* facts (descriptors known to be true, taken from the symbolic representation). Once the effect of the action that triggered the POMDP has been achieved (e.g.: determining the presence of the magazine in the current room), control is returned to the classical planner which continues the regular operation cycle. In this case POMDP planning is used, in minimal and very controlled scenarios, to answer the question "what do I do now?".

It is certainly possible to devise a robot control system that relies exclusively on POMDP planning, and in some problems this might be the only alternative, but in many practical problems it is likely that classical planning with assumptions will meet all performance requirements except when actions fail, plans don't work and observations don't match. It is in this case when efficiently solving POMDPs of varying degrees of complexity becomes necessary.

2.7 Summary

This chapter introduced and defined the MDP and the POMDP as the formal models used to solve probabilistic planning and decision problems. These models use a state and action representation, with transitions and observations governed by probability distributions. A set of real-valued rewards for each transition defines the immediate and long-term utility of a state and action pair, the latter of which is called the return. (PO)MDPs are solved by finding a policy that maximizes the expected return with respect to a starting state or belief, and there are many algorithms that achieve this. Exact algorithms such as value iteration scale poorly due to their exhaustive traversal of the state (or belief) space. POMDP planning algorithms have come a long way, from improved VI methods to heuristic, grid- and point-based approximations and to anytime, online algorithms. The current most successful family relies on Monte-Carlo sampling and belief-state approximations, capable of solving POMDPs formerly considered intractable.

In practice, however, there are many challenges that make problems even more complex. POMDP-based robot control systems require either very small or heavily simplified state spaces, often the result of expert knowledge and human intervention. This limits the applicability of POMDP planning algorithms and diminishes the autonomy of problem-solving agents. Our position is that a planning agent should be equipped to handle complex problems relying primarily on its own internal capabilities, for instance by focusing on the elements that truly matter and avoiding distractions. As such, the contribution of this project is a continuation of the line of work that precedes it: it attempts to improve the efficiency of solving even larger POMDPs, but with the requirements of a general purpose, online planner appropriate for autonomous mobile robots. This includes efficiently handling very large, unfactored state spaces and relying on minimal or no domain knowledge.

The progress achieved in POMDP planning through the years can be seen as a branching sequence, illustrated in fig. 2.5. This figure is not meant to accurately represent the quantitative differences between algorithms, but rather their perceived, qualitative position with respect to the class of problems they can solve in some given unit of time as well as their convergence properties. Given for example one hour of planning time, VI is restricted to extremely small problems while POMCP could solve one several orders of magnitude larger. Numerically all methods are ultimately only as powerful as value iteration; the approximations introduce (manageable) errors in order to speed up value computation.

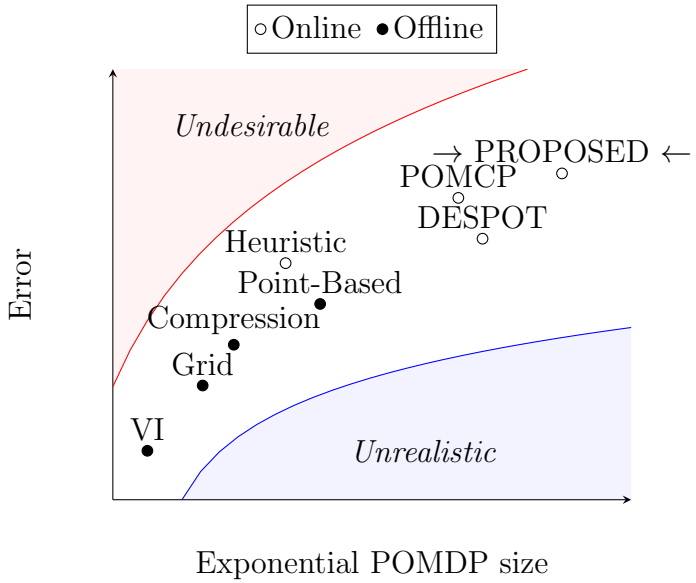


Figure 2.5: Qualitative comparison of POMDP planning methods

At the origin of this figure are the exact methods, such as VI and its improvements, including the Witness and Incremental Pruning algorithms, which are numerically powerful but severely limited in applicability. All approximate methods build upon VI-based concepts, either in the form of grids, state compression or heuristics (such as HSVI and AEMS, which are anytime and online planning methods respectively). The family of point-based algorithms, such as PBVI, Perseus and SARSOP was very successful and is still used in many cases for small POMDPs, due to its good convergence properties. Monte-Carlo methods such as POMCP constitute a significant scalability increase at the cost of a small error margin, dependent on the quality of random sampling. The shaded regions represent the scaling limitations of planning methods: solving challenging problems with arbitrarily low performance should be avoided, and achieving arbitrarily low approximation errors in difficult domains is implausible.

Contributions can go in the direction of either axis: error minimization or scalability. DESPOT, for instance, performs similarly to POMCP but has better worst-case guarantees. Building on the benefits of Monte-Carlo based methods, the results of this project place it in a comfortable position with the convergence guarantees of MCTS but with improved performance in very challenging POMDPs.

Chapter 3

Framing the problem

Towards efficient planning under uncertainty

This chapter addresses the main question that governs this thesis. Namely, *how can we improve task-planning under uncertainty for mobile robots and similar agents?*. We will begin constructing an answer by discussing the limitations of the current planning algorithms, and the reasons why they are limited in such ways. The chapter also describes the methodology used to obtain and measure results, and the planning domains used to obtain experimental results.

3.1 Challenges and Expectations

POMDP planning has advanced relatively quickly in the last decade but is still limited in several important aspects. One important limitation across most methods is their reliance on manually designed heuristics and other forms of expert domain knowledge, such as preferred actions. Approaches such as state factoring and action hierarchies also depend on a prior analysis of the problem, its variables and their dependencies, which results in an agent planning over a partially solved domain. This thesis supports the position that, in order to truly endow robots and other mobile planning agents with the ability to solve an ever growing class of complex problems, we need to grant them a form of independence. This means it is not practical or feasible to rely on prior expert analysis, and instead we should focus on planning methods that allow the agent to perform the required simplifications

internally and directly over a complex problem representation.

At the end of the previous chapter we suggested two main directions for contributions to the state of the art: scalability and error minimization. Our proposal focuses on scalability, and addresses the limitations of current planning methods in very large, complex planning domains. Algorithms such as DESPOT and POMCP have already achieved acceptable performance in relatively difficult tasks,[1] but they are still susceptible to the *apparent* complexity of many practical problems: the combinatorial complexity derived from the cross-product of many elements, actions and observations. In practice, a vast majority of these interactions are completely irrelevant to achieve the agent's goal, and yet they'll keep the agent busy for long periods of time.

In order to improve the scalability of POMDP planning, we propose granting the planning agent the ability to estimate the relevance of interactions, as well as the ability to focus on those that are most likely to be useful to achieve its goals.

3.2 Problem Description

This work is a proposal to scale the performance of online POMDP planning in large and complex domains. From a modeling perspective, POMDPs constitute a rich framework for robotic task-planning under uncertainty for several reasons: they explicitly represent the (stochastic) effect of information-gathering actions, actions are assumed to be non-deterministic, and policies depend on a correct representation and estimation of the agent's true state (its belief state). More specifically, however, we're interested in the class of POMDPs that model realistic scenarios, such as those a typical mobile robot might encounter. We'll call these *practical problems*:

Definition 3.1. A **practical problem** is situated in a complex, noisy or cluttered domain and has an open solution or terminal condition, but no specific constraints on how to achieve the solution other than general numerical performance criteria.

These problems and the underlying POMDPs are considered **complex** because they require fairly elaborate policies that contain actions with delayed rewards, and the many interactions available lead to a very large search space that complicates identifying which actions actually contribute to the problem's solution.

When attempting to solve a complex practical problem, the expected result in terms of numerical performance (such as reward maximization or

[1] In simulated benchmark problems

cost minimization) lies within some acceptable range and is not necessarily (theoretically) optimal, since there are many performance factors to consider. For instance, in many circumstances an agent must act relatively quickly even at the cost of plan "optimality" (if duration is not punished). An important challenge for a robot acting and planning in such a domain is focusing on a "sparse" set of relevant actions, instead of conducting full-width planning over a "dense" or very large set of actions with probabilistic outcomes. Such an agent is expected to achieve ϵ-optimal behavior in short amounts of time, where the error rate ϵ may vary from domain to domain.

A simple example of a small practical problem is the archetypical "coffee cup" world. If a person asks a robot to "bring a cup of coffee", the robot simply has to find any cup, some coffee, pour it, and bring it back, while avoiding any distractions such as objects that are not cups and drinks that are not coffee.

3.3 Overview of the Solution

The overarching theme of this thesis is a formal understanding of *relevance* within the context of probabilistic planning, which we previously defined as:

Definition 3.2. Relevance (in planning) is an attentional filter guiding an agent's perception and action selection, implemented through operators or functions, and leading to a series of simplifications

reproduced from (Saborío and Hertzberg, 2017). In the same publication, we argued that scaling performance in complex domains requires adopting a *satisficing* rather than an *optimizing* approach, and that simplifying assumptions are necessary to achieve this. These assumptions included using general domain knowledge to resolve certain non-deterministic transitions without engaging in full-width planning, and exploiting the structure of problems by looking for partial solutions. The main challenge outlined in that paper was identifying *when* to adopt such assumptions, and in response we sketched the idea of "context sensitive dimensionality reduction" based on a relevance-driven planning approach that attempts to avoid states that don't contribute to reaching the goal and improves action selection by using smaller action sets with valuable entries.

The work presented in this thesis directly addresses these ideas. The first component is an action selection method with improved sampling efficiency, called *Partial Goal Satisfaction* (PGS), that exploits reward sources when available leading to a series of implicit preferred actions. The second component is a relevance estimation method called *Incremental Refinement* (IRE),

that allows an agent to avoid entire subsets of state transitions that don't contribute to reaching the goal, reducing the dimensionality of complex problems while planning online. Finally we include an analytical perspective on the integration of POMDP task-planning and plan-based robot control, and suggest two algorithms to interleave POMDP planning and action execution on robots more efficiently.

We adopt the POMCP sampling approach, based on a tree of histories and belief state approximation, as well as mixed-observability states with fully and partially observable elements. Some variables, such as the agent's location, are always known, but the properties of objects (such as their type or location) are not. States are seen as enhanced feature vectors (meaning a feature is not restricted to numerical information), so state variables will also be referred to as features. For instance in a domain with a cup, a coffee maker and a spoon, each of these (as well as the agent's location) would be considered a feature.

These contributions form the basis of this approach that we refer to as *relevance-based* or *relevance-aware* planning, illustrated in fig. 3.1. These methods allow a goal-driven agent to plan efficiently in very large domains by quickly simplifying the underlying representation, autonomously and online.

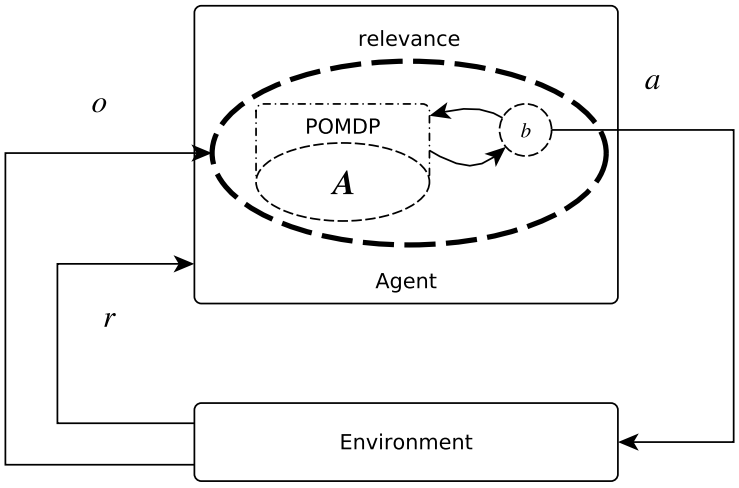


Figure 3.1: Relevance-based planning

More specifically, relevance refers to the use of goal-driven action selection (PGS) while planning with a relevance-aware version of partially observable Monte-Carlo, that also updates relevance values for IRE features. The action space as well as the current POMDP and belief state are all affected by the

dimensionality reduction of IRE, represented by the dotted lines in all of these structures in fig. 3.1.

3.4 Methodology

Before describing the solution in detail in the following chapters, we will review the methodological approach followed to verify its soundness and validity. There are two important elements in this thesis: a theoretical proposal, and its experimental validation.

The theoretical component consists mainly of the design of mathematical structures and value functions that yield the desired results, as well as their numerical approximations for efficient online computation. This results in a series of significant changes to the partially observable Monte Carlo planning algorithm, in order to enable IRE. Theoretical results include the preservation of the convergence properties of the underlying mathematical structures, and for the new elements a formal analysis of their respective error bounds is provided.

The experimental component consists of a systematic comparative analysis between the proposed system and POMCP, its closest, most significant predecessor. As previously stated, DESPOT's performance is slightly above POMCP but it requires heuristics. Since our focus is granting a planning agent independence and allowing it to generate its own simplification criteria, we can only compare against general approaches like POMCP. The scalability tests use a set of challenging problems modeled as POMDPs and multiple simulated experiments were conducted on differently sized versions of the planning domains. Planning performance is measured using the discounted return averaged over multiple executions, which is useful as an indication of plan quality and length, as well as the standard error as an indication of variability.

These experiments where mostly executed on a dedicated computer with the following specifications: two Intel Xeon E5-2660 v4 CPUs with 14 cores and 28 threads each and base frequency of 2.00 GHz, with 188 GB RAM and Linux kernel 4.4. Each individual run is sequential, but collecting performance statistics itself is *embarrassingly parallel*,[2] which allowed us to simulate many episodes with many simulations per step by running independent processes. The software used in the experiments was based on the POMCP 1.0 C++ source code, with substantial modifications and additions to its core functionality. The version used in this thesis was very experimental but

[2]Each task can be run in parallel without synchronization

a stable, standalone release will be made available.

3.5 Planning Domains

Executing experimental planning algorithms onboard physical robots introduces many additional challenges that lie outside of the scope of this thesis. These include, for example, designing and programming the communication interfaces between the planning software and the control software (such as ROS messages),[3] as well as identifying and possibly constructing a suitable physical space to run performance tests, which can also take considerable amounts of time. For this reason, experimental results were obtained through simulated runs on a set of difficult problems specifically designed to challenge the planning agent in ways that directly relate to "real world", practical problems.

Typical benchmark POMDPs like Rocksample, originally proposed by (Smith and Simmons, 2004), and even the large POMDPs used in (Silver and Veness, 2010), such as the game Battleship and a partially observable version of the videogame Pac-Man, all present the agent with an ideal, preprocessed domain that contains only the information needed to achieve goals. Despite the (combinatorial) complexity of these potentially large state spaces, the agent always receives observations and always interacts with elements that provide goal-related information. Rocksample and similar problems succeed at representing the core goals of planning, but fail to account for significant and very important challenges: in the "real world", a robot finds and interacts with many different types of objects, it has many possible actions and it receives many different observations. From these only a few are actually relevant to achieve the goal, and the rest are obstacles that must nonetheless be addressed one way or another while planning. These POMDPs are therefore insufficient, as they do not fit our definition of practical problems.

Because of its generality Rocksample was nevertheless used to test action selection, but in order to conduct meaningful experiments with relevance-driven planning two new POMDPs were designed: the Cellar domain, and the Big Foot domain. Each constitute a class of scalable POMDPs, based on more realistic scenarios and with plausible reward distributions that more accurately represent the cost of actions. Rocksample and the two new complex POMDPs are described in the remainder of this section.

[3]The *Robot Operating System* (ROS) is a stack of open-source software tools and protocols to write robot control programs. URL: `https://www.ros.org/`

3.5.1 Rocksample

In Rocksample, a "Mars rover" explores an area with rocks scattered along the surface. Rocksample$[n, k]$ defines an $n \times n$ grid with k rocks, some of which are valuable and the rover should sample. It does not know in advance which rocks are in fact valuable, but it can receive observations from a noisy sensor. The sensor follows an efficiency parameter η, such that when $\eta = 1$ the sensor returns the correct observation and when $\eta = 0$ the probability is equal for a correct and an incorrect observation. The probability of receiving a correct observation can be generalized as:

$$P(o|\delta) = \frac{1 + 2^{-\delta/\delta_H}}{2} \tag{3.5.1}$$

where δ is the distance between the agent and a rock, and δ_H is the distance at which the sensor reaches half its efficiency. When $\delta = 0$ the efficiency is 1 and as $\delta \to \infty$, $P(o|\delta) \to 0.5$, meaning correct and incorrect observations are equiprobable.

Possible actions in this problem are driving North, South, East and West, *checking* any rock, and (if available) sampling a rock, for a total of $5 + k$ actions. There are two observations: *good* and *bad*, corresponding to valuable and non-valuable rocks respectively. The reward distribution is as follows:

- -10 for sampling a non-valuable rock
- $+10$ for sampling a valuable rock
- $+10$ for leaving

with no penalty for moving or using the sensor. The state space is determined by the cross product between the agent's location and k binary rock types. In the experiments, however, Rocksample uses a slightly enhanced state description that also includes each rock's probability of being valuable, estimated from observations.

Fig. 3.2 illustrates a possible layout for Rocksample[7,8]. A straightforward solution for this problem is to scan every rock at least once, drive towards valuable-looking rocks and sample them, and continue to drive towards the exit.

This particular configuration does not constitute a very large problem and can be solved even by point-based algorithms, if properly factored. Complexity increases if the problem is scaled either in grid size or number of rocks, and quickly becomes intractable for point-based methods.

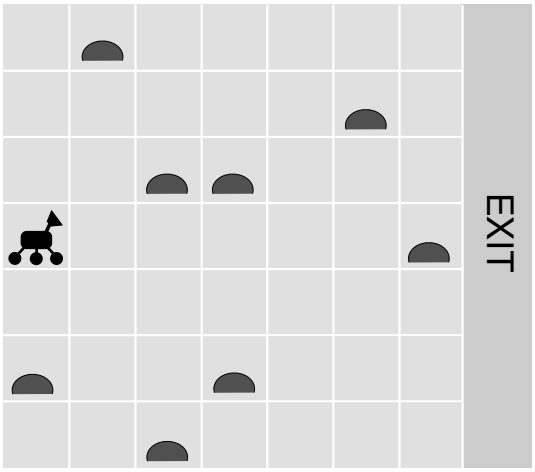


Figure 3.2: Rocksample[7,8]

3.5.2 Cellar

The Cellar domain was devised as a series of POMDPs that more closely model robot task-planning under uncertainty, avoiding some oversimplifications such as including only relevant objects or unrealistic assumptions about the cost of actions. It was first published in (Saborío and Hertzberg, 2019b). In this problem, inspired by Rocksample, the agent must navigate a wine cellar and collect *at least one* valuable bottle, while it encounters other objects such as shelves and crates that don't necessarily contribute to reaching the goal. Cellar$[n, k, s, c]$ defines an $n \times n$ grid, with k bottles, s shelves and c crates. Crates can be pushed in the direction of any free grid cell (i.e. one that contains no bottles or objects), but attempting to push anything else is not allowed and is therefore punished. A *check* action is always available for any object and any bottle, and the agent may move in four possible directions (North, West, South, East) resulting in a total of $5 + 5(k + s + c)$ actions. Each bottle can be either *good* or *bad* and each object can be either a crate or a shelf, resulting in 4 total observations in two independent classes. Initially the agent knows only the location of the bottles and of the objects, the bottles have equal probability of being good or bad and the objects equal probability of being a crate or a shelf. The sensor efficiency and the corresponding *check* actions work exactly like in Rocksample.

The reward distribution for the Cellar domain is:

- $+10$ for collecting a good bottle
- -10 for collecting a bad bottle

- +10 for leaving the grid (with at least one good bottle)
- −0.5 when using the sensor
- −1 per movement step
- −2 for pushing crates
- −10 for pushing anything other than a crate

This distribution follows an attempt to better capture robot planning, where no action is truly free and some actions (such as pushing) can be relatively expensive even if they are necessary, leading to more complex policies. Additionally, rewards implicitly relate to the goal and in domains as complex as this one, time restrictions with free actions can lead to undesirable (yet rational) behavior such as spending the allotted time performing only *check* actions and not moving or sampling, thus minimizing the loss in the total cumulative reward but not actually reaching a terminal state. This type of behavior may result in very poor policies with deceivingly acceptable performance.

The special layout of Cellar[7,8,7,8] used in the experiments is shown in fig. 3.3. Bottles that are known to be good are shown in red, whereas bottles known to be bad are shown in green and bottles of unknown type are gray. Small objects are crates and large objects are shelves, but uncertainty about their type is represented by a blurred object.

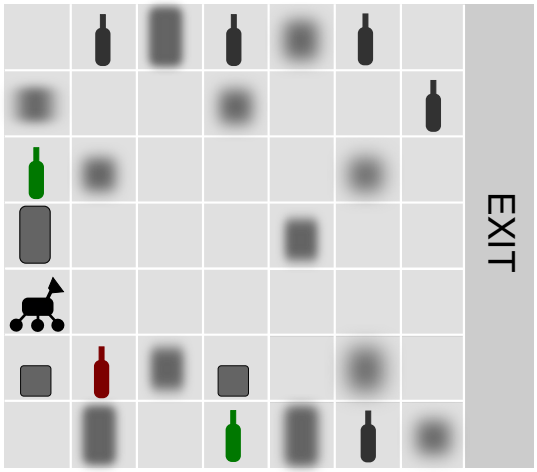


Figure 3.3: Cellar$[7, 8, 7, 8]$

State complexity in this problem scales quickly. A sketch of a solution is, for example, to check every object at least once to obtain information about

it, and drive towards good bottles to collect them. In the case a good bottle is behind an object, it might be a good idea to check the object again to confirm it is a crate before attempting to push it.

3.5.3 Big Foot

The third problem, called Big Foot, models an unmanned aerial vehicle (UAV) exploring a grid-shaped forest looking for the elusive Big Foot, the ape-like creature of North American folklore. This problem was first presented in (Saborío and Hertzberg, 2019a). The task is relatively simple: the UAV must take photos of Big Foot, which are considered valuable, and avoid taking pictures of anything else, such as trees or bears. The challenge lies in the multiple sources of uncertainty: the agent starts only with knowledge about the amount of creatures, but does not know their location or their type. The state description contains the agent's current location (which is fully observable) plus probabilistic information about the type and last seen location of each creature, all of which are *tagged* with identification numbers and (except for trees) can move to adjacent grid cells stochastically. The general problem BigFoot$[n, c, t]$ defines an $n \times n$ grid with c creatures and t trees.

Actions include moving North, South, East and West, waiting at the current location, leaving, as well as scanning a location with a noisy sensor, identifying a creature with a low resolution camera and taking a high quality picture. Grid regions are scanned with a sensor based on Rocksample's distance-sensitive scanner, and if a creature is observed its corresponding (current) location is updated. False readings are possible, in which case a randomly chosen id number is observed. Identifying creatures and taking pictures succeeds only when the UAV is hovering directly above them. Identification assumes the UAV has some form of creature or object classifier and its accuracy corresponds to the success probability. Pictures succeed always, and photographing a creature also reveals its type, but photos of anything other than Big Foot are punished. Observations include the ground and the type of creature (Big Foot or not), plus each possible creature id, resulting in a total of $6 + n^2 + 2(c + t)$ actions and $3 + c + t$ observations. The task is completed by taking a specific amount of pictures and leaving the area. Leaving without valuable pictures is also punished.

The reward distribution is as follows:

- -0.5 for sensing actions (location check and identify)
- -1 for navigation (including waiting and leaving) and misuse of identify (e.g.: creatures with unknown location)

- +5 for taking good pictures
- −10 for bad pictures (anything other than Big Foot)
- +10 when leaving with valuable pictures
- −100 in case of failure (e.g.: leaving too soon)

In the experiments the creatures moved to an adjacent cell with probability 0.15 and identification accuracy was 0.9, but these parameters can be changed freely to increase the challenge. Figure 3.4 illustrates a small version of this domain, with two creatures (one Big Foot and two bears) and two trees.

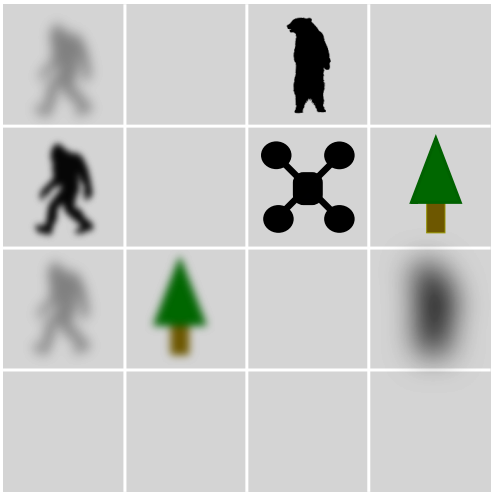


Figure 3.4: BigFoot[4,3,2]

Uncertainty is about the type of creature is represented by a blurred outline, such as the one near the bottom right. The somewhat blurry creature to the left is very likely to be Big Foot, and the lighter outlines above and below it represent uncertainty about its current location. The UAV could attempt to move towards this grid cell and take a picture, but it will be punished if the creature is not really there or if it was wrong and the creature was, in fact, not Big Foot. This is a very challenging family of POMDPs that require complex policies with delayed rewards, so it is difficult to sketch a simple solution. In general, the agent should attempt to locate the creatures, and as soon as a creature is located it should be identified. Once the agent is confident a creature is Big Foot, it should be photographed, as it may move at any moment. For this reason it is also very difficult to design heuristics and preferred actions for this problem.

3.6 Summary

This chapter introduced the concept of *practical problem*, which represents most robotic task-planning problems, and explained the limitations of POMDP planners. We outlined the methodological approach taken to extend the scalability of planning methods on large and complex POMDPs, based on a formal understanding of *relevance*. The planning problems used to obtain experimental results were also described in detail.

Chapter 4

Relevance in Action Selection

This chapter presents and formalizes PGS, a way to compute the proximity of states to goals and provide a reward bonus in action selection, with a corresponding policy for Monte-Carlo rollouts. This approach improves action selection in planning algorithms by exploiting information about the goal, allowing an agent to internally generate a set of preferred actions. As a result, newly discovered states are initialized following an optimistic approach and the planning agent is encouraged to pursue promising actions. PGS performs particularly well in domains with high variability and large branching factor, where many actions are available but agents must quickly focus on only a few useful alternatives.

The following sections contain the theoretical foundations of PGS and a comparison with the state of the art, as well as experimental results and their respective analysis. This chapter is based on previously published work. The equations, figures and some results are reproduced from (Saborío and Hertzberg, 2018) and its extended version, (Saborío and Hertzberg, 2019b).

4.1 Goal Proximity

The efficiency of a planning algorithm can be understood, intuitively, as its ability to quickly separate *good* or promising states from *bad* or unwanted states, preferring those that lead to the goal and avoiding a large number of those that don't. Most planning domains, even those without clearly specified goals (such as a pure reinforcement leaning task), have terminal states or conditions that specify what needs to be accomplished and, to some extent, what subgoals the agent should pursue. For instance, in order to "bring

a cup of coffee" a robot has to find it, collect it and deliver it, implicitly decomposing the problem into subgoals. Evidently, it is reasonable to assume planning agents are at least partially informed and aware of at least part of their goal(s) (otherwise they would hardly qualify as goal-driven agents). Any sufficiently detailed state description (such as a feature vector) provides information to compute, for any given state, a numerical score representing how many requisites from the terminal state have been accomplished at a given time. The larger this number is, the closer this state is to being a terminal or a goal state. We call this idea PGS: an implicit decomposition principle that awards additional rewards to the agent upon the completion of subgoals.

Goal proximity through PGS is counted through *points*, awarded by examining the different features in a state according to their type. Fully observable features can be easily counted since their meaning is clear in advance: with respect to the goal, their contribution is either positive, negative or neutral. This corresponds to eq. 4.1.2. The usefulness of partially observable features cannot be known in advance, so instead their contribution to the goal is the amount of information they provide, affected by information gathering actions. These actions should, for example, increase the probability that their current, estimated value is correct, which also affects the agent's belief about its current state ($b(s)$). In other words, collecting information about a given set of partially-observable features yields a better estimate of the world's current, true state. The simplest, most general approach is therefore measuring some form of uncertainty or entropy and awarding points when the uncertainty is reduced to an acceptable level, as in eq. 4.1.3.

We can now formally define PGS. Let:

- $\mathcal{F}_s$ be the set of features of state s
- $f \in \mathcal{F}_s$ be a feature of s
- $G_+ \subseteq \mathcal{F}_s$ be the set containing the observable features present in the goal
- $G_- \subseteq \mathcal{F}_s$ the set of observable restrictions
- $\Delta_f(s)$ the set of states reachable from s using f (similar to the transitive closure of $T(s, \cdot)$)
- $G_p \subseteq \mathcal{F}_s$ the set of partially or non-observable elements
- $\mathfrak{p} : S \to \mathbb{R}$ be the PGS scoring function, such that:

$$\mathfrak{p}(s) = \sum_{f_i \notin G_p} \mathfrak{f}(f_i) + \sum_{f_j \in G_p} \mathfrak{o}(f_j) \tag{4.1.1}$$

which means the total PGS score is the sum of points awarded for both fully and partially observable features, where:

$$\mathfrak{f}(f) = \begin{cases} 1 & \text{iff } f \in G_+ \\ x \in (0,1) & \text{iff } \exists s' \in \Delta_f(s) \text{ s.t.:} \\ & f' \in s' \wedge f' \in G_+ \\ 0 & \text{iff } f \notin \{G_+ \cup G_-\} \\ -1 & \text{iff } f \in G_- \end{cases} \tag{4.1.2}$$

is the fully observable point scoring function and

$$\mathfrak{o}(f) = \begin{cases} 0 & \text{iff } H(f) \leqslant T_H \\ -1 & \text{otherwise} \end{cases} \tag{4.1.3}$$

is the partially observable point scoring function, with $H(f)$ the entropy of the distribution of feature f and T_H the threshold corresponding to the minimum acceptable entropy. For example, a feature whose probability is determined by a sensor that returns either *yes* or *no* (e.g.: is this rock valuable?) constitutes a Bernoulli process and $H(f)$ is simply the binary entropy function:

$$H(p) = -p \log_2 p - (1-p) \log_2 (1-p) \tag{4.1.4}$$

This is, in fact, a very convenient way to model the probability of partially observable features in task planning domains, in which an agent usually looks for a particular (goal-related) object.

Let's summarize PGS intuitively: In PGS, observable features required to complete the goal add points and observable features that detract from the goal deduct points. Fractions of a point can be awarded for state changes that lead to a positive feature ($f' \in G_+$) in a future state ($s' \in \Delta_f(s)$), since they implicitly contribute to satisfying subgoals. An example is interacting with an object referenced in the goal, such as picking up the coffee cup that goes on the table. In this same example, once picked up, dropping the cup would reduce the PGS score so the agent would implicitly avoid it. Finally, features that don't contribute or detract from the goal don't award or remove any points. In the coffee cup example, this might include irrelevant objects on the table, such as plates or a water pitcher.

Partially-observable features are scored based on the entropy of their underlying distribution, punishing features or states with high uncertainty. Whenever enough information is gathered and the entropy is reduced below some threshold T_H, this punishment is removed. This encourages the agent to quickly get rid of this penalty, reducing the uncertainty from partially observable features by executing information gathering actions. This, in turn, may lead to the discovery of new reward sources (e.g.: relevant but previously unrecognized elements). For example, if an object in the coffee cup domain is on the table and has equal probability of being a full or an empty coffee cup, gathering information is not only crucial to completing the goal but also opens up new interaction possibilities, such as picking up the cup if it is in fact full.

In principle, any combination of the individual features in the goal may be considered a subgoal for scoring purposes, and only completing all of them simultaneously yields the total, problem-defined terminal reward. This scoring is derived from a clearly specified goal, which should always be available to a planning agent or robot. Specific problems can of course always benefit from prior domain knowledge, but the strength of this method is that it doesn't require it. Example 4.1 illustrates how PGS scoring works in practice following the aforementioned coffee cup problem.

Example 4.1. *Consider a very small POMDP in which a state is a feature vector that contains, among other features, the tuple* $\langle \ldots, g, c_1, c_2 \rangle$*, where* g *indicates which object the robot is currently grasping and* c_1 *and* c_2 *are both coffee cups with initial probability* 0.5 *of being either full or empty (defined in this case as not full). Possible actions include moving around, grasping objects and scanning them, which returns one of two possible observations: full or empty. The goal in this example is to collect the full coffee cup and leave the room. A possible PGS point distribution is:*

- -1 *point for each cup with high uncertainty*
- -1 *point if an empty cup is grasped*
- 1 *point when a full cup is grasped*

Table 4.1 shows a possible execution path and the corresponding internal changes (in bold font):

In this example $t = 0$ *corresponds to the initial state, where no goal conditions have been met, and which receives a penalty of* -2 *due to the lack of information in both cups. At* $t = 1$ *the scan reveals* c_1 *is very likely not full, so its entropy penalty is removed and the new score is* -1 *(let's assume*

Table 4.1: PGS in the coffee cup domain

t	Action	Effects: $\langle g$	$c_1.P(full)$	$c_2.P(full)\rangle$	$\mathfrak{p}(s)$
0	none	$\langle\emptyset$	0.5	0.5$\rangle$	-2
1	scan c_1	$\langle\emptyset$	**0.01**	0.5$\rangle$	-1
2	grasp c_1	$\langle\boldsymbol{c_1}$	0.01	0.5$\rangle$	-2
3	drop c_1	$\langle\boldsymbol{\emptyset}$	0.01	0.5$\rangle$	-1
4	scan c_2	$\langle\emptyset$	0.01	**0.99**$\rangle$	0
5	grasp c_2	$\langle\boldsymbol{c_2}$	0.01	0.99$\rangle$	1
6	drop c_2	$\langle\boldsymbol{\emptyset}$	0.01	0.99$\rangle$	0

$H(0.01) \leqslant T_H$). *The robot then grasps* c_1 *which is (assumed) empty and is therefore punished, but then places it back on the counter at* $t = 3$ *reverting the count from* -2 *to* -1. *The next scan at* $t = 4$ *reveals that* c_2 *is full, so its information penalty is removed resulting in a PGS score of* 0. *Notice that already at* $t = 1$ *it is easy to see scanning* c_2 *is a better action than grasping* c_1, *which is consistent with their respective scores. Cup* c_2 *is grasped at* $t = 5$ *and its contribution to the goal results in a PGS of* 1. *If* c_2 *were dropped, as in* $t = 6$, *this condition is no longer valid and the point count for this state drops back to* 0. *Again, the point count reflects that dropping* c_2 *is not a good choice and it should be avoided. Combined with an appropriate action selection method, this results in implicit criteria for action preferences in different contexts, without an explicit list of heuristics.*

It should be noted that PGS scoring represents goal proximity, and its uses may vary depending on the context. Its intended application is as an optimistic value initialization method that allows an agent to identify subgoals and exploit immediate opportunities when available. The subtask completion information provided by PGS scoring must be combined with an appropriate action selection policy and is not meant to solve classical planning problems directly. For example, in some given Blocks World configuration it may result in overly greedy actions if it is not combined with a lookahead search method. As explained in the next sections, PGS is the foundation of a subgoal reward bonus and a rollout policy in Monte-Carlo planning algorithms, where optimistic assumptions yield a significant performance improvement.

4.2 PGS in Reward Shaping

Reward shaping is a well-known technique used to improve the performance of (PO)MDP algorithms and RL problems. It works by granting an additional reward upon some state transitions, encouraging the agent to choose certain *implicitly* preferred actions. In practical applications, the magnitude of this reward bonus often comes from an in-depth analysis of the structure of the problem so it constitutes a form of heuristic bias in action selection. For example, in a simplistic navigation problem, the reward could be based on the Euclidean distance from the current location and the intended destination.

Instead of providing explicit, domain-dependent knowledge to shape rewards, we use PGS to encourage the agent to pursue courses of action leading to the completion of subgoals. Under normal circumstances, simply awarding a reward bonus would result in a new reward distribution and a (PO)MDP with potentially different policies. Instead we adopt the potential-based approach which guarantees that the introduction of implicit subgoals doesn't affect the algorithm's underyling convergence properties and, therefore, policies are guaranteed to transfer back to the original problem (Ng et al., 1999).

At its core reward shaping simply substitutes the usual reward function in an MDP with:

$$R(s, a, s') = r(s, a, s') + F(s, a, s') \tag{4.2.1}$$

where r is the problem-defined reward and F is a reward bonus. A potential-based shaping function F has the form

$$F(s, a, s') = \gamma\phi(s') - \phi(s) \tag{4.2.2}$$

with $\gamma \in \mathbb{R}$. From now on we will use $\gamma_{\mathfrak{p}}$ when in the context of PBRS, to avoid confusion with the the discount factor γ commonly used in planning and learning. We now define the potential $\phi(s)$ with respect to PGS as:

$$\phi(s) = \alpha\mathfrak{p}(s) \tag{4.2.3}$$

where $\alpha \in \mathbb{R}$ is a scaling factor, that should be chosen accordingly to transform PGS points into an appropriate reward bonus. A simple suggestion is to multiply by a magnitude within the problem's reward range, such as:

$$\alpha = \frac{|R_{max}| + |R_{min}|}{2} \tag{4.2.4}$$

In practice, transitions to states that are *closer* to a subgoal and receive some amount of points will produce a positive difference, transitions to states

that are farther from subgoals receive a punishment and therefore generate a negative difference, and other transitions cancel each other out. As previously stated, reward shaping functions are normally specific for particular problems and dependent on prior domain knowledge, but PGS manages to attain simplicity and generality using only implicit information already available to the agent.

Example 4.2. *Continuing with the coffee cup domain in example 4.1, let's assign the following reward distribution:*

- -0.5 *for scanning*
- $+10$ *if a full cup is grasped*
- -10 *if an empty cup or unknown is grasped*

and let's reiterate the PGS points distribution:

- -1 *for cups with high uncertainty*
- -1 *if an empty cup is grasped*
- 1 *when a full cup is grasped*

Now consider the initial state:

$$s_0 = \langle \ldots, g = \emptyset, c_1.P(full) = 0.5, c_2.P(full) = 0.5 \rangle \text{ with } \mathfrak{p}(s_0) = -2$$

in which all cups have equal probability of being full or empty and the robot isn't currently grasping anything. Table 4.2 shows, for each available action in s_0, the resulting rewards using the PGS shaping function. We chose $\alpha = 10$ in order to scale PGS points to the problem reward range, ± 10, and $\gamma_{\mathfrak{p}} = 1$ to weigh both point counts equally. Remember that $\mathfrak{p}(s_0) = -2$, so in each entry $F(s, a, s') = 10(\mathfrak{p}(s') + 2)$. For simplicity, the resulting state shows only the changes.

Table 4.2: Reward shaping in the coffee cup domain

a	changes in s'	$r(s, a, s')$	$\mathfrak{p}(s')$	$F(s, a, s')$	$\boldsymbol{R(s, a, s')}$
scan c_1	$c_1.P(full) = 0.01$	-0.5	-1	10	9.5
scan c_2	$c_2.P(full) = 0.99$	-0.5	-1	10	9.5
grasp c_1	$g = c_1$	-10	-2	0	-10
grasp c_2	$g = c_2$	-10	-2	0	-10

In this starting state (again, assuming $H(0.01) \leqslant T_H$ *and* $H(0.99) \leqslant T_H$*), scanning either object has a large reward bonus while grasping yields no bonus, since there is no change in goal proximity. Let's assume that* scan c_2 *is chosen as the next action and the resulting state is:*

$$s_1 = \langle \ldots, g = \emptyset, c_1.P(full) = 0.5, c_2.P(full) = 0.99 \rangle \text{ with } \mathfrak{p}(s_0) = -1$$

with its corresponding values in table 4.3, where $F(s, a, s') = 10(\mathfrak{p}(s')+1)$.

Table 4.3: Reward shaping in the coffee cup domain, part 2

a	changes in s'	$r(s, a, s')$	$\mathfrak{p}(s')$	$F(s, a, s')$	$\boldsymbol{R(s, a, s')}$
scan c_1	$c_1.P(full) = 0.01$	-0.5	0	10	9.5
scan c_2	$c_2.P(full) = 0.999$	-0.5	-1	0	-0.5
grasp c_1	$g = c_1$	-10	-1	0	-10
grasp c_2	$g = c_2$	10	0	10	20

Since c_2 *is already assumed to be full, scanning it again does not change goal proximity and yields only the action's reward without a bonus. Alternatively, obtaining information about* c_1*, which is unknown, does offer a bonus and a total reward of* 9.5*. Cup* c_1 *is still unknown, so grasping it produces a negative reward with no bonus. Grasping* c_2 *instead, which is full, does offer a bonus for a total reward of* 20*.*

As shown in example 4.2, an action selection policy can be easily built from PGS scoring. This is addressed in the following section.

4.3 PGS in Rollouts and Simulation

Monte-Carlo Tree Search algorithms, such as UCT and POMCP, work by sampling sequences of states from a probabilistic transition model. A tree is progressively expanded and the average returns and visit counts are maintained per tree node. When enough statistics are available (for instance, when the node is already initialized) the UCB1 rule is used to select an action, balancing *exploitation* of promising actions and the *exploration* of unvisited states. When a new, unvisited state is discovered, a rollout or random simulation is performed and its outcome used as an initial value estimation. Rollout policies are therefore responsible for a significant portion of the performance of MCTS planning algorithms. Using PGS as a rollout policy, an agent can quickly focus on actions that directly contribute to the completion of (sub)goals and, likewise, avoid undesirable actions that produce little

or no benefits. Selecting actions by maximizing state-to-goal proximity can also implicitly summarize a very rich array of knowledge and heuristics, that must otherwise be given explicitly. To the best of our knowledge, we are the first to study the effect of evaluating goal proximity within the context of Monte-Carlo rollouts.

The PGS rollout policy is as follows: let s be the agent's current state and $\mathcal{A}$ a set of actions. PGS action selection consists of selecting an action $a \in \mathcal{A}$ that leads to a state $s' \leftarrow (s, a)$, such that s' satisfies the largest amount of subgoals. If multiple such actions exist, the tie is broken randomly. The action selection policy is formalized in eq. 4.3.1:

$$\mathcal{A}(s) = \arg\max_{a} \mathfrak{p}(s' \leftarrow (s, a)) \tag{4.3.1}$$

which substitutes the rollout policy of standard Monte-Carlo or similar planners and initializes new states optimistically. However, when enough statistics have been gathered (e.g.: during the selection stage in MCTS), policies such as UCB1 should be used to guarantee convergence. An additional goal-inspired simplification is to define $\mathcal{A}$ as the set with legal actions and uncertainty reducing actions only, skipping for example information-gathering actions if they do not provide any more information. In the previous example, action *scan* c_2 (in table 4.3) can be avoided in rollouts since it already satisfies $H(p) \leqslant T_H$.

Example 4.3. *In order to illustrate PGS in rollouts and simulation, let's make some changes to our coffee cup domain. Now, grasping a cup also reveals whether it was in fact full or empty, so we'll maintain the PGS distribution but the reward distribution becomes:*

- -0.5 *for scanning*
- $+10$ *when a full cup is grasped*
- -10 *when an empty cup is grasped*

This means there is a certain amount of risk the agent can assume when grasping a cup.

Going back to the initial state:

$$s_0 = \langle \ldots, g = \emptyset, c_1.P(full) = 0.5, c_2.P(full) = 0.5 \rangle \text{ with } \mathfrak{p}(s_0) = -2$$

its point distribution is given by table 4.4.

We now have non-deterministic outcomes for the grasp action, which results in different rewards when a cup is grasped blindly (without enough information about its type). In this state, depending on the starting belief rollouts

Table 4.4: Action selection with PGS in the coffee cup domain

a	changes in s'	$r(s,a,s')$	$\mathfrak{p}(s')$	$F(s,a,s')$	$\boldsymbol{R(s,a,s')}$
scan c_1	$c_1.P(full) = 0.01$	-0.5	-1	10	9.5
scan c_2	$c_2.P(full) = 0.99$	-0.5	-1	10	9.5
grasp c_1	$g = c_1, c_1.P(full) = 0$	-10	-2	0	-10
grasp c_2	$g = c_2, c_2.P(full) = 0$	-10	-2	0	-10
grasp c_1	$g = c_1, c_1.P(full) = 1$	10	0	20	30
grasp c_2	$g = c_2, c_2.P(full) = 1$	10	0	20	30

will sometimes use only scan *actions, and other times only* grasp *actions (a cup* should *in fact be grasped if it is believed to be full).*

This does not yet represent the actual action-value, however. Regardless of what is selected first using this optimistic action set (which depends on the current belief sample), the grasp action value will approach its true expected value which is, at this point, lower than the scan action value. As previously explained, POMDP Monte-Carlo planning approximates the true belief with a particle filter and states are sampled for planning, so the "current" state of each cup is determined from the belief distribution. For both grasp actions (regardless of which cup), let's assume there is equal probability of sampling a state where the cup is full and where the cup is empty (since we don't have any information about the cups yet). The value of grasp *will slowly approximate:*

$$\begin{aligned} q(s,a) &= \sum_{s',r} p(s'|s,a)\left[R(s,a,s') + \gamma \max_a q(s',a')\right] \\ &\approx 0.5\left[-10 + \gamma \max_a q(s',a')\right] + 0.5\left[30 + \gamma \max_a q(s',a')\right] \\ &\approx 10 + \left[\gamma \max_a q(s',a')\right] \end{aligned}$$

which suggests grasping an unknown cup is enticing, but risky. If we had more information about the optimal future rewards resulting from these actions, we could better approximate the true action-value function. Let's continue the example one step further by selecting action scan c_2*, as we did before. Assuming c_2 is full, we obtain the state:*

$$s_1 = \langle \ldots, g = \emptyset, c_1.P(full) = 0.5, c_2.P(full) = 0.99 \rangle \text{ with } \mathfrak{p}(s_0) = -1$$

and its values in table 4.5.

Depending on the belief sample, the rollout action set might be { *scan* c_1*, grasp* c_2} *or* { *grasp* c_1} *if such a belief is sampled. Keep in mind that in*

Table 4.5: Action selection with PGS in the coffee cup domain, part 2

a	changes in s'	$r(s,a,s')$	$\mathfrak{p}(s')$	$F(s,a,s')$	$\boldsymbol{R(s,a,s')}$
scan c_1	$c_1.P(full) = 0.01$	-0.5	0	10	9.5
scan c_2	$c_2.P(full) = 0.999$	-0.5	-1	0	-0.5
grasp c_1	$g = c_1, c_1.P(full) = 0$	-10	-1	0	-10
grasp c_1	$g = c_1, c_1.P(full) = 1$	10	1	20	30
grasp c_2	$g = c_2$	10	0	10	20

either case, based on what the agent knows or believes, the action set reflects a rational choice, and an optimal one if the belief is correct (the challenge is approximating the true state of the world).

During simulation the true action values are approximated, and the reward bonus quickly introduces an action preference. Scanning c_2 in the previous step returned information that affected the belief distribution, and therefore the expected value of grasp c_2*: it now has an expected action-value of $20 + \gamma \max_a q(s', a')$, while c_1 is still unknown and grasping it results in $10+\gamma \max_a q(s', a')$. If we propagate these values back to the previous step, as a planning algorithm would, the resulting action value approximation would be:*

- scan c_2*:* $9.5 + [\gamma 20 + \gamma \max_{a'} q(s'', a'')]$
- grasp *any cup:* $10 + [\gamma 10 + \gamma \max_{a'} q(s'', a'')]$

With $\gamma = 0.95$, we obtain $28.5 + \gamma \max_{a'} q(s'', a'')$ if we scan c_2 at the beginning, while grasping any cup blindly results in $19.5 + \gamma \max_{a'} q(s'', a'')$. That is, with just one more step, it is clear scanning is the better choice.

In practice, scan c_2 *followed by* grasp c_2 *will be preferred often, and the reward bonus will make these actions clear winners during simulation, accomplishing the objective of exploiting reward sources.*

As shown in example 4.3, a reduced action set allows the agent to focus on the most promising (goal-related) actions. When not enough information is available, the agent focuses on what actions seem reasonable and quickly approximates their true values. To summarize, PGS relies on estimating goal proximity and provides a reward bonus and an action selection policy in rollouts, both of which allow a planning agent to focus on achieving its objectives and avoid actions that bring it further away from the goal (such as dropping the full coffee cup in the examples above). Current generation methods rely on manually designed heuristics and explicit preferred actions to achieve good performance, and we attempt to eliminate or minimize these dependencies.

4.4 Results

PGS was tested in three different domains. The first one is the Taxi domain, which defines a fully-observable MDP commonly used in planning and learning due to its simplicity. The other two are Rocksample and Cellar, previously defined in detail. For the taxi problem we implemented a straightforward C++ version of UCT with no enhancements other than PGS, and for the others PGS was added to the POMCP source code (also C++).

The challenge for robot planning under uncertainty is achieving good performance within a finite horizon, fast enough, even in large problems. These scenarios show the performance of PGS using limited resources (very few or relatively few Monte-Carlo simulations) and how it scales in considerably large POMDPs. The results in the Taxi problem are meant only for future reference, since basic UCT was nowhere near solving the problem using the small amount of simulations required with PGS.

This section is based on material previously published in (Saborío and Hertzberg, 2019b) and (Saborío and Hertzberg, 2018), but also contains new results.

4.4.1 Taxi

The taxi domain was first proposed to illustrate planning and learning with the MaxQ action hierarchy (Dietterich, 2000). It is a very simple MDP where the agent moves in any of four directions in a 5×5 grid and must pick up a passenger located in one of four possible depots, and bring it to another depot. A slight variation is the "fickle taxi" in which movement is non-deterministic: with a small probability (eg. $p = 0.1$) the taxi will end up East or West of its intended direction. Possible actions are moving North, East, South or West, collecting a passenger when standing on the same grid cell or dropping the passenger (when carrying one). Rewards are -1 for each regular move, -10 for dropping the passenger in the wrong location and 20 for delivering a passenger correctly, which also terminates the episode. We didn't formally introduce this problem in the previous section because we'll only use it once.

We chose one instance of the taxi domain and obtained the total discounted reward of its optimal policy, 8.652 (with $\gamma = 0.95$), in order to compare it with the experimental results. This particular configuration and in general the taxi domain are illustrated in figure 4.1, where the dark cell at the top left corner is the goal depot where the passenger must be dropped. The walls shown in the picture are also included in the experiment which

means the agent's movement is restricted, in cells next to walls, to only open, adjacent cells.

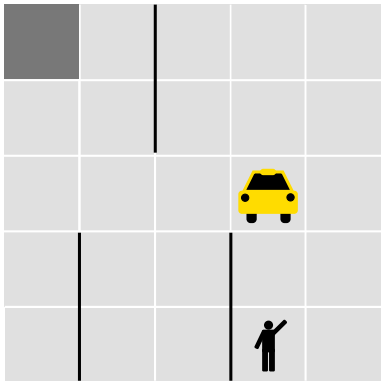


Figure 4.1: The Taxi domain

Because all elements are fully observable, PGS in the taxi domain is easy to formalize. We simply award 0.25 points for picking up the passenger and 1 point for dropping it at the correct depot (G_+). There are no restrictions ($G_- = \emptyset$). The terminal state reward is preserved but PGS reward substitution is used for the rest, with $\gamma_{\mathfrak{p}} = 1$ and $\alpha = 10$. Finally, a discount factor of $\gamma = 0.95$ and search depth of 90 steps were used within UCT. It is common to allow the taxi to start only over a depot, but in our experiments it could be anywhere on the grid. We ran UCT with PGS in both (regular and fickle) versions of the taxi domain, and obtained the average discounted returns (with standard error) and running times over 1000 runs. Table 4.6 shows the result of runs on the fixed task (fig. 4.1) and on randomly generated configurations (randomized origin and destination depots, and taxi starting location) using only 1024 simulations, extremely few for Monte-Carlo standards. This table was adapted from (Saborío and Hertzberg, 2018).

Table 4.6: Performance in the Taxi domain

Transitions	Start state	Avg. Return	Time (s.)
Normal	Fixed	6.161 ± 0.083	3.049
	Random	4.257 ± 0.193	3.531
Fickle	Fixed	3.275 ± 0.137	4.41
	Random	2.138 ± 0.212	4.176

Results are promising if we compare the average discounted reward with the optimal policy in the fixed (non fickle) task (8.652). Restricting the amount of computation to only 256 simulations per move ($\approx$ 1.6 s. per

episode), the PGS-based planner achieved an average discounted reward of 5.089. On random tasks it is important to mention that episodes were terminated after 5 s., but their (negative) reward still averaged.

Averaging performance, especially in stochastic domains, may hide interesting details of particularly good runs. We ran a separate batch of 1000 episodes using 1024 simulations, of which 616 finished in 2 s. or less and 797 in 3 s. or less. In these test the statistical mode was the maximum discounted reward (8.652), meaning most runs found the optimal policy.

Finally, figure 4.2 shows how PGS scaled in the Taxi domain. Performance and standard error were averaged over 1000 randomly generated runs, with a more strict acting budget of 35 steps but a more generous timeout per episode of 30 s. to allow for enough planning time with up to 8192 simulations. PGS quickly achieved satisfactory performance, even with very few simulations which translates into a very short planning time (shown at the top). We don't include comparative results with plain UCT (without PGS) because, in practice, it required (comparatively) excessive amounts of time and simulations to achieve even little improvements in performance. This figure has been adapted from (Saborío and Hertzberg, 2019b).

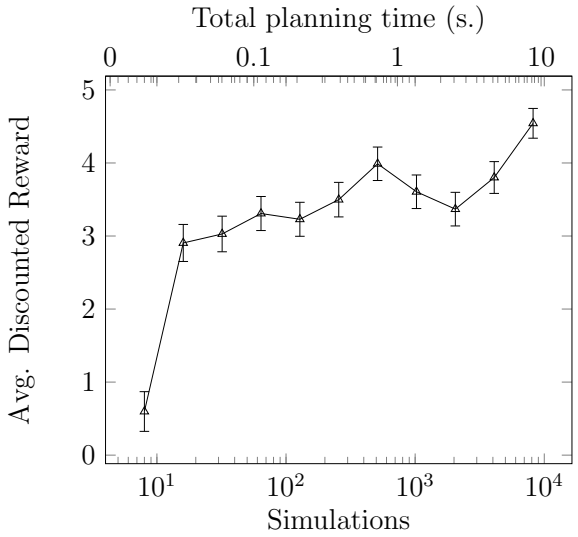


Figure 4.2: PGS in the Taxi domain

Not surprisingly, these results show how performance in fully observable tasks can be substantially improved by following a relatively straightforward goal-driven, action selection method that implicitly exploits the problem structure.

4.4.2 Rocksample

Results in the Rocksample domain are based on four problem sizes: $[11, 11]$, $[15, 15]$, $[25, 12]$ and $[25, 25]$, the latter of which is a fairly large problem with approximately 10^{10} states. The purpose of testing on $[25, 12]$ is comparing the performance of these policies in a large grid with more scarce reward sources. These results were obtained by adding PGS to POMCP and comparing the results of the uniformly random policy restricted to legal moves (we'll call it *legal*), POMCP's handmade heuristics for rocksample (called *smart*) and PGS action selection (rollout policy and reward bonus). PGS used the following parameter values:

- $\gamma_{\mathfrak{p}} = 1$
- $\alpha = 10$
- $C = G_+ \cap G_-$ is the set of sampled rocks
  - G_+ is the set of good rocks
  - G_- is the set of bad rocks
- G_p is the set of non-collected rocks.

with the following PGS point distribution:

- 1 when sampling a good rock with good observations
- -1 when sampling a bad rock
- -1 for each rock with uncertainty over the threshold

We chose an entropy threshold of $T_H = 0.5$, so using the binary entropy formula (eq. 4.1.4) the agent receives a punishment for every rock with probability $0.11 < p < 0.89$. In practice points are deducted in undesirable states (e.g.: collecting bad rocks, high entropy for any rock) and awarded only when collecting good rocks. This means that during rollouts *check* will be preferred if it reduces entropy for some rock, that *sample* will be preferred when standing over a promising rock, and that otherwise movement actions will be considered. Preferred actions in Rocksample instead follow manually encoded heuristics such as "head North if there are rocks with more positive observations" or "check rocks that have been measured less than five times and have less than two positive observations". An important aspect of PGS is that it achieves good performance and succeeds in avoiding this level of over specification.

Figures 4.3 and 4.4 show the discounted returns averaged over 1000 runs (or 1000000 maximum seconds of total experiment runtime) for all three policies in Rocksample [11, 11], [15, 15], [25, 12] and [25, 25], with up to 65536 Monte-Carlo simulations per move and a maximum of 100, 200, 300 and 350 steps per episode respectively.

In all plots, PGS drastically outperforms the legal actions policy and is only slightly outmatched by the heuristic policy. This difference is maintained and in some cases reduced as the problem size increased, particularly in [25, 25] which is a very large problem for *rocksample* standards and in [25, 12], an equally large grid but with fewer reward sources. In the medium sized problems (fig. 4.3), given enough resources PGS closely matched the heuristic policy, while in the larger problems (fig 4.4) PGS performed as well or better than using heuristics, a significant achievement if we compare the simplicity of PGS with the detail necessary to specify preferred actions.

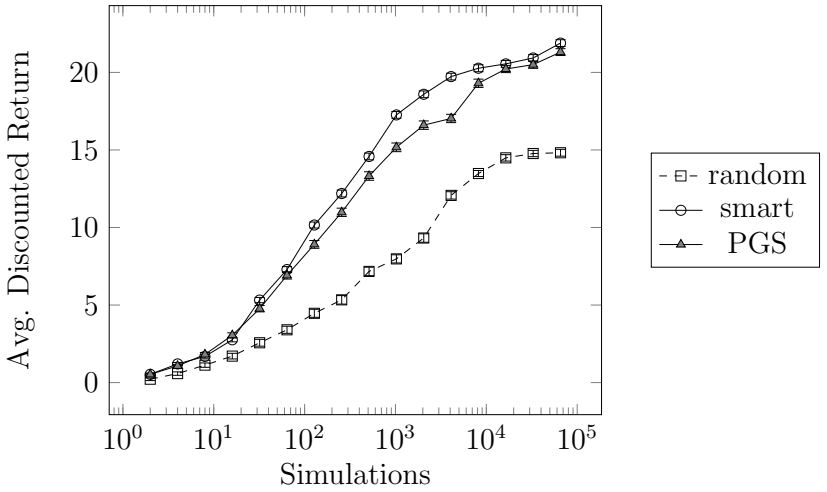
Estimating PGS values for actions in the rollout policy can be somewhat computationally expensive, particularly when all or many actions are available, since it attempts to be more selective. This is perhaps the only advantage the heuristic policy has over PGS in a straightforward problem like rocksample. Results show, however, this extra processing pays off compared to the random policy. Table 4.7 compares the average discounted returns and average runtimes between the results produced by the random policy with the maximum budget (65536 simulations), and the closest equivalent result (in terms of runtime) when using PGS. Clearly PGS outperforms random POMCP even with equivalent time restrictions: it produces higher discounted returns in less time.

Table 4.7: PGS – Runtime comparison in Rocksample

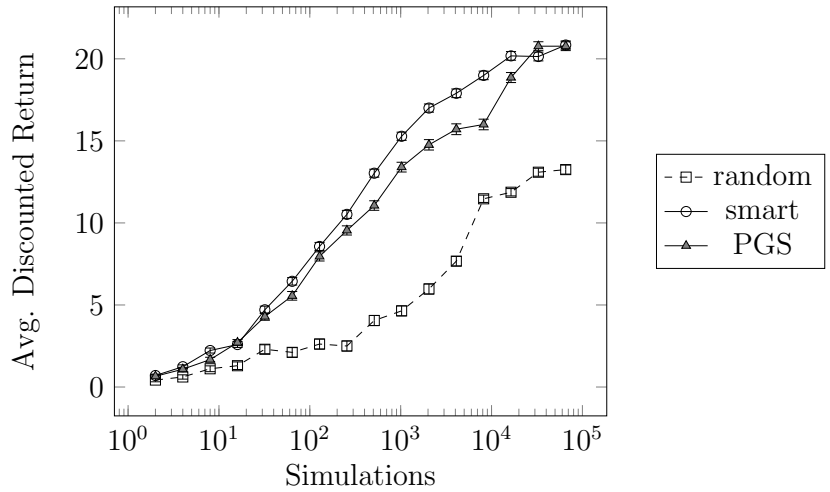
Size	**Random**		**PGS**	
	Disc. Return	**Time (s.)**	**Disc. Return**	**Time (s.)**
[11,11]	14.82 ± 0.219	47.6	16.6 ± 0.272	31.33
[15,15]	13.25 ± 0.274	126.9	14.76 ± 0.317	78.72
[25,12]	9.304 ± 0.238	235.1	13.5 ± 0.266	213.3
[25,25]	9.234 ± 0.194	376.5	13.26 ± 0.279	284.4

4.4.3 Cellar

We designed special layouts for two cases of interest in the Cellar domain: Cellar[7,8,7,8] and Cellar[11,11,15,15]. The first is a POMDP with 73 actions and 2×2 observations resulting in a state space of more than 10^{15} states.

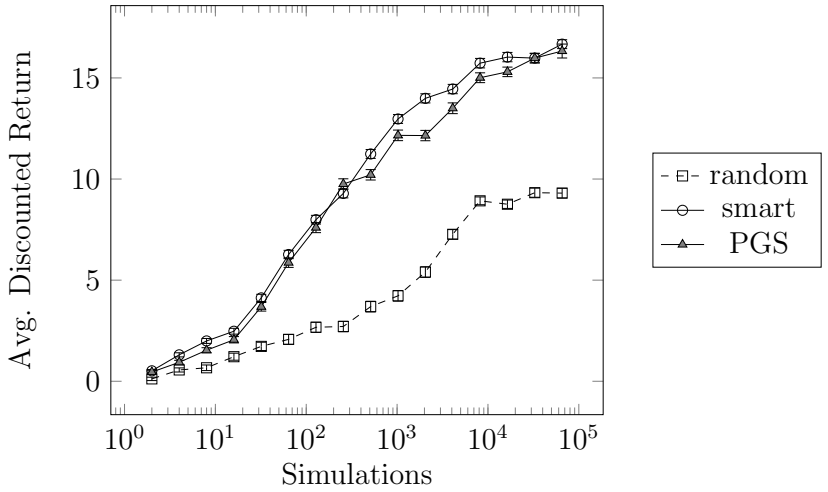


(a) [11,11]

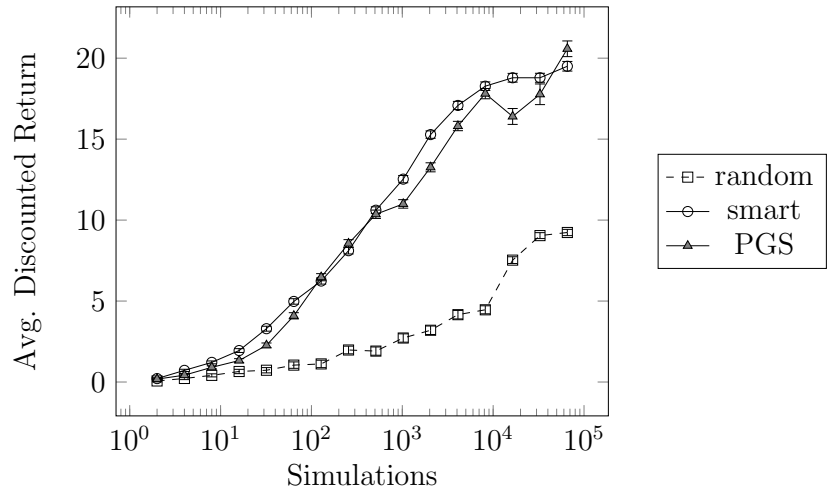


(b) [15,15]

Figure 4.3: PGS in Rocksample (medium)



(a) [25,12]



(b) [25,25]

Figure 4.4: PGS in Rocksample (large)

The second problem defines a POMDP with 127 actions, 2×2 observations, and a very large state space of approximately 10^{31}.

PGS in the Cellar domain is very similar to PGS in Rocksample. We used:

- $\gamma_{\mathfrak{p}} = 1$
- $\alpha = 10$
- $T_H = 0.5$
- $C = G_+ \cap G_-$ is the set of collected bottles
  - G_+ is the set of good bottles
  - G_- is the set of bad bottles
- G_p is the set of non-collected bottles.

and for PGS scoring:

- $+1$ for collecting valuable bottles with good observations
- -1 for collecting bad bottles
- -1 for each bottle above the entropy threshold

Despite crates and shelves also constituting sources of uncertainty, they should not be punished because they do not form part of the goal. The agent will interact with crates and shelves during the course of planning and acting, and eventually detect if it makes sense to push a crate aside or not, in which case it will also be useful to make sure it is, in fact, a crate. They might be essential to execute a plan, but they are not a critical component of the goal.

In this domain, rollouts use a reduced action set that contains only movement, sampling, pushing and uncertainty-reducing *checks*, avoiding information-gathering actions for objects that already meet the entropy requirements. We also created a heuristic, preferred actions policy based on the "smart" POMCP policy: the same movement, checking and sampling heuristics apply, plus equivalent heuristics for object checking. The "legal" rollout policy again refers to the uniformly random policy that considers all valid actions.

In order to more clearly illustrate the impact of a goal-driven bias in action planning, we will first consider a minimal version of the cellar problem: a 5×5 grid with only 1 bottle surrounded by 4 crates, called cellar[5,1,0,4] (fig. 4.5). This simple POMDP has a very clear, recognizable goal, and yet it has

around 6 million states (already beyond the scope of point-based methods, even if factored). Agents solving this problem should quickly realize they must *simply* push a crate, collect the bottle and leave, trying not to move around aimlessly or unnecessarily checking and pushing objects.

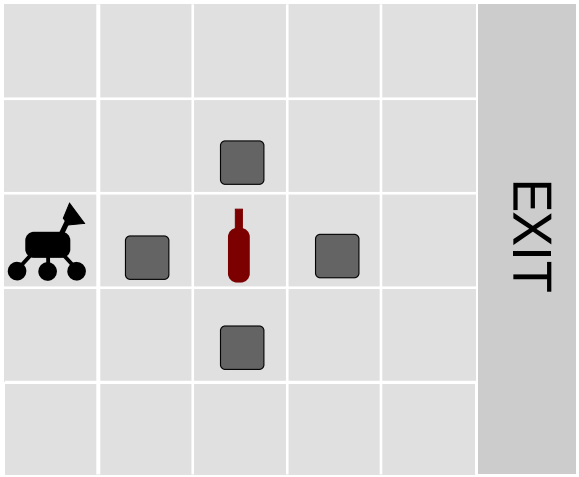


Figure 4.5: Minimal Cellar example

Figure 4.6 shows the performance of all three policies in the minimal version of the Cellar domain. The total discounted return was averaged over 1000 runs with up to 65536 Monte-Carlo simulations per step and discount $\gamma = 0.99$, and a generous acting budget of 100 steps.

With only 128 simulations PGS already achieves higher returns than the two competing action selection policies, and given more resources it dramatically outperforms them. It is also interesting that the random policy performs better than the heuristic policy.[4] It's possible that these explicit action preferences, extended from the Rocksample domain, are simply too limiting. They are defined mostly with respect to goal-related elements (bottles in this case) and appear to be insufficient to address a complex problem like the cellar domain. These restrictive heuristics, in fact, make the agent perform even worse than the uniformly random policy. Efficiently solving straightforward but challenging problems such as cellar, with increasing levels of difficulty, would require increasingly more sophisticated heuristics every time (meaning they don't scale well). Instead PGS required no additional modeling (i.e. from Rocksample to Cellar) to achieve positive results.

[4]The reason the plot starts at 16 simulations is, precisely, the very poor performance of the heuristic policy with a very small planning budget. Under 16 the range of returns was as low as -400, making the rest of the plot less legible.

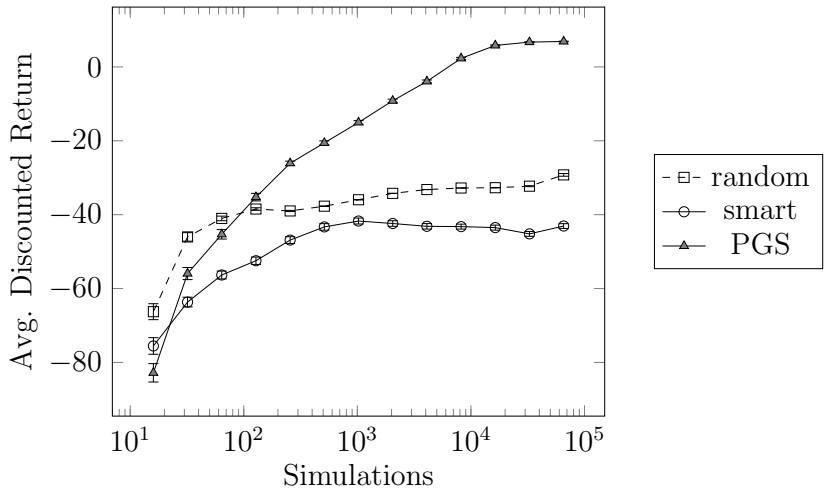


Figure 4.6: PGS in cellar[5,1,0,4]

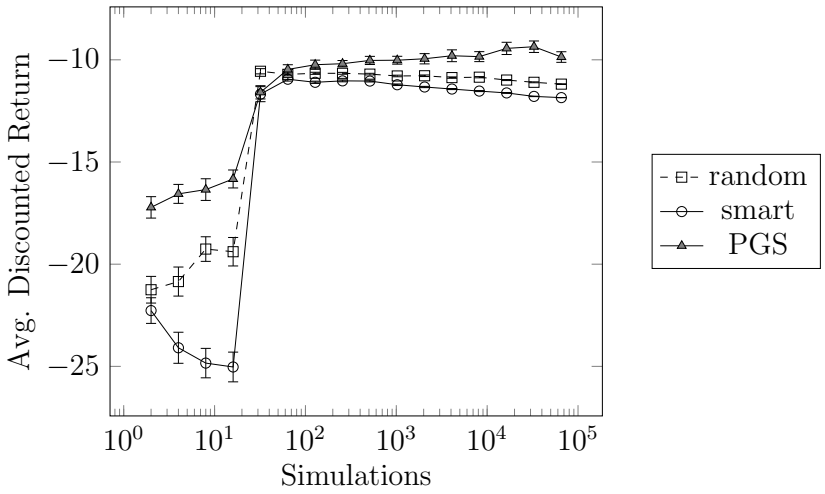
Table 4.8 shows the respective average runtimes and discounted returns for each policy in cellar[5,1,0,4] using the maximum planning budget of 65536 simulations. In this problem configuration, in addition to returns, PGS also outperformed the two others in runtime. If we consider return over time as measure of efficiency, PGS performs exceedingly well.

Table 4.8: PGS – Runtimes in cellar[5,1,0,4]

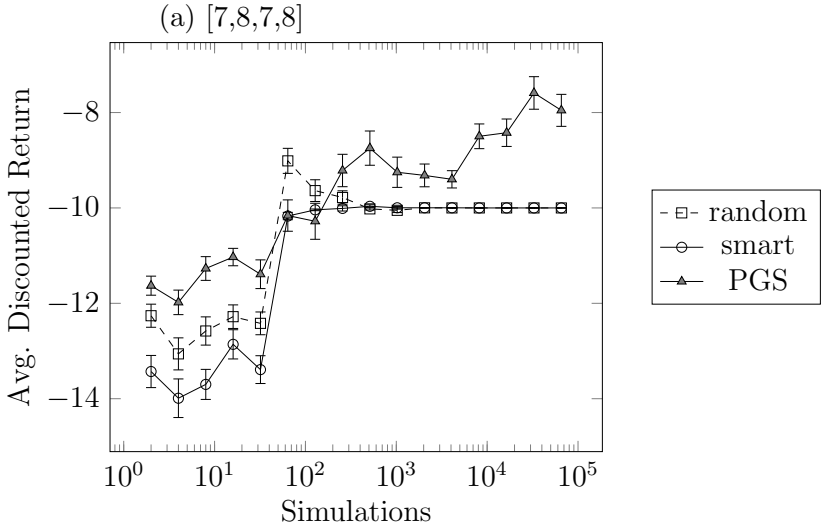
Policy	**Disc. Return**	**Time (s.)**
Random	-29.23 ± 0.364	498.3
Heuristic	-43.08 ± 0.649	164.8
PGS	6.884 ± 0.085	33.38

Cellar[5,1,0,4] represents an instance of a decision problem where policies have vastly different results, designed with the purpose of exposing the limitations of heuristic and uniformly random planning. More realistic settings, however, are likely to have a combination of scenarios such as this one as well as more straightforward choices as is the case of the more general Cellar problems. The comparative performance of PGS in the large Cellar configurations is shown in figure 4.7, which averages the discounted returns of 100 runs with up to 65536 simulations per move. We used a discount factor of $\gamma = 0.95$ in both problems and a maximum budget of 250 steps in cellar[7,8,7,8] and 350 steps in cellar[11,11,15,15].

Results in the full Cellar problems show that in very large, task-planning



(a) [7,8,7,8]



(b) [11,11,15,15]

Figure 4.7: PGS in Cellar

POMDPs, PGS scaled much better than the other two policies. In cellar[7,8,7,8] (fig. 4.7a) with very few simulations PGS already outperforms both random and heuristic rollouts, and with more simulations it gained a considerable advantage.

Cellar[11,11,15,15] (fig. 4.7b) is a very large problem that requires complex policies involving information gathering and interaction with numerous obstacles. We gathered statistics with "only" 100 episodes to avoid very long computing times, but this produced results with fairly large variability (i.e. the sharp spikes present in the plot). Despite this, results clearly indicate PGS scales much better than the two competing policies and manages to collect higher rewards in this challenging problem. The random and heuristic policies both appear to have settled on a low local maximum, possibly limited by their respective sampling approaches.

4.4.4 Discussion

The two POMDP classes we tested showed how PGS scaled when planning in incrementally larger domains. The Cellar domain in particular shows the performance of planning with obstacles, unnecessary actions and a more realistic reward distribution. The minimalist version of Cellar highlights the importance of quickly identifying reward sources and a plan that exploits these rewards, something that the two other policies failed to do (a dramatic failure considering how manageable the state space is).

The two larger Cellar problems constitute a much more complicated optimization challenge that benefits from goal-oriented behavior, as shown by the comparative performance of PGS. In both of these problems, the preferred actions policy did not scale well at all. In fact, the action recommendations appear to limit the agent's performance. This means that an entirely new set of heuristics are necessary to address Cellar, despite it clearly constituting a specialized version of Rocksample. Possible heuristic rules in Cellar seem quite complex too: an agent can only push a crate when it is directly next to it, and it *should* only push a crate when it leads to a *good* bottle. A possible rule might look like:

> Given an agent's location, if there is a crate directly to the East, and there is a bottle directly North or South of the crate, and the bottle seems good, and the crate can be pushed East, then add *push crate East* to the list of actions.

This should be repeated for every direction, and as shown in the experiments, it won't transfer well into variations of the same domain. This level of

specification is exactly what we propose should be avoided in order to solve large problems effectively and efficiently.

It is possible, however, to combine the heuristic approach and PGS in a hybrid rollout policy. Perhaps this might be useful in domains that are relatively well understood but nevertheless too complex to specify heuristic rules for every type of action. Our intention, however, is to improve an agent's autonomy when planning in difficult scenarios.

In order to scale planning algorithms to progressively more interesting and challenging problems, significant leaps in performance are necessary. Our work in this topic seems to indicate that extensive, systematic sampling or a prior domain analysis synthesized in heuristic rules do not work too well. On the other hand, exploiting the underlying model and the knowledge already available about the goal and the effects of actions produces a very noticeable increase in performance using comparable resources. The small cellar problem is a very telling example: given a similar situation, most human beings would quickly devise an effective plan and could easily explain the connection between pushing the crate and retrieving the bottle. The size of the belief space does not really matter much, since we avoid most non goal-related interactions. We have attempted to capture this process by granting a planning agent the formal tools to estimate goal completion and focus on the actions that seem to contribute the most.

4.5 Contributions, Related Work and Outlook

With PGS we attempted to grant a planning agent the ability to exploit reward opportunities when available, resulting in very interesting behavioral effects. Exploiting reward sources means a planning agent will avoid actions that don't produce positive results and prefer those that have more promising outcomes, autonomously constructing an implicit set of preferred actions based on their expected contribution.

Our experimental results show that despite its simplicity, PGS effectively improves the performance of planning in large and complex (PO)MDPs, thanks to its goal-driven approach to action selection. In the fully observable Taxi problem, PGS achieved a level of performance that far exceeds that of a uniformly random policy in standard UCT. In domains with partial and mixed observability and particularly in problems with scarce reward sources, such as larger versions of Rocksample, PGS easily outperformed the uniformly random policy and closely followed a manually designed, heuristic

policy. In such problems, the random policy scaled poorly compared to the policy based on domain knowledge, which managed to achieve acceptable performance. We showed, however, that by exploiting implicit knowledge about the goal, PGS can successfully compete against the manually designed, heuristic action-selection policy. This type of domain independent bias is particularly important for planning and acting onboard robots, and in complex domains that require considering not only expected outcomes, but also the information gain of uncertainty-reducing actions. This is supported by the observed performance of the (heuristic) preferred actions policy in the Cellar domain, in which the core heuristics become insufficient and perform even worse than the random policy, highlighting the importance of goal-oriented behavior.

We can identify three main approaches for speeding up action planning in large stochastic domains: 1) State abstractions that approximate the values of unknown states, 2) Action hierarchies that produce smaller, abstract MDPs whose solutions can be transferred to the original MDP and 3) PBRS, that encourages an agent to focus on good action prospects, avoiding potentially costly choices. As previously mentioned, hierarchical planning and learning requires human input, often in the form of Hierarchical Task Networks, that allow the agent to focus on simpler levels of abstraction (Sutton et al., 1999; Dietterich, 2000; Pineau et al., 2003a; Vien and Toussaint, 2015). Instead, PGS introduces implicit action preferences by exploiting knowledge about the goal. PBRS is a way to implement action preferences and subgoals, but is traditionally based on expert domain knowledge. Again, our approach uses a potential function based on an abstraction of goal-proximity, so the resulting bias responds to the domain and the agent's history. Because it operates at the level of action selection, PGS can and should be combined with other high level criteria to form a comprehensive, relevance-driven or relevance-aware approach to planning, such as the dimensionality reduction approach presented in the next chapter.

To conclude, we would like to reiterate our position that the intrinsic relationship between the values of states and their proximity to the goal (or subgoals) is an essential component of efficient action selection. The perceived complexity of planning problems is often justified based on their worst-case computational complexity, which is commonly a function of the number of states. Planning problems however, are defined not only in terms of their transition dynamics, but also in terms of some implicit or explicit goal: planning efficiently means quickly identifying the gaps in the values of different states in order to separate "good" or promising states from "bad" states, and focusing on the good ones. These values come uniquely from perceived (or simulated) rewards, which for a given state correspond to the

average discounted return of its children. In the early stages of planning, it might be difficult to separate promising states from less promising ones if the agent is simply too far (in terms of state or belief transitions) from any source of reward and doesn't have enough knowledge or statistics to decide. However, if an agent finds itself in proximity to its goal or a reward source (e.g.: a few actions away), the problem should be simple to address or even solve if the appropriate actions are chosen, regardless of the number of reachable states and beliefs. We attempted to capture this idea by introducing the implicit subgoal decomposition and preferred actions of PGS, but further refinements such as a more efficient value computation might be useful in some demanding scenarios. For instance, we always recompute the PGS score of all available actions, but perhaps this is not necessary in cases when actions are known to perform poorly in advance. This could be implemented in combination with heuristics that *support* PGS (or other policies like it), following ideas already suggested in this chapter, such as avoiding uncertainty reducing actions once the threshold criteria has been met. We leave it as an open topic but anticipate that the principles for goal-directed action selection introduced in this thesis might be useful.

Chapter 5

Relevance in Dimensionality Reduction

Planning and acting simultaneously comes naturally to us as human beings, but is deceivingly complicated to model and replicate in artificial agents. As the result of a long evolutionary process, human beings are capable of deliberating about relatively complicated problems and often find quick, reasonable solutions by employing a number of cognitive resources (often influenced by a personal or subjective bias). When attempting to solve problems, we can shift our attention across the entities or elements that describe the problem and selectively focus on one or more at a time, in order to examine their potential contributions or utility. Normally, however, we are not aware of the underlying selection process and do not consciously reason about the selection criteria. Artificial agents (such as mobile robots) must, on the other hand, explicitly allocate resources for deliberation – a task that grows in complexity in non-deterministic and uncertain domains. If we intend to design agents that can efficiently plan, or deliberate about the effects of actions, and subsequently execute these actions in a seamless, continuous stream to solve practical problems, we must find ways to eschew brute-force computation.

This chapter addresses the complexity of planning on large and complex POMDPs and solving practical problems using the Incremental Refinement (IRE) approach. IRE allows an agent to quickly disregard entire subsets of the action space (and their derived states and observations) by estimating *feature* relevance values. We show how this idea easily integrates into Monte-Carlo POMDP planning and provide experimental results in the Cellar and Big Foot domains. A significant part of this chapter is either based on or

reproduced from (Saborío and Hertzberg, 2019a), including the experimental results.

5.1 Domain Features and Complexity

Practical problems, as previously defined, correspond to POMDPs where an agent can move (in 2D or 3D), manipulate objects and obtain information through sensors. These objects or entities in the planning domain correspond to features, and we argue that the complexity of such a POMDP is largely dependent on them. This notation assumes states with mixed observability, so fully-observable features don't yield probabilistic observations. Previous studies of point-based POMDP algorithms also realized that this assumption more accurately reflects the true complexity of POMDP planning (Hsu et al., 2007).

For the following, let η be the number of valid positions the agent can assume, p the number of locations or positions of features, k the number of features with variable positions (e.g.: those that move or that the agent might move), and $\Omega_f \subseteq \Omega$ the subset of observations of feature $f \in \mathcal{F}$ (eg. obtained through information-gathering actions). The state-space of such a POMDP is:

$$\underbrace{\eta \binom{p}{k}}_{\text{World configurations}} \cdot \overbrace{\binom{p}{|\mathcal{F}|} \prod_{f \in \mathcal{F}} |\Omega_f|}^{\text{Observation space}} \tag{5.1.1}$$

where the left side corresponds to the number of observable world configurations and the right side to the *possible worlds* generated by the different observations. The binary coefficient in both sides is the number of positions available to the k movable features. The observation space assumes an agent doesn't know which features can actually move, so instead a belief about the position of every feature is maintained. The product of feature-related observations gives the total distribution of qualitative observations, such as "valuable", or "useful".

For example, in Rocksample any given rock can be either valuable or not valuable. In rocksample[7,8], with 8 rocks, there are 2^8 possible arrangements for a total of 12544 states. If we transform the problem into cellar[7,8,7,8] we get 15 additional objects, 8 of which are movable, with two types of observations each, resulting in more than 10^{15} states. In a configuration where no crate has to be pushed to reach a good bottle, plans in cellar and rocksample are not very different from each other and yet we obtain an exponential

increase in state complexity, and a proportional increase (exponential in the number of states) in the belief space.

Goal-driven agents can and in fact should ignore most of these features and their actions due to them being irrelevant with respect to the goal. In standard Monte Carlo planning, for instance, these non-useful actions will eventually be avoided by an action selection policy like UCB1 but this may take arbitrarily long amounts of time. We propose estimating feature relevance directly as a function of the combined values of actions for a given feature. Features are only relevant for goal-oriented planning as long as they provide or lead to useful (goal-related) interactions, so the discounted rewards of these useful action opportunities should be reflected in their relevance value.

If POMDP states are seen as feature vectors, each element is a feature that either contributes to reaching the goal (at some point) or doesn't, in which case it becomes more of a distraction. By assigning a numerical value to a feature's *usefulness*, the agent can progressively shift its focus towards more useful features and their associated actions and ignore the rest. This constitutes the basis of IRE, which in practice uses the current, perceived relevance of features (based on available opportunities) to prune the underlying POMDP and avoid entire sections of the state space that lead to low rewards. The specifics of IRE are presented in the next section, where we formalize the necessary value functions to estimate feature relevance.

5.2 Relevance Estimation

This section provides the formal background of IRE, which consists of a feature value function, a feature relevance function, and a feature activation policy.

We first define a value function that associates action-values to features and second, a feature-value function that combines these quantities to estimate an overall relevance value. In practice it is necessary to identify which actions affect which features, and for simplicity we assume the existence of a simple lookup table with an update rule.[5] Unlike tabular methods for planning or reinforcement learning, this particular table contains entries only for each feature and action pair. This is perfectly manageable for most if not all POMDPs, unlike state or belief tables.

For simplicity these equations are given in terms of states (regardless of partial observability), but it is easy to see that their equivalents exist in

[5]It is entirely possible, as with any other planning and learning method, to substitute a table with a more sophisticated function approximation method such as a neural network

a Belief-MDP. Algorithmically, transferring these methods to POMDPs is trivial using partially observable MCTS.

5.2.1 Feature Values

Planning and learning methods for (PO)MDPs estimate an action-value function $\hat{q}(s, a)$, that approximates the true $q(s, a)$ function. This represents the discounted return of executing action a in state s and then following some policy π. The action-value function q can be written in terms of a state-value function:

$$q(s, a) = \mathbb{E}[R(s, a, s') + \gamma v^*(s')]$$

where $\gamma \in [0, 1]$ is a discount factor and v^* is the optimal state-value function, that is:

$$v^*(s) = \max_{a \in A} \sum_{s'} p(s'|s, a)[r(s, a, s') + \gamma v^*(s')]$$

Bellman's optimality equation for q is therefore:

$$q(s, a) = \sum_{s'} p(s'|s, a)[r(s, a, s') + \gamma \max_{a'} q(s', a')]$$

which relates the action-value function to its transition probability, its immediate reward ($r(s, a, s')$) and its expected reward following the optimal policy. Intuitively this means that for any given state s, all of its associated $q(s, a)$ represent the immediate and the long-term benefit of executing a, which also provides some information about a feature. For example, if $a =$ "scan object 1", $q(s, a)$ offers information about object 1 (the feature) in the form of a high or a low return. However, in some given state s and under some particular policy, the opportunity to execute a and discover its potential benefit may lie too many steps into the future so its contribution will be heavily discounted. But if this action and its associated feature are relevant in the future, they might also be (at least somewhat) relevant now. For the purposes of relevance estimation, let's assume the following:

Hypothesis 5.1. For some given $a \in A$. $\exists$ $s, s' \in S$ s.t. $r(s, a, s')$ is significant $\rightarrow$ a is relevant.

In other words, there might be a transition with a substantial reward that makes this action, and its feature, relevant. This suggests all potential future rewards should be (somewhat) equally considered. The contribution of an action-feature pair can then be estimated by *expanding* the underlying

MDP and considering all possible outcomes on the subset $\mathcal{S}_a \subseteq S$ of states where a is available:

$$\text{relevance of action } a \begin{cases} q(s_0, a) \\ q(s_1, a) \\ \cdots \\ q(s_{n-1}, a) \\ q(s_n, a) \end{cases}$$

where $\forall i. 0 \leqslant i \leqslant n \;.\; s_i \in \mathcal{S}_a$, which leads to the value function:

$$\begin{aligned} v(f, a) &= \mathbb{E}[q(\cdot, a)] \\ &= q(s_0, a)p_0 + q(s_1, a)p_1 + \\ &\quad \ldots + q(s_{n-1}, a)p_{n-1} + q(s_n, a)p_n \\ &= \frac{1}{|\mathcal{S}_a|} \sum_{s \in \mathcal{S}_a} q(s, a) \end{aligned} \tag{5.2.1}$$

We assume the distribution of action values is uniform in this case because we are dealing with prospects, estimating the value of all afforded actions, and actual q-values correctly combine reward and probability.

Example 5.1. *In the previous chapter we defined a coffee cup domain where a robot has to find and retrieve the eponymous object. Let's now consider two features: an empty cup and a full cup, and their corresponding actions. We established that picking up a cup full of coffee yields a positive reward, and an empty or an unknown cup yields a negative reward. Regardless of the initial belief state, during simulation some interactions will behave as if any one cup were full and some as if it were empty (where frequency is determined by the* correctness *of the current belief state). This means that without further information, both cups (in this case, features f_1 and f_2) are assigned a variety of rewards resulting from the different outcomes or actions, such as:*

$$v(f_{1,2}, a = \textit{ pick up}) \approx \frac{n^+(10 + x_1) + n^-(-10 + x_2)}{N}$$

and

$$v(f_{1,2}, a = \textit{ check}) \approx \frac{-0.5 + x_1}{N}$$

where x_1 and x_2 represent the future, discounted returns after executing this action (that form part of the action-value function q), N is the number

of times an action was simulated and $n^{\pm}$ *the number of instances the positive and negative rewards were received respectively.*

If a "check" action is performed on either cup and new information becomes available, the frequency of these rewards is affected during simulation and so are their values. If $f_1 =$ *empty cup and* $f_2 =$ *full cup,* $n_- >> n_+$ *and* $n_+ >> n_-$ *respectively, making the value of executing the action "pick up" on cup 2 very high and the value of "pick up" on cup 2 very low.*

Coupled with a reward bonus such as PGS, even information gathering actions can receive a clear, intuitive feature-action value. Before satisfying the entropy threshold, actions like "check" would have higher values (reflecting their reward opportunities) and afterwards, they would seem less interesting from a numerical standpoint. For instance, awarding one PGS point with a scaling factor of 10*, the feature-action value of "check" becomes:*

$$v(f_{1,2}, a = \ \textit{check}) \approx \frac{n^+(9.5 + x_1) + n^-(-0.5 + x_2)}{N}$$

which, in the correct belief state, would tell the agent that the value of the action "check" is high for some feature f*. This in turn affects the agent's perception of the current utility of* f *itself.*

An effect of this average is that if there are many states where a has a high (or low) return, this will be reflected in the value of this feature-action pair.

5.2.2 Feature Relevance

Based on the feature-action value function, we can now define the relevance of a feature as the expected value of its afforded, weighted actions. This is given by eq. 5.2.2:

$$\begin{aligned} V(f) &= \mathbb{E}\left(w(v(f,a))\right) \\ &= \frac{1}{N}\sum_a w(v(f,a)) \end{aligned} \tag{5.2.2}$$

where w is some chosen importance scaling function. Scaling is necessary to more closely model preferences and intuitive criteria in relevance estimation, some of which might be domain dependent. It also addresses potential pitfalls of averaging action values in extreme cases, among which we pay special attention to underestimation, overestimation and insufficient sampling:

1. Underestimation occurs when a majority of a feature's actions have very low rewards, but there is one (or a few) extremely good, goal-

related action. Intuitively this feature should be relevant but without scaling its relevance would appear to be very low.

2. Overestimation might happen in features that have very few or no useful actions that are also not particularly punishing. Their arithmetic mean might lie above the relevance threshold, but the feature might still not be particularly relevant.

3. Partial or insufficient sampling may be the result of a feature's actions being available only in unreachable states, so during planning (such as Monte-Carlo simulation) these transitions are never sampled. If only case 2 actions are available, its resulting value (without scaling) will be inaccurate.

We propose a simple value scaling function (eq. 5.2.3) based on the idea that the contribution of values to feature relevance is linear when they are negative but grows rapidly when they are positive, with a fixed punishment for actions that cannot be executed ($N = 0$).

$$w(v(f,a)) = \begin{cases} v^n & \text{if } v > 0 \\ \kappa & \text{if } N = 0 \\ v & \text{if } v < 0 \end{cases} \tag{5.2.3}$$

with $\kappa \in \mathbb{R}^-$ and $n \in \mathbb{R}^+$. This assumes, as is the case in most POMDPs, that goal-related rewards are positive.

Example 5.2. *Still in the coffee cup domain, let's assume the values $n = 2$ and $\kappa = -8$ in the relevance scaling function, resulting in:*

$$w(v(f,a)) = \begin{cases} v^2 & \textit{if } v > 0 \\ -8 & \textit{if } N = 0 \\ v & \textit{if } v < 0 \end{cases}$$

Now consider table 5.1, which reflects (after many iterations in some state s_i) the current values of features and actions.

The expected value of "check" for features (cups) c_1 and c_2 approximates the action reward, but there is also a new cup (c_3) that is out of reach and out of view, so it cannot be scanned or pickup up and receives a value of κ. We assumed that at this point, c_1 and c_2 are full with $p = 0.6$ and empty with $p = 0.4$, which yields the feature-action value of 2 using their known reward distribution. The resulting relevance values of each feature are shown in table 5.2, in which both c_1 and c_2 receive a positive relevance value, but the punishment in c_3 yields a considerably lower relevance estimate.

Table 5.1: Feature-action values in the coffee cup domain

f	a	$v(f,a)$
c_1	check	-0.5
c_1	pick up	2
c_2	check	-0.5
c_2	pick up	2
c_3	check	-8
c_3	pick up	-8

Table 5.2: Relevance values in the coffee cup domain

f	$V(f)$
c_1	1.75
c_2	1.75
c_3	-8

If c_1 and c_2 were scanned, further information received from observations would affect their relevance estimates. For instance, if it turns out c_1 is empty and c_2 is full, the expected value of their actions will reflect this new information, as seen in table 5.3.

Table 5.3: Feature-action values in the coffee cup domain, part 2

f	a	$v(f,a)$
c_1	check	-0.5
c_1	pick up	-10
c_2	check	-0.5
c_2	pick up	10
c_3	check	-8
c_3	pick up	-8

The resulting relevance in table 5.4, suggests that c_2 is much more important (for the goal) than anything else, and that the agent should focus on it and avoid getting distracted (especially with c_3).

Notice that these estimates do not use PGS scoring, which would produce a different set of values. With PGS, features that are not yet identified (such as a cup with equal probability of being empty and full) offer additional reward opportunities and may appear interesting or relevant, and once information is received their values (coming from interaction opportunities) would be correspondingly amplified in either direction.

Table 5.4: Relevance values in the coffee cup domain, part 2

f	$V(f)$
c_1	-5.25
c_2	49.75
c_3	-8

Although outside of the scope of this dissertation, we would like to point out that eqs. 5.2.3 and 5.2.2 could also be chosen to represent and test some particular relevance criteria, such as modeling human preferences in known domains. This might also be useful to improve the performance of robots planning in certain well understood, but nonetheless challenging stochastic domains.

5.2.3 Value Approximation

In order to easily identify action-feature pairs and values in practice, we propose using a table not unlike the one in the example above.

Definition 5.1. Let $\mathbb{T}$ be a catalog or table with entries corresponding to the mapping $\mathcal{F} \times A \to \mathbb{R}$, for features $f \in \mathcal{F}$, their associated actions $a \in A$ and their corresponding feature value v.

Table entries constitute tuples of the form $\langle f, a, v \rangle$. Values can be obtained simultaneously with state-action values, since they also use a combination of immediate and discounted delayed rewards. The function $v(f, a)$ can be computed as per equation 5.2.4:

$$v(f, a) = \frac{1}{N} \sum_{s} r(s, a, s') + \gamma_f \max_{a'} q(s', a') \tag{5.2.4}$$

where N is the number of times action a was executed and γ_f is a discount factor. Eq. 5.2.4 can be easily implemented as an online average (eq. 5.2.5):

$$v(f, a) \leftarrow v(f, a) + \frac{r - v(f, a)}{N} \tag{5.2.5}$$

where $r = r(s, a, s') + \gamma_f \max_{a'} q(s', a')$ is the immediate reward plus the discounted future reward and N counts the number of updates. We approximate $v(f, a)$ using a different discount factor, $\gamma_f < \gamma$, to reflect the short-term effect of actions instead of the long horizon normally used in planning. The final value of a feature's relevance, $V(f)$, is computed from the elements in $\mathbb{T}$.

In an online planning algorithm once an action has been chosen and executed in the "real world", all table entries must be updated to reflect the new current (belief) state. Feature relevance may change overtime, behaving in practice as a nonstationary problem. We address this by including a type of learning factor to combine previous and current estimates, weighing samples from the current state more heavily. A straightforward method is resetting the count in each tuple to 1 and using the current value as a prior, which leads to a variable learning factor of $1/N$ for N samples, a solution known to minimize regret (Auer et al., 2002). With this simple update rule, if the approximation is correct then new updates won't affect the current estimate. If the approximation is incorrect but the action is promising, the action selection policy will choose it often in simulation and the value in the table will be corrected. If the relevance value of some feature was correct but no longer valid (e.g.: the reward source was appropriately utilized), the old information will quickly be overwritten.

Intuitively, the relevance of a feature at a given time is determined by the potential interactions it offers the agent. Without useful (or available) actions, a feature might be considered irrelevant. Based on their relevance values, features can be selectively enabled and disabled online, allowing the underlying planning algorithm to focus only on those actions that affect active features. This reduces the number of reachable states and consequently the effective problem size.

5.3 Dimensionality Reduction

The strength of this proposal is that, by granting a planning agent the ability to estimate relevance values, features considered irrelevant may be ignored during planning. By focusing on promising features, the action branching factor is reduced and many (belief) states avoided, pruning the search tree. Depending on the problem, ignoring a feature might involve additional considerations (e.g.: a mobile robot may ignore a chair but not drive through it). This section is reproduced from (Saborío and Hertzberg, 2019a), with minor changes.

We propose a feature activation rule based on a simple threshold check:

$$\text{active}(f) = \begin{cases} \textbf{true} & \text{if } V(f) \geqslant \tau \\ \textbf{false} & \text{otherwise} \end{cases} \tag{5.3.1}$$

where $\tau \in \mathbb{R}$. The set of available actions at some given moment t reduces to:

$$A_t = A \setminus \mathcal{A}^- \tag{5.3.2}$$

with $\mathcal{A}^-$ the set of inactive actions, defined by the actions of inactive features:

$$\mathcal{A}^- = \bigcup_{f \in \mathcal{F}^-} \mathcal{A}(f) \tag{5.3.3}$$

where $\mathcal{F}^- = \{f \mid V(f) < \tau\}$ is the set of inactive features, and $\mathcal{A}(f) \subset A$ the subset of actions linked to feature f. This set may be used with any action-selection rule to ignore actions of features with low relevance values, resulting in relevance-aware versions of UCB, ϵ-greedy, etc.

Example 5.3. *To illustrate the activation rule, let's borrow the value tables from the previous example. After planning, we started with the following value table (let's assume a threshold of $\tau = -5$):*

Table 5.5: Relevance values in the coffee cup domain (revisited)

f	$V(f)$	Status ($\tau = -5$)
c_1	1.75	active
c_2	1.75	active
c_3	-8	inactive

Only cups c_1 and c_2 satisfy the rule and remain active. Feature c_3 is deactivated, so actions such as "pick up c_3" won't be available anymore (unless its relevance value changes in the future). Simulation proceeds as usual with action selection restricted to active actions. Right after c_1 and c_2 are scanned (and they turn out to be empty and full, respectively), the new relevance table is:

Table 5.6: Relevance values in the coffee cup domain (revisited), part 2

f	$V(f)$	Status ($\tau = -5$)
c_1	-5.25	inactive
c_2	49.75	active
c_3	-8	inactive

in which only feature c_2 and its actions remain active. This means that at this point, the actions available to the agent are restricted to navigation plus actions provided by c_2, which include scanning it and picking it up.

5.3.1 Bounds for Feature Activation

We have established that active features satisfy a relevance threshold corresponding to a high expectation of their combined, weighted action values. Inactive features are, on the other hand, those with low or no contribution (to the goal). We will now explain the properties of inactive features and their connection to the activation threshold. For the following let:

$$w^+ = \sum_a \mathbb{1}_{\mathbb{R}^+}(w(v(f,a))) \tag{5.3.4}$$

denote the sum of all positive (including 0) values,

$$w^- = \sum_a \mathbb{1}_{\mathbb{R}^-}(w(v(f,a))) \tag{5.3.5}$$

represent the sum of all negative values (where $\mathbb{1}_{\mathbb{R}}(x) = x \in \mathbb{R}$ is the indicator function), and N the number of actions linked to feature f (those that participate in its value function). Inactive features satisfy $V(f) < \tau$ and correspondingly:

$$\begin{gathered} \frac{w^+ + w^-}{N} < \tau \\ w^+ < \tau N - w^- \\ \text{(and consequently) } w^+ < |w^-| \ \wedge \ \tau N - w^- > 0 \end{gathered} \tag{5.3.6}$$

This states the mean of all positive, importance-weighted values $w(v(f,a))$ is lower than the mean of negative values. From eq. 5.3.6 it follows that:

$$\nexists a \in \mathcal{S}_a \quad w(v(f,a)) \geqslant \tau N - w^- \tag{5.3.7}$$

in other words, there is no single action with an importance-weighted value of at least $\tau N - w^-$. Since the importance weight for positive values is a power of n, and $\tau N - w^- > 0$ it is also true that:

$$\forall a \in \mathcal{S}_a \quad v(f,a) < 0 \ \vee \ v(f,a) < \sqrt[n]{\tau N - w^-} \tag{5.3.8}$$

which guarantees that an inactive feature does not have any actions such that $v(f,a) \geqslant \sqrt[n]{\tau N - w^-}$. In a given setting, if n, τ and N act as constants the main influence in the activation or deactivation of a feature is w^-. This means inactive features are those with actions that yield significantly (n-degree polynomial) larger negative reward, than they do positive reward. Modifying τ online would either relax or restrict feature activation, and perhaps such a variable policy might be useful in some domains.

5.3.2 Estimation Errors

From eqs. 5.2.1 and 5.3.8 it follows that $\mathbb{E}[q(\cdot, a)] < \sqrt[n]{\tau N - w^-}$ for inactive features, so let $\mathcal{U} = \sqrt[n]{\tau N - w^-}$ be the upper bound of the expected action value. When a feature is active, it doesn't interfere with action selection so the convergence properties of the underlying policy hold (e.g. for UCT, etc.). The same applies when a feature is inactive, and its actions are not part of a (sub) optimal policy (i.e. an action selection policy with decreasing exploration rate would eventually ignore them too).

Non feature-specific actions, such as navigation, are not affected by feature activation and are always available in their corresponding states. We then have two cases of interest: the first occurs when an action is part of an ϵ-optimal policy but is unavailable during action selection due to a value approximation error (should be active instead). The second case is when the action-value approximation is correct but it happens to be very low, and so the resulting feature-value is below the activation threshold. This means the action is required but cannot be selected.

In the first case, the action value should satisfy $v \geqslant \mathcal{U}$ but due to errors, it does not. We can then express the probability of the approximation $v + \epsilon$ falling under the upper bound as $P\{v < \mathcal{U} - \epsilon\}$ which can be estimated using Hoeffding's theorem, that states:

$$P\{\bar{X} \leqslant \mu - a\} \leqslant e^{-2a^2/n_X}$$

where $\bar{X}$ is the empirical mean from n_X elements, μ is the distribution mean and a an upper bound. In our case $\bar{X} = \hat{q}(s, a)$ is our empirical mean, $\mu = q(s, a)$ is the true mean, and $a = \mathcal{U} - \epsilon$ the upper bound. We then get:

$$\begin{aligned} P\{\bar{X} \leqslant \mu - a\} &\leqslant e^{-2(\mathcal{U}-\epsilon)^2/n_q} \\ &\leqslant \exp\left\{-2\frac{\left(\sqrt[n]{\tau N - w^-} - \epsilon\right)^2}{N}\right\} \end{aligned} \tag{5.3.9}$$

This probability is maximal when $\left(\sqrt[n]{\tau N - \omega^-} - \epsilon\right) \to 0$, which can happen under two circumstances:

1. When a very large error ϵ approaches the bound $\mathcal{U}$. This reflects a poor overall approximation, often the result of insufficient *budget* (e.g.: iterations or simulations in Monte-Carlo planning), which also determines asymptotic convergence. This can be addressed by increasing the approximation budget or implementing more sample-efficient action selection techniques such as PGS.

2. If $\epsilon \to 0$ and $\omega^- \approx \tau N$, meaning the expected reward of the feature's actions is very close to the activation threshold τ. Keep in mind that an inactive feature also satisfies $\omega^+ < |\omega^-|$, so the reward of the *good* actions is in the interval $[0, |\tau|)$ suggesting this case is more likely when actions don't yield much reward. A simple way to address this issue is using a lower activation threshold, or perhaps even a variable threshold.

Additionally, if an action is not available (because its feature is inactive) and it is in fact optimal, then very likely no other feature-related action is available either (unless approximation errors occur in practice). A potential solution (apart from using a lower threshold) is to stochastically activate a feature, introducing an exploration pattern, or to limit the number of features that can be inactive, allowing an agent to always select among the most promising actions.

This analysis underlines the importance of a suitable activation threshold, which under appropriate circumstances will simplify a problem while preserving the inherent convergence properties of a planning algorithm. For example, in the coffee cup domain, "check" actions incur in a penalty of -0.5 but are essential to reach the goal, while picking up an empty cup yields -10 which can be avoided with sufficient planning. This suggests a value of τ such that $-10 < \tau \leqslant 0.5$. With a very simple understanding of the problem, such as the range of relevance values, a quick choice of τ can be made that quickly separates useful and less useful features (as in table 5.6). Without specific domain knowledge, the general recommendation is to choose a τ that is somewhat below the reward of expensive necessary actions but above other costly actions.

5.4 Relevance-based Online Planning

This section presents IRE algorithmically, building on the theoretical framework previously defined. IRE-based POMDP planning is a core element of our relevance-based online planning algorithm (alg. 5.1), where simulation is carried out by an IRE-enabled (or feature-aware) Monte-Carlo planner (line 3). During planning, matching entries in $\mathbb{T}$ receive updates to their feature-values and feature-action visit counts when their corresponding action is executed. After the simulation budget is exhausted, the best action is selected from the relevant action subset using feature-aware UCB1, or f-UCB1 (line 19 in algorithm 5.3), and upon execution its corresponding results are received. The POMDP simulator is updated to reflect these changes and

features are activated or deactivated based on their current relevance values (function BeliefRevision, in algorithm 5.2).

Algorithm 5.1 Online planning & acting with feature relevance

Input: Generative model $\mathcal{G}$, initial belief state b_0
Output: Policy π_f

```
1: procedure Online Planner
2:     repeat
3:         a ← f-MCTS(h)                          ▷ Relevance-aware
4:         Execute a                              ▷ "real-world" action
5:         Receive s', o, r
6:         Update G with a, o, r
7:         BeliefRevision(B(h))                   ▷ Feature (de)activation
8:     until s is terminal
9: end procedure
```

The BeliefRevision algorithm (alg. 5.2) is responsible for the main reduction in POMDP size, by activating and deactivating features accordingly every step, following the threshold-based activation rule in eq. 5.3.1. If all features are inactive a feature is randomly selected and activated. This prevents potentially useful actions from remaining locked or unavailable during simulation, in order to overcome the errors introduced from incorrect observations or insufficient planning.

The version of partially observable MCTS in algorithm 5.3 is called relevance-aware or feature-aware (f-aware) because, in addition to standard Monte Carlo, it computes all the values necessary to implement IRE. This is accomplished by approximating the feature-action function during simulation (line 43) alongside the action-value function, obtained from a second discounted return (line 37). Rollouts rely on a modified algorithm that uses only actions from active features (line 9), but standard UCB1 is used during simulation to guarantee a correct approximation through systematic sampling. Once the simulation budget is exhausted, a "real world" action is selected with an f-aware version of UCB1 (line 6). This is a lightly modified UCB1 (line 19) that guarantees that only actions from active features are executed. The properties of UCB1 are maintained but the action set is simplified to include only actions from relevant (active) features. As shown above, this corresponds to actions with high expected returns. This algorithm also uses PGS as a rollout policy, as in line 14, and as a PBRS bias in line 34. The combination of IRE and PGS constitutes the essence of relevance-based online planning, but both elements can operate independently: instead of PGS, IRE could utilize

Algorithm 5.2 Feature activation in the belief state

Input: belief state b
Output: revised belief state with relevant features active

```
1: function BeliefRevision(b)
2:     for all f ∈ F do
3:         Compute V(f) from every entry for f in T
4:         If V(f) < τ, deactivate f ∀s ∈ b
5:         Otherwise activate f ∀s ∈ b
6:         ∀⟨f, a⟩ N(f, a) ← 1
7:     end for
8:     If ∀f ∈ F are inactive, activate random f
9: end function
```

a standard policy resulting in IRE planning with uniformly random Monte Carlo rollouts. Without enough simulations, however, random selection will probably suffer from approximation errors.

We reiterate the feature activation rule and the scaling function were chosen to be as general as possible, and they can be easily substituted for others that fit specific domains better. An analysis of what constitutes an appropriate activation rule or scaling function might be useful, but we leave this open for future work.

5.5 Results

This section contains the performance results obtained from simulation experiments planning on two sets of complex POMDPs, reproduced from (Saborío and Hertzberg, 2019a). Our results show that an agent can considerably improve its performance in complex practical problems through feature-relevance estimation.

We used different configurations of the Cellar and the Big Foot domains, and compared the performance of IRE-based planning against standard POMCP. Naturally, POMCP's performance (just like any other planner) can be improved using manually designed heuristics, but we are interested primarily in what a planning agent can achieve without external interference. IRE and POMCP both ran with uniformly random MCTS (legal actions only) and with PGS. To the best of our knowledge there are no other online planners that can handle POMDPs this large with comparable

Algorithm 5.3 Relevance-aware Partially Observable MCTS

Input: history h, POMDP simulator $\mathcal{G}$
Output: action a

function f-MCTS(h)
  repeat
    $s \approx \mathcal{B}(h)$
    SIMULATE(s, h, 0)
  until *timeout*
  $a \leftarrow f$-UCB1(h)
  return a
end function

function f-ROLLOUT(s, h, d)
  if $d > d_{MAX} \vee s$ is terminal **then**
    return 0
  end if
  $\mathcal{A}_t = \{a \in A | f.a \text{ is active}\}$
  $a \leftarrow \arg\max_a \mathfrak{p}(s')$ s.t. $a \in \mathcal{A}_t$
  $s', o, r \leftarrow \mathcal{G}(s, a)$
  $R \leftarrow r + \gamma f$-ROLLOUT($s'$, hao, $d+1$)
  return R
end function

function f-UCB1(h)
  $\mathcal{A}_t = \{a \in A | f.a \text{ is active}\}$
  $a \leftarrow$ UCB1(h) s.t. $a \in A_t$
  return a
end function

function SIMULATE(s, h, d)
  if $d > d_{MAX} \vee s$ is terminal **then**
    return 0
  end if
  if $h \notin$ Tree **then**
    Add and initialize h
    return f-ROLLOUT(s, h, d)
  end if
  $a \leftarrow$ UCB1(h)
  $\{s', o, r\} \leftarrow \mathcal{G}(s, a)$

Immediate (r) and delayed (r^d) rewards:
  $r \leftarrow r + \gamma_{\mathfrak{p}} \alpha \mathfrak{p}(s') - \alpha \mathfrak{p}(s)$
  $\{r^d, r_f^d\} \leftarrow$ SIMULATE($s', hao, d+1$)

Compute discounted rewards:
  $R \leftarrow r + \gamma \cdot r^d$
  $R_f \leftarrow r + \gamma_f \cdot r_f^d$

Update belief state:
  $\mathcal{B}(h) \leftarrow \mathcal{B}(h) \cup \{s\}$

Increase visit counts:
  $N(h) \leftarrow N(h) + 1$
  $N(h, a) \leftarrow N(h, a) + 1$
  $N(f, a) \leftarrow N(f, a) + 1$

Update action and feature values:
  $Q(h, a) \leftarrow Q(h, a) + \frac{R - Q(h,a)}{N(h,a)}$
  $Q(f, a) \leftarrow Q(f, a) + \frac{R_f - Q(f,a)}{N(f,a)}$
  return $\{R, R_f\}$
end function

performance,[6] so we did not perform an exhaustive comparative analysis.

All results are averaged over 100 runs and use up to 2^{16} Monte-Carlo simulations per step, but in the plots we skip the results with very few simulations to better appreciate the region with significant differences. In practice, given the size of these problems, using more simulations might be desirable (and attainable due to the speedup we've achieved).

For testing we selected parameter values that showed promising results, so performance is limited by these choices and may be improved with further analysis for specific, real-life scenarios. This level of fine tuning is outside the scope of this thesis however, since we are attempting to show how performance scales with respect to problem size and not trying to solve specific problems.

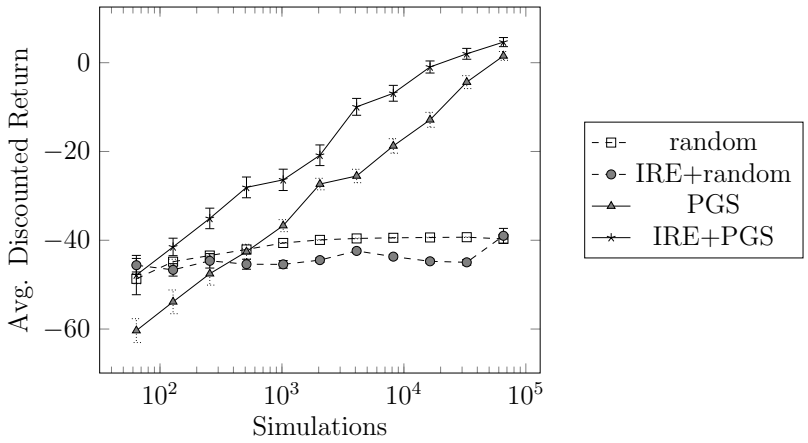
5.5.1 Cellar

We used three different configurations of the Cellar domain: cellar[5,2,6,4], cellar[7,8,7,8] and cellar[11,11,15,15]. The parameters are as follows:

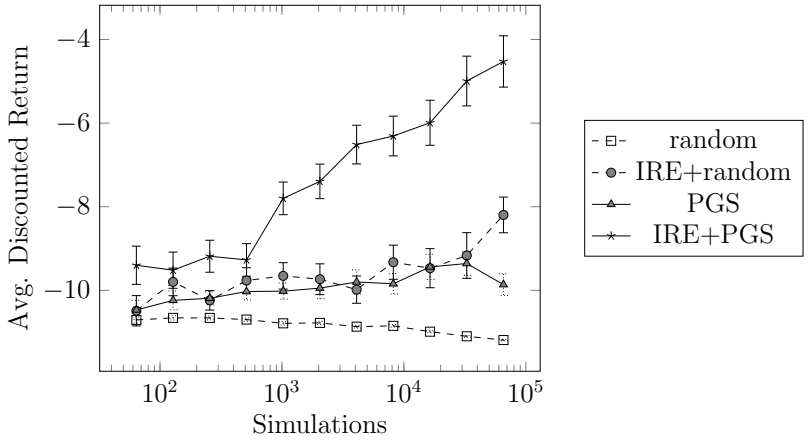
- $\mathcal{F}$ contains all the crates and shelves, but no bottles
- IRE discount factor $\gamma_f = 0.5$
- PGS binary entropy threshold $T_H = 0.5$
- cellar[5,2,6,4]:
  - Feature activation threshold $\tau = -6$
  - Discount factor $\gamma = 0.99$
- cellar[7,8,7,8] and cellar[11,11,15,15]:
  - Feature activation threshold $\tau = -5$
  - Discount factor $\gamma = 0.95$
- Maximum number of steps: 150, 250 and 350 for each size respectively

Figure 5.1 shows the average discounted returns in the cellar domain. All plots consistently show that IRE-based methods match or outperform non-relevance-aware planning, regardless of the chosen policy. PGS is already effective at gathering larger rewards using less simulations, but its performance can be greatly improved with relevance estimation.

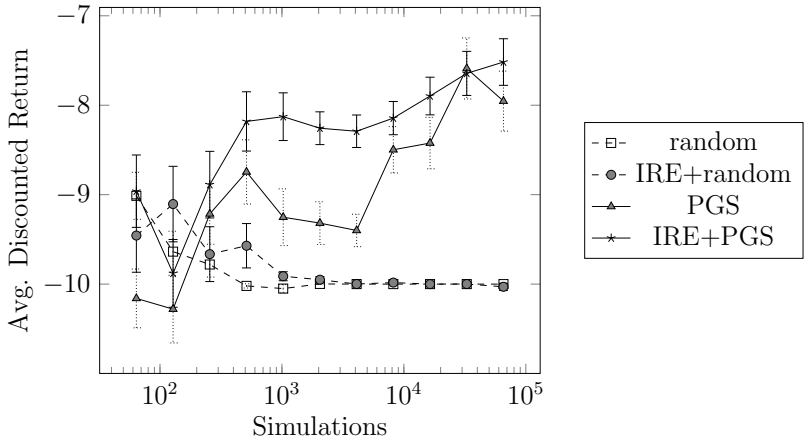
[6] Performance with DESPOT is very close to POMCP but uses heuristics



(a) Cellar[5,2,6,4]



(b) Cellar[7,8,7,8]



(c) Cellar[11,11,15,15]

Figure 5.1: IRE in Cellar
Source: (Saborío and Hertzberg, 2019a)

Cellar[5,2,6,4], the smallest of the three cellar layouts, is deceivingly complicated (see figure 5.2). This cellar layout contains "only" two valuable bottles but both are out of reach, so the agent must necessarily push at least one crate and avoid the crates that don't lead to bottles as well as every single shelf. This difficulty is reflected in the performance difference of fig. 5.1a, where PGS's advantage is exploited by IRE to achieve even higher rewards.

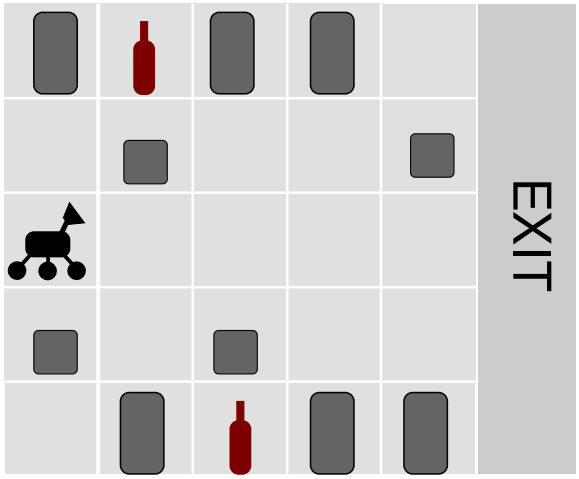


Figure 5.2: Cellar[5,2,6,4]

The medium-sized cellar[7,8,7,8] makes an excellent case for relevance-based planning: the IRE-enabled random policy manages to match simple PGS, and the performance of simple PGS is doubled when using IRE (fig. 5.1b).

The largest problem has many possible bottles to collect but several of them are easily accessible, so the challenge is to "simply" focus on the bottles while determining which (if any) non-bottle objects are useful. In this case, the performance difference of feature-relevance estimation is quite noticeable but not as dramatic. We could always construct further, realistic scenarios where pushing crates is required, and based on the results of cellar[5,2,6,4] we can anticipate a substantial difference in performance between traditional and relevance-based methods.

With respect to runtime, an important metric for online planning, IRE was consistently faster in all cases, with speedups of 1.7, 2.62 and 1.41 for the "small", medium and large cellar layouts respectively. Table 5.7 shows the runtimes, discounted returns and standard error when using the maximum budget.

5.5.2 Big Foot

In the Big Foot domain we also chose three different sizes to study scalability: BigFoot[3,3,2], BigFoot[4,3,2] and BigFoot[5,8,8]. The chosen parameters were:

- $\mathcal{F}$ contains every creature
- Feature activation threshold $\tau = -7$
- Discount factor $\gamma = 0.95$
- IRE discount factor $\gamma_f = 0.5$
- PGS binary entropy threshold $T_H = 0.4$, and $T_H = 0.5$ in BigFoot[5,8,8]
- A maximum of 150, 250 and 450 steps respectively

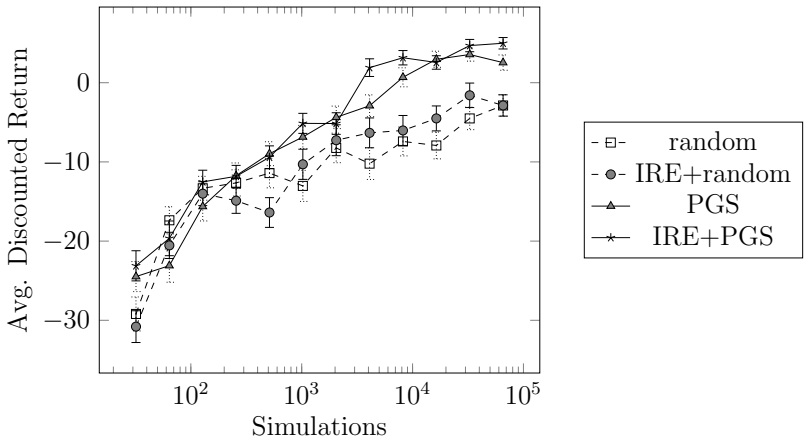
Figure 5.3 shows the average discounted returns in the three BigFoot problems.

Rewards in the Big Foot domain have very long delays, collected only after the agent locates, identifies and photographs valuable creatures. Consequently, the accuracy of relevance estimation is affected by the quality of the underlying policies. These challenges give incremental refinement an advantage over non-IRE methods, in some instances by a significant margin. As stated before, the Big Foot domain like any other difficult problem can always benefit from fine-tuning parameters but, as table 5.7 shows, IRE already provides a clear advantage in both cumulative reward and runtime. In BigFoot[3,3,2] IRE collected almost twice the reward of PGS very quickly, and a similar trend can be observed in BigFoot[4,3,2]. BigFoot[5,8,8] is considerably more demanding and this difference is reflected in runtimes and rewards, both of which were improved with IRE. We achieved speedups of 1.74, 1.66 and 1.21 respectively.

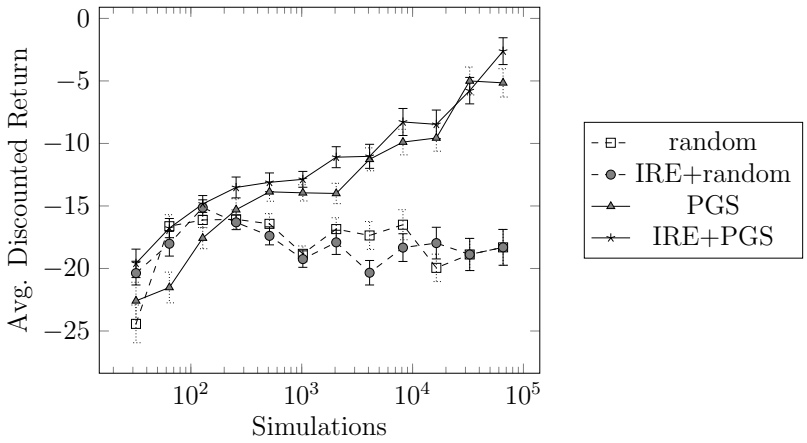
It is important to mention that by comparing IRE and PGS in these tests we are technically competing against ourselves. We have already established PGS alone already outperforms pure Monte Carlo, which performs and scales very poorly in these problems.

5.5.3 Runtimes

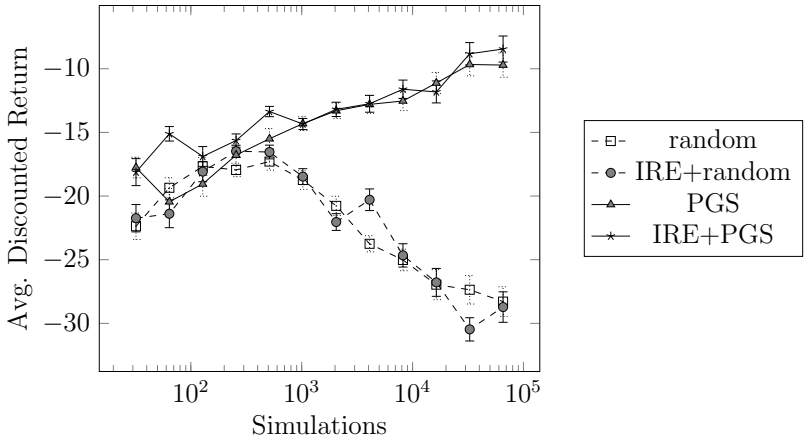
The purpose of IRE and relevance-based planning as a whole is to simplify complex domains. IRE in particular reduces the dimensionality of large POMDPs by *pruning* subsets of actions and states, allowing the agent to



(a) BigFoot[3,3,2]



(b) BigFoot[4,3,2]



(c) BigFoot[5,8,8]

Figure 5.3: IRE in Big Foot
Source: (Saborío and Hertzberg, 2019a)

focus on *relevant* transitions. The intended effect is, therefore, solving large POMDPs faster. Our experimental results have already showed that IRE, particularly when using PGS, consistently and substantially outperforms standard Monte Carlo in terms of returns. Note that these runtimes correspond to complete episodes with many consecutive actions; planning for a single action requires only a fraction of the time. In practice, online planning concerns itself only with providing the next best action.

Uniform random sampling provides good long-term convergence guarantees due to its systematic approach, and UCB is known to balance exploration and exploitation optimally. This process is often very slow however, so many large problems are still out of reach for online planning. Table 5.7 shows that in addition to better returns, planning with IRE also improves response times due to the agent planning over a POMDP with reduced complexity, as an effect of deactivating irrelevant features. We compiled the average runtimes and standard error of the two best performing policies in each problem, both of which are ours: simple PGS and full IRE (IRE with PGS) with 2^{16} simulations over 100 runs. The table shows that IRE consistently collects higher rewards in considerably less time than PGS alone.

Table 5.7: IRE – Returns and runtime
Source: (Saborío and Hertzberg, 2019a)

Problem	**IRE**	**Disc. Return**	**Time** (s.)
Cellar[5,2,6,4]	✓	4.645 ± 1.02	175.1
	✗	1.532 ± 0.99	310
Cellar[7,8,7,8]	✓	-4.525 ± 0.61	2169
	✗	-9.864 ± 0.26	5693
Cellar[11,11,15,15]	✓	-7.517 ± 0.26	9029
	✗	-7.954 ± 0.33	12790
BigFoot[3,3,2]	✓	4.983 ± 0.72	20.45
	✗	2.539 ± 0.97	35.54
BigFoot[4,3,2]	✓	-2.619 ± 1.07	129.8
	✗	-5.154 ± 1.14	215.5
BigFoot[5,8,8]	✓	-8.45 ± 1.02	2264
	✗	-9.711 ± 0.96	2735

Just like with PGS, computing relevance values incurs in some additional computational cost. As we showed this cost is more than justified given the performance and episode runtimes, but nonetheless we compared IRE-based planning with uniformly random Monte Carlo using equivalent resources. Table 5.8 shows the performance and runtime of the random policy using the

maximum number of simulations, as well as the performance of the closest equivalent runtime using IRE+PGS.

Table 5.8: IRE – Runtime comparison with random MCTS

Problem	**Policy**	**Disc. Return**	**Time** (s.)
Cellar[5,2,6,4]	IRE+PGS	4.645 ± 1.02	175.1
	Random	-39.69 ± 0.059	1108
Cellar[7,8,7,8]	IRE+PGS	-5.992 ± 0.54	658.1
	Random	-11.19 ± 0.017	653
Cellar[11,11,15,15]	IRE+PGS	-8.144 ± 0.185	1298
	Random	-10 $\pm 2.6 \times 10^{-8}$	1218
BigFoot[3,3,2]	IRE+PGS	3.155 ± 0.91	4.44
	Random	-2.872 ± 1.35	3.55
BigFoot[4,3,2]	IRE+PGS	-11.03 ± 0.96	28.93
	Random	-18.31 ± 1.43	29.32
BigFoot[5,8,8]	IRE+PGS	-12.75 ± 0.66	218.1
	Random	-28.29 ± 1.168	172.8

We can see the random policy is generally fast except in cellar[5,2,6,4], a straightforward problem that nevertheless requires an elaborate policy. However, even with similar amounts of time, IRE consistently outperforms random Monte Carlo in cumulative return. One of the most important considerations, beyond specific runtimes, is how poorly uniform random sampling scales when given more resources (such as more simulations or planning time). IRE and PGS utilize additional resources effectively and manage to improve their performance, but in our tests in complex POMDPs the random policy (and the heuristic policy as discussed in the previous chapter) failed to do so even with very large planning budgets.

5.5.4 Discussion

This section presented the experimental results of our approach to relevance-based planning. With IRE we were able to efficiently solve very large POMDPs, even when they require long and complex policies as in the Big Foot domain. The uniformly random rollout policy does not scale well in these environments, and as seen before in the Cellar domain, neither does a heuristic policy. IRE was capable of improving not only runtimes, which is expected when planning over a smaller POMDP, but also returns, possibly as a result of filtering out less useful actions and states.

In a couple of cases it seemed like the purely random rollout policy performed better than random with IRE. It is important to point out that relevance estimation is only as good as the underlying action-value estimation, so it is possible that the poor performance of the random policy in these challenging POMDPs induces a very large error in relevance estimation, and consequently results in poor choices of features to activate and deactivate. With PGS, a more sample efficient rollout policy, IRE manages to collect larger rewards in considerably less time. Naturally, PGS is an essential component of our relevance-based planning approach.

In other cases the mean discounted return decreased with more simulations using the random policy. We already discussed the limitations of random sampling in domains this large, but we will add that the planning budget might also have an effect in the perceived performance of the random policy. It is possible that the agent is more likely to select more useful and "expensive" actions when a more generous planning budget is available, instead of executing only "cheaper", less useful actions. Given the restrictions of our experiments, however, the agent still runs out of time. Nevertheless, in a few cases IRE managed to improve performance despite the limitations of the random rollout policy. An exemplar case is cellar[7,8,7,8], in fig. 5.1b, where IRE with random outperformed simple PGS.

Given the size of these problems and the total runtime of the experiments, gathering statistics was limited to batches of 100 episodes. It is possible that this number might affect the observed variance in returns, but it should be noted that feature deactivation did not incur in a significant change in standard error. This suggests that the stability properties of the underlying numerical estimation are preserved.

The intuition behind IRE originally came from noticing that the typical POMDPs used as planning benchmarks tend to resemble games and not real-life robot planning scenarios. These types of POMDPs, unlike practical problems, contain exclusively the elements necessary to reach the goal, without distractions of any kind. Additionally, they tend to have actions that incur in no cost and don't cause the agent to spend resources such as time or energy (a reason we presented our results using discounted returns). Rocksample is one such problem: the agent must collect rocks but it is presented with a world where only rocks exist.

The Cellar domain was our first attempt to model a more realistic planning problem, and can be understood as an extension of Rocksample with additional (non-goal related) objects and a more punishing reward distribution. Optimally simplified, Cellar becomes barely more complex than Rocksample, which is (as we explained) a relatively simple problem. The challenge for the planning agent is focusing only on useful objects and not getting distracted,

so that it may transform its planning domain into something that resembles Rocksample plus a few crates and shelves.

Big Foot is a generalization of this class of POMDPs and a good example of a practical problem with distractions and costly actions, where in addition there is no *a priori* knowledge about the types of objects and their location is not only unknown, but also variable and determined by external factors. In Big Foot it is crucial to quickly locate, identify and exploit reward opportunities because they might become unavailable at any moment.

Both problems contain actions where the agent incurs in some amount of punishment (e.g.: using the scanner, pushing crates, identifying creatures) that must nonetheless be executed to achieve the goal conditions. This generates complex policies where the agent must realize that a small short-term penalty is necessary to maximize the expected return in the long-term. Like in the real world, these actions are never *free*, so it becomes necessary for an online planning agent to quickly focus on the interactions that justify their cost and divert its attention from the ones that do not lead to satisfactory terminal states. As shown, this is crucial to achieve reasonable performance when planning online in complex, variable domains, where both purely random sampling and static heuristics are insufficient.

5.6 Contributions, Related Work and Outlook

IRE is a relevance-estimation method for online POMDP planning that approximates feature relevance by a weighted combination of action values. Features that satisfy the threshold criteria are considered relevant, and the rest are ignored by the planner. Our theoretical results show that there exists a threshold value that separates valuable features from those with overall low expected rewards, effectively simplifying large problems. The experimental results confirm that this approach significantly improves runtime and expected returns, without the use of detailed domain knowledge such as heuristics or pre-constructed prior knowledge in the form of action hierarchies or dependencies for state factoring. Efficient, online POMDP planning opens up many opportunities in a variety of challenging scenarios that combine exploration, manipulation and information-gathering in dynamic environments.

We have chosen to address the dimensionality challenge of very large POMDPs by allowing the planning agent to establish internal criteria to simplify its own available transition model. Historically, state and belief state dimensionality has been managed by a variety of methods that range

from grid-based to point-based approximations (such as those reviewed in chapter 2). However none of these methods exploit the underlying structure of problems, instead focusing on a general abstraction of POMDPs. As previously stated, action hierarchies are known to provide abstractions that also improve performance in POMDPs (Vien and Toussaint, 2015), but they must be built in advance whereas our proposal works entirely online and does not rely on prior knowledge.

Machine Learning, in particular techniques such as deep neural networks, is another direction to construct abstraction criteria internally, but learning POMDPs is particularly tricky since the agent does not have access to the true state and must work based only on observations. Additionally all learning frameworks require extensive data and resources for training before they reach a usable state. There are however some modern successful approaches to learning POMDPs. For instance, in (Igl et al., 2018) they focus on reactive decision problems in video games (problems without much deliberation), where partial observability is added using screen flickering and is not an intrinsic component of the domain or of the effect of actions. In IRE, relevance estimation is affected by the results of observations and the current expectations of feature and action values. Another example is (Karkus et al., 2017), where they attempt to learn a planning model and solve small navigation tasks modeled as POMDPs (mazes in the order of 10^4 states, whereas our tests scaled to 10^{30}). Underneath, their approach relies on a naive value iteration algorithm that (as stated by the authors themselves) suffers in high dimensional domains, which is where a more sophisticated planning algorithm is truly necessary. Learning and planning are complementary but only one can be addressed at a time. We focused on the challenge of deliberating over the long-term effect of actions in very large state spaces by estimating what we dubbed "feature relevance", doing it entirely online with minimal human input or prior domain knowledge.

Existing research in POMDP state factoring, such as (Feng and Hansen, 2014) and (Ong et al., 2010), is also not satisfactory from our perspective. The practical problems we are interested in cannot typically be conveniently factored, so tools for planning directly over unfactored representations are necessary. The resulting simplification provided by IRE, however, could perhaps be seen as somewhat similar to state factoring: the resulting POMDP, with deactivated features, essentially contains clues about how features and their values are affected, and states with similar active features may be considered similar regardless of their deactivated features. An important difference is that, unlike IRE, state factoring techniques assume independence of state variables and require a prebuilt dependency model as well as a *good* choice of basis functions. IRE is perhaps best seen as an online pruning

method, that limits which states are reachable based on relevance estimation.

Continuous POMDPs may be seen as a special case of very large POMDPs. Recent developments such as DESPOT-α (Garg et al., 2019), GPS-ABT (Seiler et al., 2015) and POMCPOW (Sunberg and Kochenderfer, 2018) attempt to improve sampling in domains with very large or continuous action and observation spaces in which standard MCTS is inefficient (although POMCP can manage continuous domains to an extent). These domains, however, constitute navigation and motion planning problems where low-level control is required. In contrast, we addressed high-level task-planning problems (such as Cellar and Big Foot), which correspond to very large and complex POMDPs that don't necessarily have very large action and observation spaces, since reasoning is conducted primarily on qualitative aspects of the world. We argued that the complexity of practical problems comes from the combinatorial explosion generated by (even manageable) observation spaces when many elements are present in a planning domain, and this is where relevance-based planning becomes necessary. All of these approaches might be complementary though, since they address different levels of abstraction. An important distinction is that, unlike IRE and PGS, all of these methods depend on heuristics.

With IRE and the resulting relevance-aware online planning algorithm, we have provided a method to solve very large POMDPs efficiently and showed that a simple choice of parameters can produce very promising results. Together, IRE and PGS constitute a formal approach to relevance that an agent can use to develop and refine pruning criteria and action preferences internally, without the need for a detailed prior analysis of the domain or expertly designed heuristics. Naturally, as is the case with any conceivable planning problem, a set of useful, carefully designed heuristics can possibly help improve performance even further, but we have made an effort to make this a choice and not a necessity.

There are some open topics that might warrant further research. For instance, experimenting with different relevance scaling functions and feature activation policies might produce even better results in some domains. Possible choices include exploiting histories, general prior knowledge or using, at least partially, some predefined preference criteria (e.g.: relevance hints from experts). Perhaps the most productive application of this framework is partially observable task-planning onboard robots. Our theoretical and experimental results already show what improvements can be expected, but developing and implementing a full-fledged POMDP planner for mobile robots comes with multiple challenges that lie out of the scope of this thesis.

Chapter 6

Planning and Acting Under Uncertainty

We have presented a relevance-based approach to online planning that allows an agent to address large problems by developing internal simplifying criteria. Techniques such as PGS and IRE not only increase the expected return, but also decrease the expected runtime when planning for individual actions and, as a result of this, improve the overall efficiency. As previously stated, we expect to ultimately contribute to improving the performance of mobile robots planning in difficult domains, but a practical implementation is out of the scope of this thesis. As a closing argument, however, we offer a preliminary proposal towards the integration of our probabilistic planning framework onboard robots.

In this chapter we present two candidate algorithms and an introductory analysis that expands upon the integration of planning methods with delayed action results, as is the case in most mobile robot platforms. These suggestions intend to offer an efficient utilization of hardware resources, such as memory and CPU, and minimize wall-clock time[7] when planning online.

6.1 Interleaving Planning and Acting

Interleaved planning and acting is already at the core of online planning, which concerns itself only with the next action. In order to determine the

[7]Wall-clock time corresponds to the duration of an event as perceived by a person, as opposed to the computing time allocated within a system's processing units.

next best action, the planner uses prior information such as the observation received after executing the previous action and estimates the current (belief) state. This process alternates between action planning and action execution, and continues until the agent determines it has reached a satisfactory terminal condition.

Mobile robots have different kinds of actions available that allow them to interact with the world and obtain information about it, similar to the agent solving the Cellar and the Big Foot domains. From the perspective of planning, possible actions might include driving towards a specific point in space, using laser range scanners or RGB-D cameras (plus the corresponding data processing) and engaging grippers or arms to attempt to grasp or move objects. All of these actions have an array of possible results, ranging from a simple failure (e.g.: to grasp) to noisy or unreliable sensor information. Going full circle, the uncertain and probabilistic nature of these problems is precisely why we focused on POMDP planning.

Planning online onboard a mobile robot is relatively straightforward, and follows the aforementioned action-observation cycle. We have also established that task-planning relies on relatively abstract, high-level actions, such as *grasp* or *scan*. The physical execution of such actions takes time, so the planner must wait until the results or outcome have been received before moving on to plan for the next action. In very few cases does it make sense to construct a full plan or even a partial plan several steps ahead, given the number of possible outcomes (as well as their probabilities). This does not mean, however, that the computing resources devoted to planning have to be wasted during action execution. Once an action is recommended by the planner and the robot attempts to execute it, there is a window of opportunity roughly equal to the duration of the action where the agent may continue to plan, in order to reduce the expected planning time for the next action. This is illustrated in fig. 6.1, where the top half corresponds to standard online planning, and the bottom half to overlapping action planning with execution.

Because of the action-observation dependency of online planning, this requires assuming or guessing some given outcome out of a probability distribution (e.g.: the distribution of observations $\omega \in \Omega$ in POMDPs). We propose an optimistic solution in which outcomes are sampled from the underlying generative model, following the problem's observation distribution. However, not all observations are possible after executing some given action, nor are all state transitions possible for a given initial state. We are therefore concerned with the action-specific observation space and state space, that is, the sets of possible observations and states, respectively, with respect to some action a. The observation space after executing an action is:

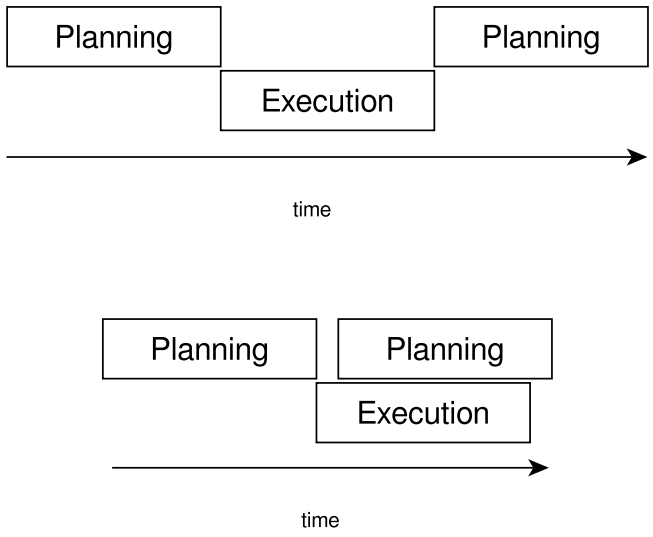


Figure 6.1: Overlapped action planning and execution

$$\Omega_a = \{\omega | \omega \in \Omega \ \wedge \ p(\omega|s,a) > 0\} \tag{6.1.1}$$

And the state space for action a is:

$$S_a = \{s | s \in S \ \wedge \ p(s'|s,a) > 0\} \tag{6.1.2}$$

After selecting an action a, the agent is allowed to continue planning with the assumption that an observation ω_a randomly drawn from Ω_a will match the observation $\omega \in \Omega_a$ received from the outside world.

There is a probability equivalent to $p(\omega_a|s,a)$ that $\omega = \omega_a$, in which case the assumption is correct and the planning done in advance is useful. If $\omega \neq \omega_a$, the work done is simply discarded and the correct observation is used to plan for the next action, as would be the case in standard online planning.

In practice, a planning algorithm is normally given a budget in terms of the number of (Monte-Carlo) simulations or planning time per action. Let T_p be the maximum time budget for planning for one action, and T_a the expected duration of executing action a. Normally, one step in the action-observation cycle would last:

$$T_{a\omega} = T_p + T_a \tag{6.1.3}$$

By sampling a probable observation, we're proposing to reduce the expected step duration to:

$$\mathbb{E}(T_{a\omega}) = p(\omega_a|s,a)\max\{T_p, T_a\} + (1 - p(\omega_a|s,a))\,(T_p + T_a) \tag{6.1.4}$$

Intuitively, this is simply planning ahead for a likely result using background resources. For example, a person playing a game such as chess has a maximum amount of time to plan before committing to an action, and once executed they have to wait for the second player to do the same. In this case the direct results are deterministic, but the second player's response is not so the first player can utilize this *idle* time by planning ahead for a likely outcome.

In the next section we provide two suggestions of algorithms that integrate the idea of sampling likely scenarios into our relevance-based planning approach.

6.2 Planning with Delayed Action Results

This section provides a formal definition of the proposed method for planning with delayed observations sketched above. In POMDPs, anticipating the outcome of actions means obtaining or sampling observations and resulting states from their respective distributions. This is easy to represent in planning algorithms that rely on a generative model.

Given the POMDP simulator $\mathcal{G}$, it is possible to simulate the transition $T(s,a,s')$ and receive an observation ω and reward r as is done in simulation and rollouts during planning. For the following, let's assume a generative model $\mathcal{G}$ and the function $\mathcal{G}.T(h,a)$, that simulates the aforementioned transition for a history h and an action a.

Once an action is found by the planner, the agent attempts to execute it and the time window for pre-planning begins. It is then that an observation is sampled and a separate, independent planner initiates a new search using a separate, independent generative model. This is necessary to avoid the conflicts that may arise in concurrent computing environments, such as deadlocks and data races.[8] This induces a series of small modifications to the MCTS planning algorithm, which is now expected to maintain an execution status that indicates whether the search is complete, as shown in alg. 6.1.

[8] A *deadlock* is a situation that arises when multiple computing threads or processes are blocked due to an unsatisfiable condition, usually related to resource utilization. A *data race* occurs when multiple processes require access to a variable and at least one of them attempts to write, possibly destroying or invalidating previous and future uses of its value.

Algorithm 6.1 Status dependent search
Input: Generative model $\mathcal{G}$, history h
Output: Action a

```
1: function GetAction(G, h)
2:     if MCTS is ready then
3:         return a ← MCTS.SelectBestAction(G, h)
4:     else
5:         return a ←MCTS(G, h)
6:     end if
7: end function
```

Status dependent MCTS performs a standard search in the normal, online planning case, but takes advantage of an ongoing search if available. If the search is complete and the status is "ready", instead of searching it simply requests the best action (selected with feature-aware UCB1 for the current history).

We propose two algorithms that attempt to exploit pre-planning by sampling observations. The first one uses one core MCTS planner and one background MCTS planner, each with their own respective generative models $\mathcal{G}$ and $\mathcal{G}_2$. Alg. 6.2 begins by searching for an action with the underlying MCTS planner, and once the action is determined it is sent to the robot control system for execution without blocking the planning process. Meanwhile a resulting state, observation and reward are sampled from the background generative model $\mathcal{G}_2$, which should at this point be equivalent to $\mathcal{G}$, and a new tentative history h_2 is generated based on ω_2 and used for planning. Once the real results arrive, the real observation ω is compared with the sampled observation ω_2 and, if they match, the complete or partial search is preserved. Otherwise, the background search is discarded and the generative models are updated to reflect the real outcomes. In the new iteration we can better understand the purpose of GetAction: if the background search was preserved and planning was complete, a new action can be safely chosen immediately. Otherwise, the search may begin or continue until the planning budget is exhausted.

As discussed before, the probability that the background planner is correct corresponds to the probability of the observation sampled. This is an optimistic approach that speeds up planning for the next action with probability $p(\omega|s, a)$, and otherwise behaves like a standard online planner. We can also, in principle, increase the probability of planning on a correct observation by sampling more than one and pursuing multiple execution paths.

Algorithm 6.2 Planning with delayed results: Dual Planners

Input: Generative models $\mathcal{G}, \mathcal{G}_2$, initial belief state b_0
Output: Policy π_f

procedure ONLINE DUAL-PLANNER
  repeat
    $a \leftarrow \text{GETACTION}(\mathcal{G}, h)$
    Execute a without blocking

  Parallel process:
    timeout $= \max\{\mathbb{E}(T_a), T_p\}$
    $s'_2, \omega_2, r_2 \leftarrow \mathcal{G}_2.T(h, a)$
    Update $\mathcal{G}_2$ with a, ω_2, r_2
    $h_2 \leftarrow h \cup \{a\omega_2\}$
    MCTS$(\mathcal{G}_2, h_2)$ with timeout

  Parent process:
    Receive s', ω, r from outside world
    if $\omega = \omega_2$ **then**
      $\mathcal{G} \leftarrow \mathcal{G}_2$
      $h \leftarrow h_2$
    else
      Update $\mathcal{G}$ with a, ω, r
      $\mathcal{G}_2 \leftarrow \mathcal{G}$
    end if
    $\text{BELIEFREVISION}(\mathcal{B}(h))$
  until s is terminal
end procedure

The second algorithm (alg. 6.3) constitutes a generalization of the Dual Planners approach that uses one core generative model and many background models that together constitute the set $\mathbb{G} = \{\mathcal{G}_C, \mathcal{G}_1, \ldots, \mathcal{G}_n\}$, as well as many planners concurrently executed on many independent processes.

The Multiple Planners approach starts by searching for an action with the underlying MCTS planner, and communicating this action to the robot control system. While it executes, the set of generative models is processed in parallel following a sequence of steps similar to the previous algorithm: first the transition for the current action is simulated and its results obtained, the models are updated with the sampled observation, the histories are updated to reflect this transition, and multiple searches are initiated each using a different model and starting history, and restricted to the preallocated budget. Once the results of the original action arrive, the model from the set that matches the real world observation is chosen, and the core planner and history are updated accordingly. If no such model exists, then the core planner and every planner in the set are updated following the external observation. In the next iteration, a new search is started if no progress could be made in advance (no planning was done with the correct observation). Otherwise, if one of the sampled observations was correct and planning was carried out with such a model, then the search is either underway or ready and a new action can be selected quickly or immediately depending on the circumstances.

With these algorithms or variations of them, the physical resources onboard a mobile robot can be exploited to speed up the response time when planning online. In domains with fewer observations and enough computing resources, perhaps each element in the action-specific observation space Ω_a could be mapped directly to a process to guarantee pre-planning for every possible outcome, but in general this is not advisable as it is likely that the amount of observations will outnumber the available computing resources. In addition, more refined sampling techniques could be used to ensure observations are unique.

6.3 Challenges and Outlook

The ideas presented in this chapter are not required to implement a fully functional relevance-based planner as presented in the previous sections; they are a suggestion to better utilize the computing hardware onboard a mobile robot. Implementing the multiplanner approach requires, as is to be expected, physical hardware with enough computing resources. The dual planner solution can be implemented sequentially if the background process

Algorithm 6.3 Planning with delayed results: Multiple Planners

Input: Generative model set $\mathbb{G}$, core generative model $\mathcal{G}$, initial belief state b_0

Output: Policy π_f

procedure ONLINE MULTI-PLANNER
  repeat
    $a \leftarrow$ GETACTION$(\mathcal{G}_C, h)$
    Execute a without blocking

  Parallel block:
    timeout $= \max\{\mathbb{E}(T_a), T_p\}$
    for all $\mathcal{G} \in \mathbb{G}$ **do**
      $\mathcal{G}.s', \mathcal{G}.\omega, \mathcal{G}.r \leftarrow \mathcal{G}.T(h, a)$
      Update $\mathcal{G}$ with $a, \mathcal{G}.\omega, \mathcal{G}.r$
      $\mathcal{G}.h \leftarrow h \cup \{a\mathcal{G}.\omega\}$
      MCTS$(\mathcal{G}, \mathcal{G}.h)$ with timeout
    end for

  Parent process:
    Receive s', ω, r from outside world
    if $\exists \mathcal{G} \in \mathbb{G}$. $\mathcal{G}.\omega = \omega$ **then**
      $\mathcal{G}_C \leftarrow \mathcal{G}$
      $h \leftarrow \mathcal{G}.h$
    else
      Update $\mathcal{G}_C$ with a, ω, r
      $\forall \mathcal{G} \in \mathbb{G}$. $\mathcal{G} \leftarrow \mathcal{G}_C$
    end if
    BELIEFREVISION$(\mathcal{B}(h))$
  until s is terminal
end procedure

is designed to both search and expect the incoming results. The multiplanner algorithm also requires minimal synchronization and it should scale well with the number of available resources, as long as there are no memory limitations. The underlying parallel computing model presented here is *coarse-grain*: independent search processes are assigned to processing units. There is some progress in *fine-grain* parallel MCTS however, including a parallel version of DESPOT (Cai et al., 2018).

Due to the restrictions of current computing hardware, among which weight and energy consumption are important factors, it might be important to follow at least some of the suggestions in this chapter to more efficiently interleave planning and acting onboard mobile robots. The fundamental improvement in POMDP task-planning comes as a result of PGS and IRE, but a more efficient utilization of physical resources can never be underestimated.

Chapter 7

Conclusions

In this thesis we have presented a formal approach to relevance estimation in planning, and we have shown that it allows an agent to perform better and faster in challenging domains where traditional algorithms scale very poorly. This chapter presents our closing arguments and discusses the integration of online POMDP planning into robot control systems, one of our main motivations. We offer some recommendations and provide suggestions for future work by summarizing the open problems from the previous chapters.

7.1 The Role of Relevance in Planning and Acting

One of the core principles behind this work, addressed in numerous ways throughout the dissertation, is the application of *relevance* to planning algorithms. Intuitively it is not hard to understand that a concept or entity is *relevant* if it is appropriate in a given context, and that *relevance* is an estimation of its degree or the intensity of the connection between the concept (or entity) and said context. Based on the definition we provided early on, we framed the idea of relevance in a way that is applicable for planning, problem solving and decision-making. Our hypothesis was that focusing on *relevant* features, an agent can improve its performance solving challenging problems.

There is no doubt automated planning is challenging. Particularly in POMDPs, the dense action sets require extensive amounts of computation to separate useful from less useful actions. With IRE and PGS we have

attempted to grant a planning agent the resources to speed up this process, by exploiting the underlying structure of problems and utilizing information that is already part of the planning domain. We reviewed previous work that attempted to simplify the underlying transition model but ours does it by constructing its own criteria internally, therefore reducing the dependency on human experts as well as improving the scalability of planning in scenarios that are not only large but also complex, with potentially long or intricate policies with delayed rewards.

Undernearth these methods lies the principle of *attention*. It is perhaps naive to claim that even a complex problem is easy if we can "focus on the things that matter", but it is undeniably true. The challenge is, of course, correctly identifying which "things" really do matter, and then focusing the majority of resources in them. As stated throughout the thesis, an action selection algorithm such as UCB1 eventually converges to optimal actions, meaning it focuses on the few truly useful actions. Through relevance-based planning, the agent can reach this point faster, essentially focusing its attention on the most promising features and their corresponding actions.

Lastly, granting an agent focus and biasing attention towards particular features generates a rich array of interesting types of behavior, and perhaps some elements of the underlying process (such as feature activation derived from feature-action value estimation) may also contribute to an important element in planning and acting under uncertainty: explanation and justification. In IRE the justification for actions and the relevance of features comes from their expected return, which parallels their contribution to the goal. It is possible, therefore, that with some additional work a numerical framework like IRE could be used to provide not just a policy but also explanations to grant context and justification to an agent's behavior. For example, if a mobile service robot operating in a wine cellar is asked to bring some wine (but not one specific bottle), and it returns with a sufficiently good bottle the problem is completed successfully. However, perhaps the person expected a particularly good or even a specific bottle but the robot did not collect it. The robot could then claim that it did find this specific bottle, but it was out of reach and the resources required to collect it far exceeded other more easily accessible bottles (e.g.: it would have taken twice the amount of time). Without specific constraints, this is a perfectly reasonable explanation for such type of satisficing behavior. In the end, evaluating the correctness and soundness of plans depends not only on the specific actions taken, but also on the state of the environment and the history of actions as well as their expected outcomes, which gives rise to context and explanation.

We have provided methods to improve the efficiency of planning by exploiting information available in the transition model, corresponding to a

formal approach to relevance. We expect this framework will lead to further refinements in automated planning and acting, especially if combined with complementary methods that address additional levels of abstraction in decision-making onboard artificial agents.

7.2 Towards Full-scale POMDP Planning Onboard Robots

Throughout the document we have referred to agents that are capable of executing relatively complex actions, from navigation to information gathering to grasping and manipulation. The planning algorithms we proposed operate at the level of *tasks*, or relatively abstract, high-level actions. The "physical" (so to speak) counterpart of such agents are mobile robots with sensors, manipulation devices and local computing resources, often referred to as "cognitive robots". As mentioned in previous sections, there are examples of robots operated by planning on POMDPs, but these are heavily pre-processed, factored POMDPs with a heavy dependence on expert analysis.

Relevance-based planning onboard robots is, in principle, no more difficult than implementing any other type of planner which also assumes the existence of prior information such as a (semantic) map and reliable localization. In the simplest form a POMDP planner simply provides an action based on the expected results of the modeled planning domain, the robot attempts to execute it and communicates the results back to the planner. This requires an interface to communicate observations and actions to and from the planner, as well as a system capable of translating high-level actions from the planning domain (such as "pick up bottle") to the corresponding low level operators (possibly engaging motion planners). In addition both the Cellar and the Big Foot problems, despite representing plausible challenges, assumed the existence of a relatively reliable system for sensor information processing such as classification and object recognition. A complete solution for general robot control using POMDP task-planning requires access to such components, since it operates at the level of actions, observations and their effect over probability distributions. Naturally a POMDP can also represent low-level control operations and continuous domains, but that is a different line of research.

Robot control does not have to be exclusively dependent on POMDP planning, however. We can conceive two scenarios for a relatively smooth integration of deterministic planning and POMDP task-planning onboard

robots. The first is as a fallback system that addresses emergent probabilistic problems that are complex but very specific. For instance, a deterministic planner finds a suitable plan which is executed successfully up to a certain point, after which it fails leaving the robot unable to decide due to a lack of information. The POMDP planner takes control and, through a combination of manipulation and information gathering actions, brings the robot back to a state recognized by the deterministic planner. In the second scenario, a deterministic planner and a POMDP planner coexist and work independently but with a certain degree of synchronization. The actions recommended by either planner can be assessed and weighted differently depending on the context: the POMDP planner is well suited for planning with partial information and uncertainty and integrating different observations, while a deterministic planner is fast and effective, as long as the environment is very well known in advance.

Finally, because of the stochastic nature of the planning domains of interest, it should be noted that when relying on a probabilistic planner a small margin of error is to be expected. In other words, certain small mistakes should be expected and allowed, as long as they don't involve dangerous actions. Any kind of risky behavior can be addressed either with a more detailed transition model or directly at the lower level of control (perhaps even through hardware). Let's remember that in comparison, a human being solving similar stochastic and partially observable problems would perform much worse than the planner.

7.3 Future Directions

Based on the theoretical and experimental results we can conclude that relevance-based online planning offers a substantial advantage to a planning agent, particularly in complex domains. There are still open topics in both action selection and dimensionality reduction, however.

PGS requires computing points for several derived states and this makes it comparatively slower than random rollouts, albeit with better results. In domains that require more complex solutions, such as cellar[5,1,0,4] and larger, PGS does perform much better than all competing policies but a more efficient point counting strategy should be designed in order to improve its applicability in extremely large problems. The response time when using PGS was indirectly addressed with IRE due to the reduction in dimensionality, but a more drastic speed up could possibly be obtained by optimizing the underlying PGS point computation.

Relevance estimation and dimensionality reduction work quite well with

IRE, but due to the nature of the dissertation we did not thoroughly research the effect of different feature activation policies or feature scaling functions. As previously mentioned, these functions could be chosen to represent the perceived relevance criteria of experts, generating a reduced POMDP that responds to particular needs and interests.

Finally, the core motivation for relevance-based planning was granting an agent independence and autonomy in its decision-making process, contributing to more scalable algorithms. This does not mean that domain knowledge should be completely avoided; our position is that it shouldn't be *necessary* because, among other reasons, it doesn't scale well. In very challenging problems, especially those with very high associated risk or cost, a combination of heuristic and relevance-based planning will probably offer the best possible results. Prior knowledge has traditionally been integrated into planning through some form of pre-existing knowledge representation, but we suspect that as applications grow in complexity a more dynamic interface will become necessary. The efficient integration of prior knowledge (or external suggestions) dynamically into an online POMDP planner constitutes an interesting area of research, that might contribute to more reliable planning systems through human-machine collaboration.

Bibliography

Abramson, B. D. (1987). *The Expected-outcome Model of Two-player Games*. PhD thesis, Columbia University, New York, NY, USA.

Åström, K. J. (1965). Optimal control of Markov Processes with incomplete state information. *Journal of Mathematical Analysis and Applications*, 10:174–205.

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

Barto, A. G., Bradtke, S. J., and Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1):81 – 138.

Bertsekas, D. P. (2005). *Dynamic Programming and Optimal Control*, volume I. Athena Scientific, Belmont, MA, USA, 3rd edition.

Bonet, B. (2002). An epsilon-optimal grid-based algorithm for partially observable markov decision processes. In *Proceedings of the Nineteenth International Conference on Machine Learning*, ICML '02, pages 51–58, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Bonet, B. and Geffner, H. (2003). Labeled RTDP: improving the convergence of real-time dynamic programming. In *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS 2003), June 9-13, 2003, Trento, Italy*, pages 12–21.

Bradtke, S. J. and Barto, A. G. (1996). Linear least-squares algorithms for temporal difference learning. *Machine learning*, 22(1-3):33–57.

Cai, P., Luo, Y., Hsu, D., and Lee, W. S. (2018). HyP-DESPOT: A hybrid parallel algorithm for online planning under uncertainty. In *Robotics:*

Science and Systems XIV, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, June 26-30, 2018.

Cassandra, A. R., Kaelbling, L. P., and Littman, M. L. (1994). Acting optimally in partially observable stochastic domains. In *Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 2.*, pages 1023–1028.

Cassandra, A. R., Littman, M. L., and Zhang, N. L. (1997). Incremental pruning: A simple, fast, exact method for partially observable markov decision processes. In *UAI '97: Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence, Brown University, Providence, Rhode Island, USA, August 1-3, 1997*, pages 54–61.

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303.

Eck, A., Soh, L.-K., Devlin, S., and Kudenko, D. (2016). Potential-based reward shaping for finite horizon online POMDP planning. *Autonomous Agents and Multi-Agent Systems*, 30(3):403–445.

Feng, Z. and Hansen, E. (2004). An approach to state aggregation for POMDPs. In *AAAI-04 Workshop on Learning and Planning in Markov Processes–Advances and Challenges*, pages 7–12.

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

Feng, Z. and Hansen, E. A. (2014). Approximate planning for factored POMDPs. In *Sixth European Conference on Planning*.

Fox, D., Burgard, W., Dellaert, F., and Thrun, S. (1999). Monte carlo localization: Efficient position estimation for mobile robots. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence, July 18-22, 1999, Orlando, Florida, USA.*, pages 343–349.

Garg, N. P., Hsu, D., and Lee, W. S. (2019). DESPOT-α: Online POMDP planning with large state and observation spaces. In *Robotics: Science and Systems XV, Freiburg, Germany, June 22-26, 2019*.

Ghallab, M., Nau, D., and Traverso, P. (2016). *Automated Planning and Acting*. Cambridge University Press, San Francisco, CA, USA. (to be published).

Hanheide, M., Göbelbecker, M., Horn, G. S., Pronobis, A., Sjöö, K., Aydemir, A., Jensfelt, P., Gretton, C., Dearden, R., Janicek, M., Zender, H., Kruijff, G.-J., Hawes, N., and Wyatt, J. L. (2017). Robot task planning and explanation in open and uncertain worlds. *Artificial Intelligence*, 247:119 – 150. Special Issue on AI and Robotics.

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

Hauskrecht, M. (2000). Value-function approximations for partially observable markov decision processes. *J. Artif. Int. Res.*, 13(1):33–94.

Hester, T. and Stone, P. (2013). TEXPLORE: Real-time sample-efficient reinforcement learning for robots. *Machine Learning*, 90(3).

Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30.

Hsu, D., Lee, W. S., and Rong, N. (2007). What makes some POMDP problems easy to approximate? In *Proceedings of the 20th International Conference on Neural Information Processing Systems*, NIPS'07, pages 689–696, USA. Curran Associates Inc.

Hutter, M. (2016). Extreme state aggregation beyond markov decision processes. *Theoretical Computer Science*, 650(Supplement C):73 – 91. Algorithmic Learning Theory.

Igl, M., Zintgraf, L., Le, T. A., Wood, F., and Whiteson, S. (2018). Deep variational reinforcement learning for POMDPs. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2117–2126, Stockholmsmässan, Stockholm Sweden. PMLR.

Karkus, P., Hsu, D., and Lee, W. S. (2017). QMDP-Net: Deep learning for planning under partial observability. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 4694–4704. Curran Associates, Inc.

Kearns, M., Mansour, Y., and Ng, A. Y. (2002). A Sparse Sampling Algorithm for Near-Optimal Planning in Large Markov Decision Processes. *Mach. Learn.*, 49(2-3):193–208.

Kim, S., Salzman, O., and Likhachev, M. (2019). POMHDP: search-based belief space planning using multiple heuristics. In *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2018, Berkeley, CA, USA, July 11-15, 2019.*, pages 734–744.

Kocsis, L. and Szepesvári, C. (2006). Bandit based monte-carlo planning. In *In: ECML-06. Number 4212 in LNCS*, pages 282–293. Springer.

Konidaris, G. (2016). Constructing abstraction hierarchies using a skill-symbol loop. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 1648–1654.

Kurniawati, H., Hsu, D., and Lee, W. S. (2008). SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In Brock, O., Trinkle, J., and Ramos, F., editors, *Robotics: Science and Systems*. The MIT Press.

Kushmerick, N., Hanks, S., and Weld, D. (1994). An algorithm for probabilistic least-commitment planning. In *AAAI-94 Proceedings*, pages 1073–1078.

Li, X., Cheung, W. K. W., Liu, J., and Wu, Z. (2007). A novel orthogonal nmf-based belief compression for POMDPs. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 537–544, New York, NY, USA. ACM.

Lovejoy, W. S. (1991). Computationally feasible bounds for partially observed markov decision processes. *Operations Research*, 39(1):162–175.

McMahan, H. B., Likhachev, M., and Gordon, G. J. (2005). Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, pages 569–576, New York, NY, USA. ACM.

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

Ng, A. Y., Harada, D., and Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *In Proceedings of the Sixteenth International Conference on Machine Learning*, pages 278–287. Morgan Kaufmann.

Ong, S. C. W., Png, S. W., Hsu, D., and Lee, W. S. (2010). Planning under uncertainty for robotic tasks with mixed observability. *Int. J. Rob. Res.*, 29(8):1053–1068.

Pajarinen, J. and Kyrki, V. (2017). Robotic manipulation of multiple objects as a POMDP. *Artificial Intelligence*, 247:213 – 228. Special Issue on AI and Robotics.

Peng, J. and Williams, R. J. (1993). Efficient learning and planning within the dyna framework. In *Adaptive Behavior*, pages 437–454.

Pineau, J., Gordon, G., and Thrun, S. (2003a). Policy-contingent abstraction for robust robot control. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence*, UAI'03, pages 477–484, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Pineau, J., Gordon, G. J., and Thrun, S. (2003b). Point-based value iteration: An anytime algorithm for POMDPs. In *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pages 1025–1032.

Pineau, J., Gordon, G. J., and Thrun, S. (2006). Anytime point-based approximations for large POMDPs. *Journal of Artificial Intelligence Research*, 27:335–380.

Poupart, P. and Boutilier, C. (2002). Value-directed compression of POMDPs. In *Proceedings of the 15th International Conference on Neural Information Processing Systems*, NIPS'02, pages 1579–1586, Cambridge, MA, USA. MIT Press.

Ross, S. and Chaib-draa, B. (2007). AEMS: an anytime online search algorithm for approximate policy refinement in large POMDPs. In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 2592–2598.

Rummery, G. A. and Niranjan, M. (1994). On-line Q-learning using connectionist systems. Technical Report TR 166, Cambridge University Engineering Department, Cambridge, England.

Saborío, J. C. and Hertzberg, J. (2019a). Efficient planning under uncertainty with incremental refinement. In *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019, Tel Aviv, Israel, July 22-25, 2019*, page 112.

Saborío, J. C. and Hertzberg, J. (2019b). Planning under uncertainty through goal-driven action selection. In van den Herik, J. and Rocha, A. P., editors, *Agents and Artificial Intelligence*, pages 182–201, Cham. Springer International Publishing.

Saborío, J. C. and Hertzberg, J. (2017). Practical assumptions for planning under uncertainty. In *Proceedings of the 9th International Conference on Agents and Artificial Intelligence - Volume 2: ICAART,*, pages 497–502. INSTICC, SciTePress.

Saborío, J. C. and Hertzberg, J. (2018). Towards domain-independent biases for action selection in robotic task-planning under uncertainty. In *Proceedings of the 10th International Conference on Agents and Artificial Intelligence - Volume 2: ICAART,*, pages 85–93. INSTICC, SciTePress.

Seiler, K. M., Kurniawati, H., and Singh, S. P. N. (2015). An online and approximate solver for POMDPs with continuous action space. In *IEEE International Conference on Robotics and Automation, ICRA 2015, Seattle, WA, USA, 26-30 May, 2015*, pages 2290–2297.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T. P., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

Silver, D., Sutton, R. S., and Müller, M. (2012). Temporal-difference search in computer go. *Machine Learning*, 87(2):183–219.

Silver, D. and Veness, J. (2010). Monte-Carlo Planning in Large POMDPs. In *In Advances in Neural Information Processing Systems 23*, pages 2164–2172.

Singh, S. P., Jaakkola, T., and Jordan, M. I. (1995). Reinforcement learning with soft state aggregation. In Tesauro, G., Touretzky, D. S., and Leen, T. K., editors, *Advances in Neural Information Processing Systems 7*, pages 361–368. MIT Press.

Singh, S. P. and Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. *Mach. Learn.*, 22(1-3):123–158.

Smallwood, R. D. and Sondik, E. J. (1973). The optimal control of partially observable markov processes over a finite horizon. *Operations Research*, 21(5):1071–1088.

Smith, T. and Simmons, R. (2004). Heuristic Search Value Iteration for POMDPs. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, UAI '04, pages 520–527, Arlington, Virginia, United States. AUAI Press.

Somani, A., Ye, N., Hsu, D., and Lee, W. S. (2013). DESPOT: Online POMDP planning with regularization. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 26*, pages 1772–1780. Curran Associates, Inc.

Spaan, M. T. J. and Vlassis, N. A. (2005). Perseus: Randomized point-based value iteration for POMDPs. *J. Artif. Intell. Res.*, 24:195–220.

Strehl, A. L., Diuk, C., and Littman, M. L. (2007). Efficient structure learning in factored-state MDPs. In *AAAI-07*, pages 645–650.

Sunberg, Z. N. and Kochenderfer, M. J. (2018). Online algorithms for POMDPs with continuous state, action, and observation spaces. In *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018, Delft, The Netherlands, June 24-29, 2018.*, pages 259–263.

Sutton, R., Precup, D., and Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 216–224. Morgan Kaufmann.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA, 2nd edition.

Sutton, R. S., Szepesvári, C., and Maei, H. R. (2008). A convergent O(n) temporal-difference algorithm for off-policy learning with linear function

approximation. In *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8-11, 2008*, pages 1609–1616.

Tesauro, G. (1995). Temporal difference learning and TD-Gammon. *Commun. ACM*, 38(3):58–68.

Theocharous, G. and Mahadevan, S. (2010). Compressing POMDPs using locality preserving non-negative matrix factorization. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*.

Thiébaux, S. and Hertzberg, J. (1992). A semi-reactive planner based on a possible models action formalization. In *Artificial Intelligence Planning Systems: Proceedings of the First International Conference (AIPS92)*, pages 228–235. Morgan Kaufmann.

Thrun, S. (2000). Probabilistic algorithms in robotics. *AI Magazine*, 21(4):93.

Vien, N. A. and Toussaint, M. (2015). Hierarchical monte-carlo planning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 3613–3619.

Watkins, C. J. (1989). *Learning from Delayed Rewards*. PhD thesis, King's College, Oxford.

Zhou, R. and Hansen, E. A. (2001). An improved grid-based approximation algorithm for POMDPs. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'01, pages 707–714, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.