UNIVERSIDADE DE LISBOA

INSTITUTO SUPERIOR TÉCNICO

Information Gain and Value Function Approximation in Task Planning Using POMDPs

Tiago Santos Veiga

Supervisor: Doctor Matthijs Theodor Jan Spaan

Co-Supervisor: Doctor Pedro Manuel Urbano de Almeida Lima

Thesis approved in public session to obtain the PhD Degree in Electrical and Computer Engineering

Jury final classification: Pass With Distinction

2015

UNIVERSIDADE DE LISBOA
INSTITUTO SUPERIOR TÉCNICO

Information Gain and Value Function Approximation in Task Planning Using POMDPs

Tiago Santos Veiga

Supervisor: Doctor Matthijs Theodor Jan Spaan
Co-Supervisor: Doctor Pedro Manuel Urbano de Almeida Lima

Thesis approved in public session to obtain the PhD Degree in
Electrical and Computer Engineering

Jury final classification: Pass With Distinction

Funding Institutions

Fundação para a Ciência e Tecnologia (FCT) through the PhD Studentship SFRH/BD/70559/2010, through ISR/LARSys strategic fundings PEst-OE/EEI/LA0009/2013 and UID/EEA/50009/2013, and the research grant at ISR in the Strategic Project RD0461/IST/ISR.

2015

Título: Ganho de Informação e Aproximação da Funcão de Valor em Planeamento de Tarefas Usando POMDPs
Nome: Tiago Santos Veiga
Doutoramento em: Engenharia Electrotécnica e de Computadores
Orientador: Doutor Matthijs Theodor Jan Spaan
Co-Orientador: Doutor Pedro Manuel Urbano de Almeida Lima

Resumo:

Esta tese foca-se no planeamento em ambientes dinâmicos e móveis. Em particular em tarefas de percepção activa cooperativa (ACP), onde sensores estáticos e activos devem cooperar para melhorar a informação perceptual disponível no sistema enquanto, possivelmente, executam outros tipos de tarefas.

A tarefa de planeamento é formalizada como um processo de Markov parcialmente observável (POMDP), dado que fornece um método matematicamente sólido e permite modelar as incertezas no sistema.

Tipicamente, o objectivo em ACP é o ganho de informação, mas, na sua formulação tradicional, POMDPs são optimizados para reduzir a incerteza apenas se for benéfico para a execução de tarefas. Apresenta-se um método para modelar a tomada de decisão sob incerteza para ganho de informação, extendendo POMDPs para recompensar crenças com baixa incerteza e mantendo funções de valor convexas e lineares por troços. Apresenta-se avaliação experimental e um caso de estudo de vigilância com robôs.

No entanto, POMDPs são difíceis de solucionar e a sua aplicação em sistemas reais continua limitado por problemas de escalabilidade. Introduz-se aproximação linear da função de valor em POMDPs. Desenvolvem-se algoritmos de iteração de valor com funções de valor lineares e propõe-se uma abordagem para a construção automática de funções de base que explora a estrutura de modelos factorizados. A performance dos algoritmos é avaliada em problemas ilustrativos.

Palavras-chave: Planeamento Sob Incerteza; Processos de Markov Parcialmente Observáveis; Percepção Activa Cooperativa; Sistemas de Robôs Ligados em Rede; Ganho de Informação; Vigilância Assistida por Rôbos; Funções de Base; Escalabilidade; Função de Valor Linear; Iteração de Valor Aproximada.

Title: Information Gain and Value Function Approximation in Task Planning Using POMDPs

Abstract:

This thesis focuses on planning in dynamic and mobile environments. In particular in active cooperative perception (ACP) tasks, in which static and active sensors must cooperate in order to improve the perceptual information available to the system, while possibly performing other tasks.

The planning task is formalized as a partially observable Markov decision process (POMDP), as it provides a sound mathematical framework and allows for modelling the uncertainties in the system.

Typically, the goal in ACP is to perform information gain as a task itself, but traditional POMDP formulation is optimized to reduce uncertainty only if it helps task performance. We present a POMDP framework to model decision-making under uncertainty for information gain. POMDPs are extended to reward low-uncertainty beliefs while remaining with piecewise linear value functions. Experimental evaluation is provided and a case study for robotic surveillance is presented.

However, POMDPs are hard to solve and their applicability in real systems is still limited by scalability issues. This thesis introduces linear value function approximation in POMDPs. We develop value iteration algorithms with linear value functions and propose an approach for automatic basis function construction which exploits factored model structure. We experimentally evaluate its performance in illustrative problems.

Key-words: Planning Under Uncertainty; Partially Observable Markov Decision Processes; Active Cooperative Perception; Networked Robot Systems; Information Gain; Robot-Assisted Surveillance; Basis Functions; Scalability; Linear Value Function; Approximate Value Iteration.

Acknowledgements

Each person who passes through our life passes alone but does not leave alone, neither leave us alone, because he leaves a little of himself and takes a little bit of us.

Antoine Saint-Exupéry

Life is full of adventures, but none can be completed by ourselves and I am grateful to everyone that left a bit of themselves with me during these years.

First of all, to my supervisors, who have been and still are an example to me and to whom I am grateful for the opportunity I had. To Matthijs Spaan, for his guidance and support, for welcoming so well during my short stay in Delft, and for his humorous and easygoing manner. Also, a special word to Professor Pedro Lima, for his valuable opinions and advice, allways with a smile to everyone.

I would like to remember all the members the IRSGroup, in particular everyone with whom I shared the room 6.15. Thanks for all the companionship, for all the scientific discussion, for the good leisure times and good coffee breaks.

One important lesson that I have learned during these years was inspired by Andrey Markov, and what I like to call a Markovian inspired life: no matter what we have done so far we should always try to get the best ahead of us. Many people helped me realize that and I would also like to remember all of them. The list would be too big to fit here but I am thankful for all my friends who shared their time with me and taught me valuable lessons. In particular, I would like to remember everyone who shared their time with me at GASTagus hoping to change the world. We might not have changed the world yet, but we definitely have changed ourselves.

To all my family, for being so supportive and patient with me. To my sister, allways ready to be helpful. To my niece and nephew, for bringing me joy. To my parents for all they taught me during my childhood and made me what I am today. Without them none of this was possible.

Lastly, but definitely not least, to Catarina who found me during this journey and always made me believe in myself. You bring out the best in me, and I am now looking forward to our next adventures. I am grateful that I have met you.

Contents

List of Figures **vii**

List of Tables **ix**

1 Introduction **1**

1.1 Planning . . . 2

1.2 Information-Gathering . . . 3

1.2.1 Networked Robot Systems . . . 4

1.2.2 Active Cooperative Perception . . . 4

1.3 Complexity . . . 5

1.4 Contributions . . . 6

1.5 Thesis Outline . . . 7

2 Decision-theoretic planning **8**

2.1 Markov Decision Processes . . . 8

2.1.1 Value Iteration . . . 12

2.1.2 Policy Iteration . . . 13

2.1.3 Linear Value Function Approximation . . . 13

2.2 Partially Observable Markov Decision Processes . . . 16

2.2.1 Value Function . . . 18

2.2.2 Value Iteration . . . 19

2.2.3 Exact value iteration . . . 20

2.2.4 Approaches to Approximation . . . 21

2.2.4.1 Model approximation . . . 22

2.2.4.2 Policy approximation . . . 22

2.2.4.3 Value function approximation . . . 22

2.2.5 Point-based value iteration . . . 23

2.2.6 PERSEUS . . . 25

2.3 Model Representation . . . 25

2.3.1 Factored models . . . 26

2.3.2 Algebraic Decision Diagrams . . . 28

2.3.3 Factored linear value functions . . . 29

3 Active Perception with POMDPs **31**

3.1 Information Gain in POMDPs . . . 31

3.1.1 Running example: PATROL . . . 32

3.2 POMDPs with Information Rewards 33
3.2.1 Models for Active Perception 34
3.2.2 Rewarding Low-Uncertainty Beliefs 35
3.2.3 Choosing the Information Rewards Parameters 36
3.2.4 Extensions and Variations 38
3.3 Experiments 39
3.3.1 Information Rewards in the PATROL Example 39
3.3.1.1 Model 39
3.3.1.2 Results 39
3.3.1.3 Discussion 42
3.3.2 Comparison With ρPOMDPs on the ROCK DIAGNOSIS Problem . 42
3.3.2.1 ROCK DIAGNOSIS problem definition 43
3.3.2.2 Experimental Setup 44
3.3.2.3 Results 45
3.4 Case-study: Robot-Assisted Surveillance 46
3.4.1 Model and Experimental Setup 48
3.4.2 Experiments with Information Rewards 49
3.4.3 Discussion 51
3.5 Related Work 51
3.5.1 POMDPs in a robotic context 51
3.5.2 Active sensing 52
3.5.3 POMDP frameworks for rewarding information gain 53
3.6 Discussion 54

4 Value Function Approximation **56**
4.1 Related Work 56
4.1.1 Linear Value Function Approximation 57
4.1.2 Basis Function Construction 57
4.1.2.1 MDPs 57
4.1.2.2 POMDPs 58
4.2 Linear Value Function Approximation Extension to POMDPs 59
4.3 Point-Based Methods With Linear Value Function Approximation 62
4.4 Belief-based Projection Methods 65
4.5 Automatic Basis Function Construction 67
4.6 Exploiting Factored Observation Spaces 71
4.7 Experiments 73
4.7.1 Model Description 73
4.7.1.1 Network Management 74
4.7.1.2 Fire Fighting 76
4.7.1.3 Experimental Setup 76
4.7.2 Comparing Basis Functions 77
4.7.3 Factored Observations 78
4.7.4 Comparing Projection Methods 80
4.7.5 Basis Function Construction 81
4.7.6 Scaling POMDP Solving 82
4.8 Discussion 85

5 Conclusions and Future Work **87**
5.1 Conclusions 87
5.2 Future work 89

Bibliography **91**

List of Figures

1.1 Diagram of an agent. The agent live in an environment which is at a particular state, it receives information about the state and acts in the environment according to the information received [1]. 2
1.2 Example of a robot cooperating with a camera network for (a) person localization and (b) person identification. 4

2.1 Diagram of a MDP agent interacting with the environment 9
2.2 Diagram of a POMDP agent interacting with the environment 16
2.3 Example of a Value Function with two states. 19
2.4 Example of a Dynamic Bayesian Network which encodes the states and transitions of a MDP problem [2]. 27
2.5 Example of how to encode the reward function of a MDP in a Dynamic Bayesian Network [2]. 28

3.1 Patrol example. (a) Illustration of the problem, showing a corridor of length 3, a robot which has to travel between two goal nodes (marked by stars) and an alarm device in the middle. (b) Dynamic Bayesian Network representation of the problem with state factors Y = *robot position*, C = *alarm color* and G = *current goal*. The shaded nodes indicate the POMDP-IR additions to the model. 33
3.2 Patrol problem results. Each figure shows belief evolution (top row), robot position (2nd row), true alarm color (3rd row) and received rewards (bottom row). Corridor length is 3 except for (d). 40
3.3 Performance measures for Patrol problem with 5 alarms, and l *commit* actions with $\beta = 0.9$. 41
3.4 Rock Diagnosis computation times. ρPOMDP timings are taken from [3] and hence cannot be compared directly. 45
3.5 Experimental setup. (a) Robot Pioneer 3-AT with camera and laser range finder onboard. (b) Example of an image captured by the camera network. (c) Feature detection by the robot. 47
3.6 Two-stage dynamic Bayesian network representation of the proposed POMDP-IR model. In this figure we assume $m \equiv k$ for simplicity, i.e., each person has only one feature. 47
3.7 Environment map with observation model learned for fixed cameras and predefined model for robot camera. Each matrix is positioned at each node in the map, and represents $p(O^F|F)$, that is, probabilities of positive (o_f) or negative ($o_{\bar{f}}$) observations given the true state of the feature. . . . 47

3.8 POMDP-IR simulation results for the case study. Experiments A and B: $\beta = 0.75$. Experiments C and D: $\beta = 0.6$. Action $commit_1$ corresponds to asserting that the person is wearing red while action $commit_2$ asserts the opposite. 50

4.1 Example of a vector representation with basis functions with two states. . 60
4.2 Example DBN with basis. 61
4.3 Projection Errors example. α is the original vector and $\widetilde{\alpha}$ the approximated one. ϵ_s is the error minimized in state-based projection, while ϵ_v is the error minimized in belief-based projections. 65
4.4 Network management problem. (a)(b) Illustration of the problem, showing different configurations. Shaded node represents the server machine and arrows represent connections in the network.(c) Dynamic Bayesian network representation of the problem. State factors X_i represent the state of each machine i. 74
4.5 Fire fighting problem.(a) Illustration of the problem. Squares represent agents, while circles represent houses. Each agent can act in one single house per timestep, but it can move freely between houses. (b) Dynamic Bayesian network representation of the problem. State factors H_i represent the fire level at each house i. For visualization purposes we show the dynamics of the problem for a single action $A = (1, 3)$, meaning that Agent 1 goes to house 1 and agent 2 goes to house 3. 75
4.6 Fire fighting problem. (a) Basis functions tested. (b), (c) Comparison of different basis functions. 77
4.7 Two-stage DBN for the modified fire fighting problem with one observation factor for each state factor, where $X_i = \{0, 1, \ldots, k-1\}$; $O_i = \{0, 1, \ldots, l-1\}$. 79
4.8 Factored observations (log-scale y-axis) 79
4.9 Policy quality comparison with Krylov Iteration in *3legs* network problem with 7 machines (128 states). SVD Basis is our method with fixed number of basis functions (8 basis for this problem) 82
4.10 Average and standard deviation time to perform a backup operation (Algorithm 5) in the network management problem with n machines. 83
4.11 Network management problem with increasing number of machines and *cycle* (a),(d),(f) and *3legs* (b),(c),(e) topologies. Comparison with Symbolic Perseus and its extension with linear value function approximation. . 84
4.12 Fire fighting problem with 2 agents and increasing number of houses. Comparison with Symbolic Perseus and its extension with linear value function approximation. 85

List of Tables

2.1 Outline of point-based algorithms, with the associated collection and updating method [4] . . . 25

3.1 Example rewards for varying β. . . . 38
3.2 Rock Diagnosis results, comparing POMDP-IR with ρPOMDP [3]. . . 44

4.1 Problem sizes. $|S|$, $|A|$, $|O|$, represent, respectively, the number of states, actions and observations for each problem. . . . 76
4.2 Size n_h of set of basis functions found automatically with our method. . . 82
4.3 Time comparison with Krylov Iteration in *3legs* network problem with 7 machines (128 states). SVD Basis is our method with fixed number of basis functions (8 basis for this problem) . . . 82

List of Algorithms

1 MDP Value iteration . . . 13
2 MDP Policy Iteration . . . 13
3 General Point-Based POMDP Solver . . . 24
4 Perseus . . . 26
5 Backup Operator with ADDs, $backup(b, V)$ . . . 29
6 Krylov Iteration (R,$T^{a,z}$) . . . 58
7 Truncated Krylov Iteration (R,$T^{a,z}$,k,ϵ) . . . 59
8 Point-Based Backup With Linear Value Function Approximation . . . 63
9 Point-Based Backup With Linear Value Function Approximation, Optimized For ADDs . . . 64
10 Automatic Construction of Basis Functions . . . 71

Chapter 1

Introduction

Making decisions is at the heart of autonomous task performance, but decisions are essentially difficult to make. Even we humans many times have many doubts when it comes to take decisions, and often based on intuitive strategies in opposition to theoretically sound reasoning rules. Thus, intelligent decision support systems add some reasoning to the intuitive based human thinking [5].

In the meantime, technology research led to new applications where autonomous decision making in challenging environments plays an essential role. For instance, in robotics research history, areas of application have gone from static industrial environments to more dynamic and human-inhabited environments. Nowadays, we may find driverless cars, interactive museum tour-guides robots [6], collaborative robots that perform service tasks for humans [7], among several other examples. In these scenarios the planning process involves being aware of the changes in the environment and act according to those changes in order to perform some given tasks. However, in real problems there is uncertainty associated with sensors and actuators. For instance, a robot must take into account on its planning process that its sensors are noisy and it may fail to observe some event in the environment. Also, its actions can have uncertain effects. We will look at the planning problem in the presence of uncertainty from an artificial intelligence perspective, in order to compute sequence of actions which guides the system to desired states. Typically, a goal of a planning problem is to reach some goal state. We will consider a broad definition of goal, such that it can include some desired level of information about the environment.

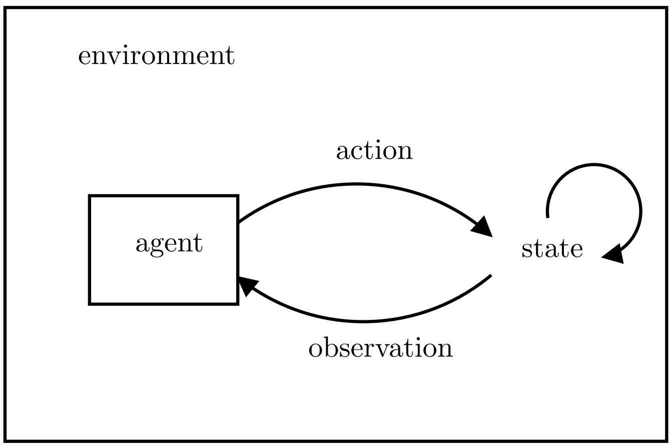


Figure 1.1: Diagram of an agent. The agent live in an environment which is at a particular state, it receives information about the state and acts in the environment according to the information received [1].

1.1 Planning

A planning problem in artificial intelligence is usually described as follows: given a description of the current state of some system, a set of actions that can be performed on the system and a description of a goal set of states for the system, find a sequence of actions that can be performed to transform the system state into one of the goal states [8]. An agent is a general concept, which may include robots and intelligent computer programs. Agents interact with the environment as shown in Figure 1.1. In planning agents are considered as acting in an environment which is at a particular state at every moment. The agent may interact with the environment by acting within it, and therefore influences how the state changes. It chooses which actions to execute dependent on the current state of the environment [9].

In the case of deterministic environments the agent would plan a sequence of actions, given that each action would certainly produce the desired outcome. However, in practice the outcome of actions and the information that the agent receives from the environment are not certain. Therefore, when planning for real world problems, inevitably, an agent must deal with uncertainty. For instance, in our diagram of an agent's operation, if the environment where the agent lives is stochastic, i.e., there are several different possible outcomes for each action, then the agent must plan ahead how to act upon each different possible next state.

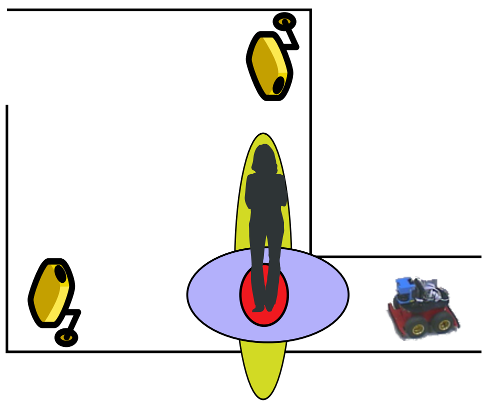
In the field of artificial intelligence (AI) planning has been tackled for a long time. Classic AI planning algorithms focus on the task of finding a sequence of actions to take a system to a goal state in controlled, static environments, but are no longer applicable when we take agents out of controlled environments. Decision-theory [10] provides a framework for evaluating multiple plans under uncertainty (i.e, with uncertain outcomes), based on probability theory and utility theory. However, decision-theory does not tackle the problem of planning, which under this framework would be formalized as the task of constructing a plan with maximum expected utility. From this, both frameworks were merged [11] in order to model planning under uncertainty. In another context, Markov Decision Processes (MDPs) became popular in the operations research community. In a MDP, there is a reward associated with each possible world state, and the goal of an agent is to compute the plan that maximizes the long-term cumulative reward.

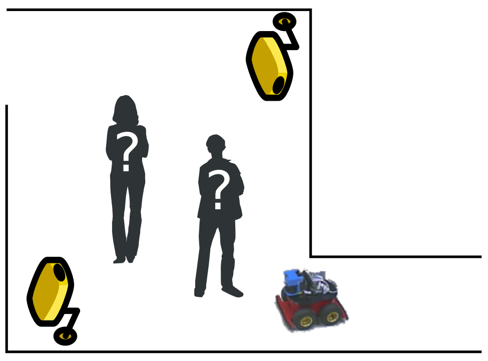
MDP-based techniques have been used to model decision making in several problems. Under this framework, we model decision making in stochastic environments, where outcomes of actions are uncertain. When we extend it to problems where we add uncertainty to what we observe, we enter into the field of Partially Observable Markov Decision Processes (POMDPs) [12, 13]. POMDPs are modeled in a similar way as MDPs, with a reward associated with every state. But in POMDPs we also take into account uncertainty in sensing, and we no longer know the true state of the environment at every moment. POMDPs have been used in problems where agents need to reason over uncertainty, in numerous applications, such as robot navigation [14], controlling the temperature of chemical sensors [15] or in customer relationship management [16], among others [17].

POMDPs establish a sound mathematical framework to model decision-making under uncertainty in several problems. POMDPs provide a principled way to model the interaction between agents and the environment, and allows to directly model uncertainties associated with the system, as encoding the goal of a particular task. With a model of the environment, we may compute policies that tell agents how to act based on the observations the system receives.

1.2 Information-Gathering

So far, we described the planning goal as reaching some goal states. Even when planning under uncertainty, the system typically will act in a way to reduce its uncertainty about the environment only if that improves task performance. However, in some systems information gain can be an objective itself and planners should plan for reaching some particular information state. Multi-objective problems can combine both objectives and plan balance regular task objectives with information objectives.


(A) Person localization



(B) Person identification

Figure 1.2: Example of a robot cooperating with a camera network for (a) person localization and (b) person identification.

1.2.1 Networked Robot Systems

An example of challenging environments where planning is crucial for task performance and information objectives play an important role are the scenarios of networked robot systems (NRS) [18], defined as any distributed system which consists of a multitude of networked robots and other devices and which, as a whole, is capable of interacting with the environment through the use of perception and actions for the performance of tasks [19]. A system of networked mobile and static sensors can substantially improve situational awareness compared to a single sensor or a network of static sensors. However, the benefit of mobile sensors is maximized if actions, such as positioning a mobile sensor, are carefully planned and executed [20]. In our work, we consider the problem of planning in NRS, in which mobile robots carrying sensors interact with each other as well as with static sensors present in the environment to accomplish certain tasks [21] (as illustrated in Fig.1.2). For instance, in a shopping mall, we can consider a NRS where cameras detect humans in need of help, but also detect a fire eruption or an abnormal activity which requires assistance. Robots might be used both to improve the confidence of event detection and to provide assistance for any of the above situations.

1.2.2 Active Cooperative Perception

We take a comprehensive approach to this problem, denoted here as *active cooperative perception* (ACP) [22]. In our context, *cooperative perception* refers to the fusion of sensory information among the static surveillance cameras and each robot, with the goal of maximizing the amount and quality of perceptual information available to the system. *Active* perception means that an agent considers the effects of its actions on its sensors, and in particular it tries to improve their performance. This can mean selecting sensory actions, for instance pointing a pan-and-tilt camera or choosing to execute a computationally expensive vision algorithm. Other effects might require reasoning about the future, such as adjusting a robot's path planning: given two routes to reach a goal location, take the more informative one, for instance. Combining the two concepts, active cooperative perception is the problem of active perception involving multiple sensors and multiple cooperating decision makers.

There are many benefits of cooperation between sensors, in particular when some are mobile. An obvious advantage is that a mobile sensor can move to regions in which fixed sensors have no coverage. However, even when such coverage exists, it might not be sufficient, as illustrated in Fig. 1.2. First, while a person might be observed by a surveillance camera (Fig. 1.2a, where the uncertainty of the camera's measurements is indicated in green), additional sensor readings by the robot (blue) result in a more precise estimate of the person's location (red). Second, often not all relevant visual features required for person identification might be reliably detected by the fixed sensors (Fig. 1.2b), in which case the up-close and adjustable field of view of a mobile sensor can provide the required extra information.

This is a challenging environment for planning algorithms, as it goes further than dealing with uncertainty associated with the robot. There is uncertainty associated with everything in the environment, which must be considered when making decisions. In particular, it is important that the agent considers uncertainty in sensing and the effects of its actions on its observations of the environment. Decision-theoretic methods, in particular POMDPs, provide a comprehensive approach to model the interaction between an active sensor and the environment.

1.3 Complexity

The expressiveness of POMDPs to model decision-making under acting and sensing uncertainties comes at a cost. Solving a MDP is *P-complete* [23], meaning that it can be solved in polynomial time. However, adding the sensing uncertainty in the decision process increases its complexity. It has been proved that finding optimal policies for POMDPs is *PSPACE-Complete* for finite horizon [23] and undecidable for infinite-horizon [24]. Thus, unless $P = PSPACE$, POMDPS are inherently harder to solve.

In general, a POMDP suffers from two reasons for its limited scalability, usually called curse of dimensionality and curse of history [25]. The first refers to the dimension of the problems meaning that it is more complex to compute policies for problems with a higher number of states. In turn, the second is associated with the fact that algorithms consider all possible future scenarios and compute policies for all cases.

As a consequence research has focused on proposing approximate methods which may alleviate the complexity of computing policies while maintaining good policy quality. Computing policies for reduced models which approximate the original [26, 27], reducing the space of possible policies [28] or computing sub-optimal value functions within some error from the optimal solution are common approaches [4, 28, 29].

1.4 Contributions

This thesis presents contributions in two directions, related to planning using decision-theoretic algorithms. First, it tackles decision-making under uncertainty for information gain. Here, the main contributions are:

- A POMDP based decision-theoretic framework for information-gain, POMDP-IR (POMDPs with Information Rewards), that extends POMDPs to reward low-uncertainty beliefs. This extension comes at the cost of extending the action space, with information rewarding actions which have no effect on the state of the environment, but rewards the agent for reaching a particular level of knowledge. We show how these information rewards have to be set based on a desired level of certainty.
- An experimental evaluation in a toy problem showing the behavior of POMDP-IR in multi-objective problems with information-gain, and an experimental comparison with state of the art methods for information-gathering with POMDPs in a benchmark problem.
- Analysis of agent's behavior in a robot assisted surveillance case study in which a robot assists a network of surveillance cameras to identify a person. Though performed in simulation, the setup of the case study is based in a real scenario, with observation models obtained from real data.

These contributions were published in Spaan et al. [30].

Second, the main contributions about linear value function approximation in POMDPs are:

- Formalization of a point-based POMDP solver extension with linear value function approximation. With minor changes we can adapt the backup operator to use linear value functions. Moreover, by exploiting specific properties of POMDPs and linear value functions we present techniques which lead to more efficient operations. In particular, a procedure that, under certain conditions, exploits independence between observation factors to speed up the backup of the value function, and a projection method that reduces the approximation error directly over belief points.

- An automatic procedure to automatically construct suitable sets of basis functions for POMDPs with linear value functions. This method captures the dynamics of each vector update, exploits factored transitions and basis functions. We test it experimentally against the only method that exploits POMDP's dynamics to construct sets of bases, and show that we compare favorably.
- An empirical evaluation of linear value function approximation in POMDPs. Besides proving the effects of previous contributions, we provide a study on the effect of different basis functions for the same problem. Moreover, we show scalability gains for larger problems.

These contributions were partially published in Veiga et al. [31, 32].

1.5 Thesis Outline

- Chapter 2 introduces the existing framework of decision-theoretic planning under uncertainty. We will go from MDP and their solutions to the partially observable case, with exact and approximate solutions.
- Chapter 3 presents an approach to deal with information gain in active cooperative problems. We review the information-gathering problem and formalize it in a decision-theoretic framework. The POMDP-IR framework is presented, which extends classic POMDP solving methods to information-gathering. We illustrate our framework on a toy problem, and compare against other POMDP framework for information gain. Furthermore, a case study of information-gathering in a robot-assisted surveillance is presented.
- Chapter 4 introduces an approach to linear value function approximation in POMDPs. We will extend the concept of linear value functions from MDPs to POMDPs, and exploit model structure to efficiently reduce the size of value functions. We adapted a point-based POMDP solver and experimentally tested the performance of our methods.
- Finally, in Chapter 5 we present general conclusions and outline directions for future research.

Chapter 2

Decision-theoretic planning

As introduced in Chapter 1, the focus of this thesis is on decision-making for autonomous agents under the presence of uncertainty. We model the decision making problem under the Markov decision process model [33, 34]. This chapter provides an overview of decision-theoretic planning under uncertainty in stochastic environments, starting with the fully observable case, in which the agent can with full certainty identify the state of the environment. Finally, we will review the general case in which the agent can only obtain noisy information about the state of the environment, formalized as partially observable MDPs, following Spaan [1].

2.1 Markov Decision Processes

An MDP models the interaction between an agent and a stochastic environment when the agent needs to take decisions. It can be viewed as an extension of Markov chains with a set of decisions and state-based rewards (or costs). The agent influences the world through the actions it executes, attempting to change the world state to achieve a given objective. The MDP provides a framework to compute plans, by computing utilities. Formally, a MDP is specified as a tuple $(S,\ A,\ T,\ R,\ \gamma)$ where:

- S is a finite set of $|S|$ environment states that can be reliable identified by the agent;
- A is a finite set of $|A|$ actions available to the agent;
- $T : S \times A \times S \rightarrow [0, 1]$ is a state transition model of the environment. It gives for each state $s \in S$ and action $a \in A$ a probability distribution over world states,

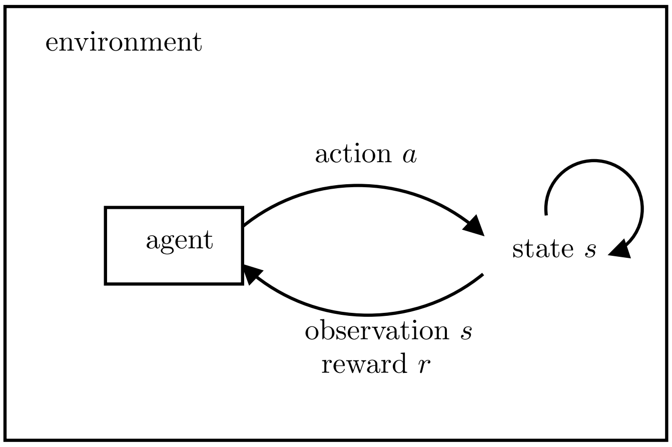


FIGURE 2.1: Diagram of a MDP agent interacting with the environment

representing the probability that the resulting state is s' when the the world is at a state s and a given action a is taken: $T(s', a, s) = p(s'|s, a)$;

- $R : S \times A \rightarrow \Re$ is a reward function that specifies the expected immediate reward $R(s, a)$ received by the agent for taking an action a in a state s.
- γ is the discount factor, which is used to weight rewards received over time. Therefore, rewards received in a distant future are less valued.

In a MDP, at every timestep t the environment is in a state $s \in S$, the agent takes an action $a \in A$ and receives a reward $R(s, a)$. When performing an action, the state changes according to the transition model $p(s'|s, a)$. MDPs possess the Markov property, i.e., at each transition the probability of reaching any state s' depends only in the current state s and the current action a.:

$$p(s_{t+1}|s_t, s_{t-1}, \dots, s_0, a_t, a_{t-1}, \dots, a_0) = p(s_{t+1}|s_t, a_t) \tag{2.1}$$

This means that the current state must provide enough information in order to predict the next state. Thus, in an MDP we can forget about all previous states since the future state is conditionally independent of all previous states and actions.

In order to automate decision making, there should be any measure on how good or bad is the behavior of the agent. This is encoded in the reward function, which gives a numerical value to the combination of being at some state and take some action. Typically, reaching good states or performing good actions will be rewarded with positive values, while bad actions and/or states receive negative values. The reward function is also useful to define priorities, when an agent can perform several tasks. For instance, in a surveillance task it is more important to detect fires than abandoned objects, therefore it receives a higher reward for correctly detecting a fire.

A measure of the long-term reward is usually called a criterion of optimality, and reflects the way how solutions are evaluated. There are two main changes between criteria, whether there is a finite horizon or an infinite horizon for decision making. And also, if rewards are discounted or not.

A finite horizon means that the agent only plans for a limited number of steps, while in the infinite horizon the agent plans for an unlimited or undecided number of steps. This influences how the agent behaves, for instance, if an agent needs to take higher risks in order to reach his goals. In an infinite horizon case, it might decide to take safer decisions to reach its goals.

Under a finite horizon model, a possible criterion is to measure the expected cumulative reward, in which performance is measured by the expected cumulative reward after acting for a finite number of steps, usually known as the planning horizon:

$$E\left[\sum_{t=0}^{h-1} R_t\right] \tag{2.2}$$

where $E[.]$ denotes the expectation operator, h is the planning horizon and R_t is the reward received at timestep t.

Under an infinite-horizon model, $h = \infty$, the cumulative reward might become infinite, making different policies indistinguishable under a numerical point of view. Therefore a discount is introduced, which models the concept that reward received earlier in the agent's lifetime are more valuable than later rewards. In this case, the optimality is measured by the expected discounted cumulative reward:

$$E\left[\sum_{t=0}^{\infty} \gamma^t R_t\right] \tag{2.3}$$

where γ is the discount rate, $0 \leq \gamma \leq 1$.

Thus, the discount factor has two roles: ensuring that a reward received in the near future is more valuable than a reward received in the far future, and ensuring that the performance measure has a finite sum. In the following we will assume an optimality criterion based on the expected discounted cumulative reward, and though we use a notation with a horizon length, those notions are general, even if $h = \infty$.

Given this, unlike classic planners, in MDPs we do not need to compute a long-term plan for each starting state, but rather computes an optimal policy, i.e., a mapping from states to actions, which tell the best action to take at every single state. A policy $\pi : S \to A$ is a mapping between states and actions, and $\pi(s)$ indicates the action $a \in A$ the agent must perform when the environment is in state $s \in S$, when following policy π. The quality of a policy is measured by the long-term expected reward the agent receives by following it, or in another words, by the expected utility of the possible environment histories generated by that policy. An optimal policy $\pi^* : S \to A$ is the one which maximizes the considered performance measure.

To the performance measure we call it the value function associated with each policy, $V^\pi : S \to \Re$, which maps a state $s \in S$ to a real value. The value function indicates how valuable it is to follow a policy π when starting a plan at state s.

Using the optimality criterion (2.3) the value function of state s, $V^\pi(s)$ is:

$$V^\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R_t(s, \pi(s))\right]. \tag{2.4}$$

Given a policy, the immediate reward for each state is known, therefore we can take it out of the expectation operator and replace the formulation as:

$$V^\pi(s) = R(s, \pi(s)) + E\left[\sum_{t=1}^{\infty} \gamma^t R(s_t, \pi(s_t))\right]. \tag{2.5}$$

The expectation operator averages over the stochastic transition model, which leads to the following recursion, known as the Bellman recursion [35]:

$$V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} p(s'|s, \pi(s)) V^\pi(s') \tag{2.6}$$

The policy π corresponding to any value function V can be easily extracted by the equation:

$$\pi(s) = \operatorname*{argmax}_{a \in A} \left[R(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) V(s')\right] \tag{2.7}$$

which instructs the agent to take the action which maximizes not only the immediate reward, but which as part of a long run policy maximizes the sum of immediate reward and future discounted rewards.

2.1.1 Value Iteration

As mentioned, the goal in an MDP is to find the optimal policy, i.e., the one which maximizes the value function. Therefore, we may rewrite the Bellman recursion to find the optimal value function $V^*(s)$:

$$V^*(s) = \max_{a \in A} \left[R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V^*(s') \right] \tag{2.8}$$

This is called the Bellman equation, and forms the basis for solving MDPs. If there are n possible states, then there are n Bellman equations, one for each state. These n equations contain n unknowns. Thus, if those were linear equations it would be straightforward to solve them using linear algebra. However, the equations include the nonlinear *max* operator, which induces nonlinearity in the equations. Besides, for large numbers of states it might not be feasible to a system with such a large number of equations. Therefore, a successive approximation technique has been introduced, called value iteration, which turns (2.8) into an update, computing the optimal value function of each state by iterating over successive steps into the future. First the value function is initialized at the optimal value function V_0^*, defined as the maximum reward the agent gets when it can take only one action:

$$V_0^*(s) = \max_{a \in A} R(s,a) \tag{2.9}$$

The value function is then iterated over time, following the rule:

$$V_{n+1}(s) = \max_{a \in A} \left[R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V_n(s') \right] \tag{2.10}$$

This operation is known as a Bellman backup [35], denoted as $\mathcal{T}_{MDP}$, and converges to the fixed point V^*. We may rewrite (2.10) as:

$$V_{n+1} = \mathcal{T}_{MDP} V_n \tag{2.11}$$

Value iteration is proved to converge for the optimal value function when $n \to \infty$, in which case it is possible to extract the optimal policy. In practice, it is iterated until the value function has converged, i.e., until the largest state value update during one step iteration is below a certain threshold. If the difference between two successive iteration is bounded by:

$$\|V_n - V_{n-1}\| \leq \frac{\epsilon(1-\gamma)}{\gamma} \tag{2.12}$$

then the difference to the optimal value function [34] is bounded by:

$$\|V^* - V_n\| \leq \epsilon \tag{2.13}$$

The algorithm for value iteration solving of MDPs is shown in Algorithm 1.

Algorithm 1 MDP Value iteration

```
Input: $\epsilon$
Output: An approximate to optimal value function $V^*$ with Bellman error less than $\epsilon$
Set $V(s) = 0$ for all $s \in S$
repeat
  $V' \leftarrow V$
  for all $s \in S$ do
    $V(s) \leftarrow \max_{a \in A} \left[R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V'(s')\right]$
until $\max_{s \in S} |V'(s) - V(s)| \leq \frac{\epsilon(1-\gamma)}{\gamma}$
```

2.1.2 Policy Iteration

Policy iteration is an alternative method which searches for solutions in the policy space. This method alternates iteratively between calculation of the expected value for a given policy (2.6), and the improvement of that policy using a greedy one-step look-ahead value calculation (2.7). Policy iteration is guaranteed to converge for the optimal policy.

Algorithm 2 MDP Policy Iteration

```
Input: $\epsilon$, $\pi_0$
$i \leftarrow 0$
repeat
  $i \leftarrow i + 1$
  for all $s \in S$ do
    Solve $V(s) = R(s, \pi_{i-1}(s)) + \gamma \sum_{s' \in S} p(s'|s, \pi_{i-1}(s)) V(s')$
  for all $s \in S$ do
    $\pi_i(s) \leftarrow \text{argmax}_a \left[R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V(s')\right]$
until $\pi_i = \pi_{i-1}$
```

2.1.3 Linear Value Function Approximation

In the MDP framework, a popular approach to compactly represent value functions, and overcome the *curse of dimensionality*, is the use of linear value function approximation.

Linear value functions are defined as value functions represented by a linear combination of basis functions, inspired by work on function approximation [36].

When approximating value functions by linear regression the space of allowable value functions $\mathcal{V} \in \mathcal{H} \subseteq \Re^{|S|}$ is defined via a set of basis functions $H = \{h_1, \ldots, h_k\}$. A linear value function is a function defined over a set of basis functions that can be written as $V(s) = \sum_{j=1}^{k} \omega_j h_j(s)$ for some coefficients $\boldsymbol{\omega} = (\omega_1, \ldots, \omega_k)$. $\mathcal{H}$ is the linear subspace of $\Re^{|S|}$ spanned by the basis functions H. If basis functions are represented as an $|S| \times k$ matrix $\boldsymbol{H}$, then an approximate value function is represented by $\boldsymbol{H\omega}$.

The assumption when approximating value functions using basis functions is that we can represent the value function within $\mathcal{H}$. In the value iteration algorithm, whenever we take a step that takes the value function outside this space, we must project it back into the space by finding the value function in the space which is closest to the result.

In practice, the value iteration with approximate linear value functions alternates between the application of the Bellman operator and projecting the results back to space $\mathcal{H}$, as follows:

$$\bar{V}_{n+1} = \mathcal{T}_{MDP} \boldsymbol{H\omega}_n \tag{2.14}$$

$$\boldsymbol{H\omega}_{n+1} = \Pi \bar{V}_{n+1} \tag{2.15}$$

where $\mathcal{T}_{MDP}$ is the Bellman backup operator for MDPs, Π is the projection operator which brings the value function representation back to $\mathcal{H}$ and $\bar{V}_i$ is an intermediate value function at time step i. One definition of the projection operator is given in [37]:

Definition 2.1. A projection operator Π is a mapping $\Pi : \Re^{|S|} \to \mathcal{H}$. Π is said to be a projection operator w.r.t. a norm $\|\cdot\|$ if: $\Pi\bar{V} = \boldsymbol{H\omega}^*$ such that $\boldsymbol{\omega}^* \in \operatorname{argmin}_{\boldsymbol{\omega}} \left\|\boldsymbol{H\omega} - \bar{V}\right\|$.

The choice of the norm used in the projection operator is crucial for its properties. In particular, the Bellman backup operator is a contraction in the $\mathcal{L}_\infty$ norm and the composite operator $\Pi\mathcal{T}_{MDP}$ must be a contraction in some norm. Several norms have been studied and convergence proofs presented for certain cases. The $\mathcal{L}_2$ seems a natural choice but was proven to diverge for some cases [38], and convergence was established with restrictions on basis functions. Besides, projection in this case is not guaranteed to be a contraction in the $\mathcal{L}_\infty$ norm [39]. Weighted $\mathcal{L}_2$ was proven to be a valid alternative which meets the given criteria [40], given that the Bellman operator is a contraction in that norm [41]. Convergence was also proved in an approach using a projection operator in $\mathcal{L}_\infty$ norm [42]. This can be solved by linear programming, but takes a number of constraints linearly dependent on the number of states, which might be impracticable for large state spaces. It is argued that the use of factored MDPs can contribute to represent those constraints more efficiently, and contribute to reduce the computational cost of the algorithm.

When using a $\mathcal{L}_2$ norm this is a problem of least squares. By definition, parameter estimation with least squares for a model $y \approx X\beta$ is given by [39]:

$$\begin{aligned} \hat{\beta} &= \operatorname*{argmin}_{\beta} \|X\beta - y\|_2 && (2.16) \\ &= (X^T X)^{-1} X^T y && (2.17) \end{aligned}$$

It directly follows that the solution for the projection operator is given by:

$$\begin{aligned} \hat{\boldsymbol{\omega}}_{n+1} &= \operatorname*{argmin}_{\boldsymbol{\omega}} \left\|H\boldsymbol{\omega} - \bar{V}_{n+1}\right\|_2 && (2.18) \\ &= (H^T H)^{-1} H^T \bar{V}_{n+1} && (2.19) \end{aligned}$$

We may also formulate the problem of parameter estimation with a $\mathcal{L}_\infty$ norm:

$$\hat{\boldsymbol{\omega}}_{n+1} = \operatorname*{argmin}_{\boldsymbol{\omega}} \left\|H\boldsymbol{\omega} - \bar{V}_{n+1}\right\|_\infty \qquad (2.20)$$

Although this formulation defines a non-linear mapping, it may be solved by linear programming [43]:

$$\begin{aligned} &\text{variables} && \omega_1, \ldots, \omega_n, \phi \\ &\text{minimize} && \phi \\ &\text{subject to} && \phi \geq \sum_{i=1}^{n_h} \omega_i h_i(s) - \alpha(s), \\ & && \phi \geq \alpha(s) - \sum_{i=1}^{n_h} \omega_i h_i(s),\ s = 1, \ldots, |S| \end{aligned} \qquad (2.21)$$

The constraints in this linear program ensure that $\phi \geq |\sum_{i=1}^{n_h} \omega_i h_i(s) - \alpha(s)|$ for each state, or equivalently, that $\phi \geq \| \sum_{i=1}^{n_h} \omega_i \boldsymbol{h}_i - \alpha\|$. This LP has $n_h + 1$ variables but $|S|$ constraints, which can make it impractical for large state spaces.

2.2 Partially Observable Markov Decision Processes

In the previous section we described MDPs, which assume that the agent has complete knowledge of the environment state, or in other words, that the environment is fully observable to the agent. A more realistic assumption would be to say that the agent's observation of the environment is incomplete, due to noise and uncertainty in sensing. In this case, we say that the environment is partially observable, and the agent does not know in which state it is in. Thus, Partially Observable Markov Decision Processes (POMDPs) [12] have been introduced as an extension to MDP settings to deal with uncertainty in state observation. Figure 2.2 depicts a POMDP agent interacting with the environment. Note the difference to Figure 2.1: instead of receiving the actual state from the environment, the agent receives an observation, which is captured by a probabilistic model.

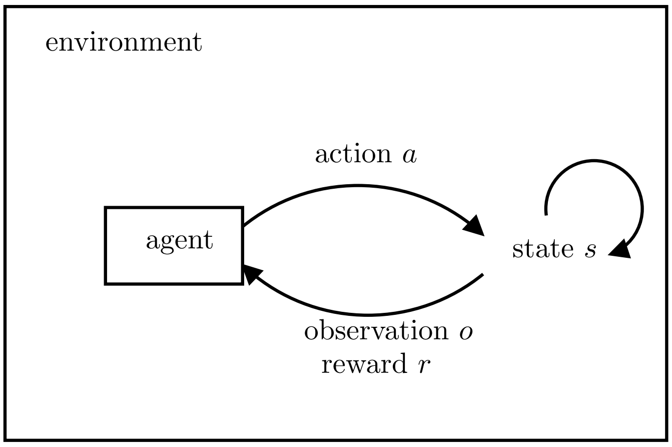


FIGURE 2.2: Diagram of a POMDP agent interacting with the environment

Formally, a POMDP is a tuple $(S,\ A,\ T,\ R,\ Z,\ O,\ \gamma)$, where:

- S is the state space, A is the action space, T is the transition function, R the reward function and γ the discount factor as defined in the MDP case (Section 2.1);
- Z is a finite set of observations available to the agent;
- $O : S \times A \times Z \to [0, 1]$ is an observation function. It gives for each state $s' \in S$ and action $a \in A$ a probability distribution over observations, representing the probability that the agent receives an observation o when the world is in state s and an action a is performed: $O(z, s', a) = p(o|s', a)$.

Under this framework, the system is no longer Markovian, since observations do not provide unique state identification and some sort of memory needs to be provided to the system such that the agent is able to infer from it what might be the world current state. One possible implementation would be to maintain an history of observations, but that such implementation would grow indefinitely over time, turning it impractical for large problems.

Researchers have shown that it turns out that it is sufficient for an agent to maintain a probability distribution over states, called belief, such that it keeps the same information as saving the complete observation history. The belief $b(s)$ is a probability distribution over the state space S, denoting the probability that the environment is in state $s \in S$. As it is a probability distribution, it must sum to 1, thus a belief can be represented with a finite vector with size $|S| - 1$, where $|S|$ is the size of the state space. Every time the agent takes an action and receives an observation the belief is update by the Bayes' rule:

$$b_a^o(s') = p(s'|o,a,b) = \frac{p(o|s',a)}{p(o|b,a)} \sum_{s \in S} p(s'|s,a)b(s) \tag{2.22}$$

where $p(o|a,b) = \sum_{s' \in S} p(o|s',a) \sum_{s \in S} p(s'|s,a)b(s)$ is a normalizing constant.

The belief b is itself a Markovian signal, as its dynamics depends on the previous belief, the action taken and the current observation. Given that the belief is fully observable to the agent, solving a POMDP can be transformed into solving an MDP defined on the space of belief states, i.e., a belief-state MDP [13]. Although solving a POMDP can be reduced to solving a MDP, this is a continuous MDP, which increases complexity of computing solutions.

Formally, a belief-state MDP is a completely observable continuous-space MDP, defined as a tuple $(B,\ A,\ r,\ \tau,\ \gamma)$ where B is the set of belief states, A is the set of actions, r is the reward function constructed from the MDP reward taking expectations according to the belief state and γ is the discount factor:

$$r(b,a) = \sum_{s \in S} b(s)R(s,a) \tag{2.23}$$

and τ is the belief state transition function:

$$\tau(b,a,b') = p(b'|b,a) = \sum_{o \in O} p(b'|b,a,o)p(o|b,a) \tag{2.24}$$

where $p(b'|b,a,o) = 1$ if $SE(b,a,o) = b'$ or $p(b'|b,a,o) = 0$ if $SE(b,a,o) \neq b'$. SE is the state estimator, which updates the belief state as defined in (2.22)

2.2.1 Value Function

Following a belief-state MDP formulation, we may represent solutions like in the fully observable MDP. The goal is also to compute an optimal plan such that the agent successfully completes its tasks. However, in the MDP framework policies maps states to actions, while in POMDPs policies maps beliefs to actions: $\pi : B \to A$. The quality of a policy $\pi(b)$ is measured by by a value function $V^{\pi}(b)$ which is defined as the expected future discounted reward the agent can gather by following π starting from a belief b with the planning horizon h:

$$V^{\pi}(b) = E_{\pi}\left[\sum_{t=0}^{h-1} \gamma^t r(b_t, \pi(b_t)) \middle| b_0 = b\right]. \tag{2.25}$$

An optimal policy π^* is the one which maximizes the value function V^{π}. The expected operator averages over the belief transition model, and like in the MDP framework we can rewrite the equation, leading to the recursive Bellman equation. The optimal value function satisfies the Bellman optimality equation, $V^* = H_{POMDP}V^*$:

$$\begin{aligned} V^*(b) &= \max_{a \in A}\left[r(b,a) + \gamma \sum_{b'} \tau(b,a,b')V^*(b')\right] \qquad (2.26)\\ &= \max_{a \in A}\left[r(b,a) + \gamma \sum_{o \in O} p(o|b,a)V^*(b_o^a)\right] \end{aligned}$$

The solution is optimal when (2.26) holds for every belief b in the space defined by the state space S. The value function defined over the belief space may contain infinitely many values, as the belief space is continuous. Fortunately, it has been proved that in such framework the value function presents a particular structure. It was shown that value functions for finite-horizon POMDPs are piecewise linear and convex (PWLC) [13], and can be approximated arbitrarily closely by PWLC functions for infinite-horizon POMDPs [44]. Thus, value functions can be represented by a finite set of vectors $\{\alpha_n^i\},\ i = 1, \ldots, |V_n|$, and each vector has an action $a(\alpha_n^i) \in A$ associated. The action to execute under a particular policy is the one associated with the maximizing vector for a particular belief. Given a set of vector $\{\alpha_n^i\}_{i=1}^{|V_n|}$ at stage n, the value for a belief b is given by:

$$V_n(b) = \max_{\{\alpha_n^i\}_i} b \cdot \alpha_n^i \tag{2.27}$$

Given that value functions are represented by a finite set of vectors, we easily note that vectors divide the belief space into regions. Each region is associated with a particular maximizing vector, and thus with a particular action. We say that the maximizing vector dominates the set of vectors for that particular region. In Figure 2.3 we see a practical example of a value function, represented as a set of vectors. This value function for a two-state problem has 3 different regions, each one dominated by a vector. The upper surface of the vectors is the PWLC function corresponding to the value function.

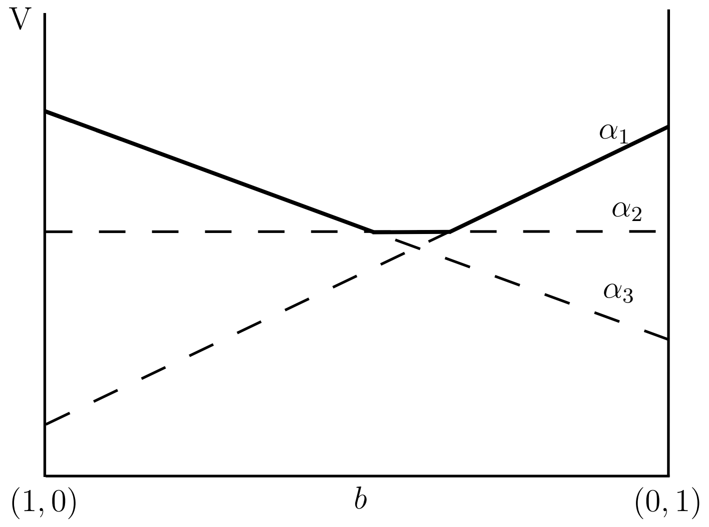


FIGURE 2.3: Example of a Value Function with two states.

2.2.2 Value Iteration

We have seen that solving a POMDP is equivalent to solve a belief-state MDP, where we account for transitions between beliefs, instead of transitions between states. We are able to keep under the MDP framework, since transitions between beliefs form a Markovian signal. Given this, it is possible to apply value iteration methods to POMDPs, through the belief MDP implementation. However, those are defined over continuous spaces, since the belief space is continuous, and value iteration needs to be adapted to this condition.

Recalling, in value iteration the goal is to compute the policy which maximizes the value function. In a POMDP the value function is parametrized by a finite number of hyperplanes (α-vectors) over the belief space. This set of vectors partition the belief space into regions, each one associated with a given action, corresponding to the vector which maximizes the value function in this region. In a fully observable MDP value iteration computes the optimal value function by considering successive steps, looking one step deeper into the future at each iteration, considering all possible actions that the agent can take at each action. In a POMDP value iteration also looks successive into the future, considering all possible actions and observations for an agent.

The idea behind value iteration in POMDPs is that we can backup α-vectors iteratively such that at iteration n the value function is parametrized by a finite set of vectors $\{\alpha_n^i\},\ i = 1, \ldots, |V_n|$. We can compute the value of a belief b by (2.27) and the policy is obtained by finding the action associated with the maximizing vector:

$$\pi(b) = a\left(\underset{\{\alpha_{n+1}^k\}_k}{\operatorname{argmax}}\, b \cdot \alpha_{n+1}^k\right) \tag{2.28}$$

2.2.3 Exact value iteration

One way to apply value iteration is to generate all possible next stage vectors, while removing the set of dominated vectors. These methods are called exact value iteration methods, as they search in the whole belief space for all possible vectors. Monahans's enumeration algorithm [45] builds an intermediate set of vectors, and combines all possible actions and observations the agent might receive.

We start by defining g_{ao} vectors, which represent the vectors resulting from back-projecting α_n^k for a particular action a and observation o:

$$g_{ao}^k(s) = \sum_{s'} p(o|s', a) p(s'|s, a) \alpha_n^k(s') \tag{2.29}$$

And we can define the backup operator as:

$$H_{POMDP} V_n = \bigcup_a G_a, \text{ with} \tag{2.30}$$

$$G_a = \bigoplus_o G_a^o, \text{ and} \tag{2.31}$$

$$G_a^o = \left\{\frac{1}{|O|}\alpha_0^a + \gamma g_{ao}^k\right\}_k \tag{2.32}$$

where $\bigoplus$ and $\bigcup$ represent the cross-sum and union operators.

The overall complexity of one iteration is $O(|A| \times |S| \times |V|^{|O|})$, and $|A||V|^{|O|}$ new vectors are generated, which easily explode the memory demand for large problems.

One step to overcome this is by noting that many of the newly generated vectors will not be part of the value function, as its maximizing region will be empty. Since the value function at intermediate stages is a lower bound for the optimal value function, it is safe to remove dominated vectors, as those will not influence the dominant set of vectors at future iterations. Therefore, an additional step is included in value iteration methods to prune such vectors:

$$V_{n+1} = prune\left(H_{POMDP}V_n\right) \tag{2.33}$$

The *prune* operator may be implemented by solving a linear program [46].

To improve the way to prune vectors a more clever algorithm was presented, called Incremental Pruning [47]. It is noted that the pruning step can be more efficiently written as:

$$\text{prune}\left(A \oplus B \oplus C\right) = \text{prune}\left(\text{prune}\left(A \oplus B\right) \oplus C\right) \tag{2.34}$$

so we may rewrite part of (2.32) as:

$$\begin{aligned} G_a &= \text{prune}\left(\bigoplus_a G_a\right), \text{ with} && (2.35)\\ &= \text{prune}\left(G_a^1 \oplus G_a^2 \oplus G_a^3 \oplus \ldots \oplus G_a^{|O|}\right) && (2.36)\\ &= \text{prune}\left(\ldots \text{prune}\left(\text{prune}\left(G_a^1 \oplus G_a^2\right) \oplus G_a^3\right) \ldots \oplus G_a^{|O|}\right) && (2.37) \end{aligned}$$

Despite the improvements generated by the introduction of pruning steps, in practice, exact value iteration is only feasible for small problems. Moreover, the complexity of the linear programs used in for pruning depend on the size of the set of vectors V_n and the vectors' dimensionality, thus becoming too complex for larger problems.

2.2.4 Approaches to Approximation

The intractability of exact solutions led to several approximations to POMDP solving. We will outline the most common approaches to approximation. We focus on approaches in the partially observable setting, although similar approaches may be found in fully observable MDPs. Note that, although we split this section into different approximation methods, they are not exclusive. For instance, model or policy approximation indirectly also induce an approximated value functions in the sense that it might differ from the value function computed with exact methods. Nonetheless, approximation methods are grouped according to their main focus.

2.2.4.1 Model approximation

In model approximation methods, the approximation does not takes place in the POMDP solving, but rather it attempts to reduce the complexity of the model itself. For instance, clustering the state, observation and/or actions spaces might be a form of model approximation. One approach is to transform a partially observable model into a region observable, in which the agent knows in which region the true state of the system is [26]. We are still able to solve a reduced POMDP with existing solving methods. Approximations which use the underlying MDP as an approximation to the original POMDP model have been proposed [48, 49]. State partitioning was explored in the context of fully observable MDPs, with techniques that may be extended to POMDPs as well [50].

2.2.4.2 Policy approximation

Policy approximation schemes approximate optimal solutions by searching for policies in a reduced policy space. Naturally, such methods should be more useful in problems with large action spaces. This helps to alleviate the problem of finding solutions to POMDPs by simplifying the policy optimization problem, as a consequence of restricting the allowed space of policies to the agent [51]. Examples of policy approximation methods are memoryless policies [52, 53], policies based on truncated histories [54, 55] and finite state controllers with bounded size [56]. In the framework of reinforcement learning, gradient-based approaches [57], which follows the gradient of parametrized policy space, have been proposed [58–60].

2.2.4.3 Value function approximation

Lastly, a popular family of approximation methods is the value function approximation. Generally, this can be defined as the set of methods which compute sub-optimal value functions, which yield sub-optimal policies. Typically, approximated value functions are less complex and easier to compute. For instance, a family of approximations already mentioned as model approximators, which use the underlying MDP, may also be seen as value function approximation, given that, in the end, the resulting value function is also sub-optimal [51].

One of the major sources of intractability is the domain size of a value function in a POMDP, the continuous belief space. Given the PWLC property of the value function exact solutions can still be found by enumerating all possible vectors and pruning useless vectors, at the cost of high computational complexity. Grid-methods [61] approximate the value function by discretization of the belief space in a grid pattern, then interpolate over those to compute values of beliefs not in the grid, but the number of points can grow exponentially and many points might be unreachable. The popular family of point-based methods [62] also approximates by computing value functions over a subset of belief points. They compute solutions for the reachable parts of the belief space, which are sampled beforehand by letting the agent interact with the environment. By working on a reduced set of belief points, these methods end up reducing the number of vectors used to represent the value function, which makes computation simpler.

2.2.5 Point-based value iteration

By dropping the requirement for optimal solutions and look instead for approximate optimal solutions, new methods for POMDP solving have been developed. Those methods compensate the loss of optimality with their ability to scale better than exact methods. Most common methods are heuristic approaches [28, 63, 64], policy search [65], grid-based methods [66] and point-based methods.

While exact algorithms search for solutions in the whole belief space, point-based methods plan over a finite subset of beliefs $B \subset \Delta(S)$, and expect that those results extend for the whole belief space. Methods differ in the way that belief points are sampled, which vary from sampling points to let the agent interact with the environment.

Point-based methods take from equation (2.32) and derive an equation to compute the value function at each particular belief b:

$$\begin{aligned} V_{n+1}(b) &= \max_a \left[b \cdot \alpha_0^a + \gamma b \cdot \sum_o \operatorname*{argmax}_{\{g_{ao}^k\}_k} b \cdot g_{ao}^k \right] && (2.38) \\ &= \max_{g_a^b} b \cdot g_a^b, && (2.39) \\ \text{with } g_a^b &= \alpha_0^a + \gamma \sum_o \operatorname*{argmax}_{\{g_{ao}^k\}_k} b \cdot g_{ao}^k && (2.40) \end{aligned}$$

Finally, the backup operator which selects the maximizing vector for the belief b becomes:

$$\text{backup}(b) = \operatorname*{argmax}_{\{g_a^b\}_{a \in A}} b \cdot g_a^b \tag{2.41}$$

The value function at the next step is the union of all the vectors resulting from backup all the belief points in the set B.

$$H_{PBVI} V_n = \bigcup_{b \in B} \text{backup}(b) \tag{2.42}$$

All point-based methods share the backup(b) operation and differ in two main aspects. Algorithm 3 outlines a generic point-based algorithm, where the $COLLECT$ method selects the new belief points, possibly depending on the current value function V_n and a previous set of beliefs B. The $UPDATE$ method decides at which belief points are backed up, generating new value functions.

Algorithm 3 General Point-Based POMDP Solver

```
$t \leftarrow 0$
Initialize $V_0$
Initialize $b$
**while** stopping condition not met **do**
  $B_{new} \leftarrow COLLECT(V_t, B, N)$
  **for** $U$ iterations **do**
    $V_{t+1} \leftarrow UPDATE(V_t, B, B_{new})$
    $t \leftarrow t + 1$
  $B \leftarrow B \cup B_{new}$
```

The most known point-based methods for solving POMDPs are Point-Based Value Iteration [62], Perseus [67], Heuristic Search Value Iteration [68], Point-Based Error Minimization Algorithm [69], Forward Search Value Iteration [70] and Successive Approximation of the Reachable Space under Optimal Policies [71].

As mentioned, those methods differ on the collection and update stages. The original PBVI method expands the belief set by maximizing the L_1-norm between beliefs already in the set and their successors, whereas PERSEUS performs random forward simulation starting from an initial belief distribution. HSVI and GapMin use the bound uncertainty heuristic that search for points that maximize the gap between bounds of the value function. PEMA is a variation of PBVI that searches for belief points which minimize the approximation error between a point-based backup and a complete backup. Finally, FSVI guides the belief collections by following the policy of the underlying MDP.

In this set of methods we can find three different update strategies. The most straightforward is the full backup strategy, which executes a backup on every belief in the set of

Algorithm	Collect	Update
PBVI	L_1-norm	full backup
Perseus	Random	asynchronous backup
HSVI	Bound uncertainty	newest points backup
GapMin	Bound uncertainty	full backup
PEMA	error minimization	full backup
FSVI	MDP heuristic	newest points backup

Table 2.1: Outline of point-based algorithms, with the associated collection and updating method [4]

belief points. Instead, the newest points strategy applies, at each iteration, the backup operator only to the new points in the set. The asynchronous backup, used solely by the PERSEUS method, uses a random order of backups, but ensuring that the value of each belief are improved after each iteration. Table 2.1 summarizes the most known point-based algorithms. A thorough and more detailed survey on point-based POMDP solvers may be found in the paper by Shani et al. [4].

2.2.6 PERSEUS

We highlight here the PERSEUS method, as it will be our method of reference for POMDP solving throughout this thesis. PERSEUS is a randomized algorithm, which can be read with two different meanings. In one sense, it collects belief points by randomly exploring the environment, thus collecting only reachable points. In the other hand, in the update stage it randomly chooses the order of belief points at which it updates the value function. The method saves additional computation by keeping track of the non improved beliefs during the update stage, consisting of the points $b \in \tilde{B}$ whose new value $V'(b)$ is still lower than $V(b)$. At each backup, all points whose value is improved by the new vector are removed from the set $\tilde{B}$. This ensures that the method does not need to compute backups for all belief points, while requiring that the value of all points, at least, does not decrease at each iteration. One typical convergence criteria is the difference between successive value function-estimates, i.e., $\max_{b \in B}(V'(b) - V(b))$. The procedure for POMDP solving with PERSEUS is shown in Algorithm 4.

2.3 Model Representation

So far, we have assumed a (PO)MDP models with a flat representation that enumerates all possible states, observations and actions. In this section we will review the use

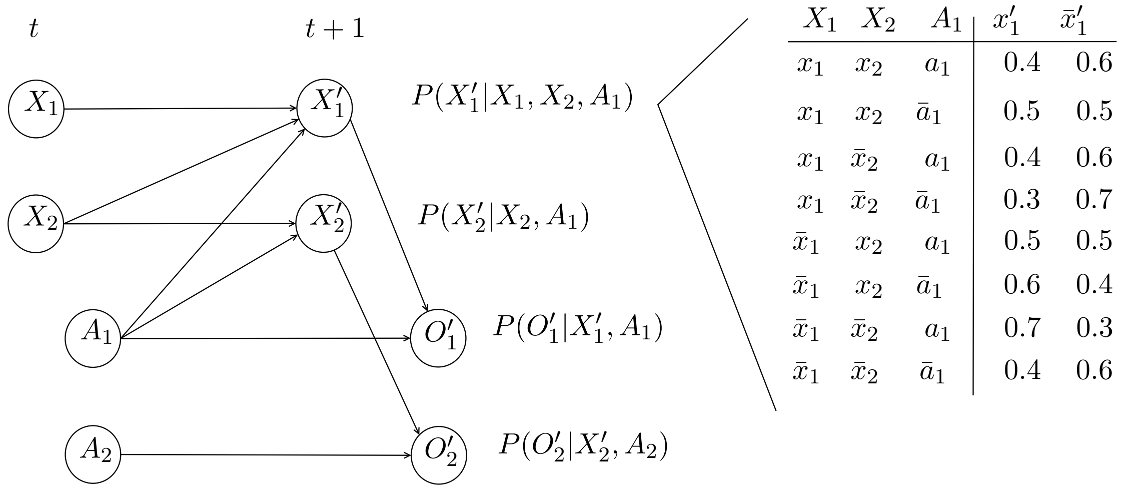
Algorithm 4 Perseus

```
Function Perseus
$B \leftarrow RandomExplore(n)$
$V \leftarrow PerseusUpdate(B, \emptyset)o$

Function RandomExplore(n)
$B \leftarrow b_0$
$b \leftarrow b_0$
repeat
  sample $a \in A$
  sample $o \in O$ following the $P(o|b,a)$ distributions
  $b \leftarrow b^{a,o}$
  $B \leftarrow B \cup b$
until $|B| = n$

Function PerseusUpdate(B,V)
repeat
  $\tilde{B} \leftarrow B$
  $V' \leftarrow \emptyset$
  while $\tilde{B} \neq \emptyset$ do
    sample $b \in \tilde{B}$
    $\alpha \leftarrow backup(b, V)$
    if $\alpha \cdot b \geq V(b)$ then
      $\tilde{B} \leftarrow \{b \in \tilde{B} : \alpha \cdot b < V(b)\}$
      $\alpha_b \leftarrow \alpha$
    else
      $\tilde{B} \leftarrow \tilde{B} - b$
      $\alpha_b \leftarrow \text{argmax}_{\alpha \in V}\, \alpha \cdot b$
    $V' \leftarrow V' \cup \alpha_b$
  $V \leftarrow V'$
until V has converged
```

of factored model representation that allows for a compact but exact representation of transition, observation and reward functions. Moreover, linear value function approximation is extended to factored models, such that basis functions with reduced scope can be compactly represented.

2.3.1 Factored models

The definition of the state space in a Markovian problem is of extreme importance, since it defines how we model a description of the world. Usually it is considered a state space S consisting of a discrete and finite set of states, without assuming any structure. However, we may consider for each particular problem a number of different features which characterize the environment. If each feature is represented by a variable X_i with domain $\mathcal{D}_i$, the set of factored states X of the system may be seen as a cross-product of all the possible variables of each variables: $X = \mathcal{D}_1 \times \mathcal{D}_2 \times \ldots \times \mathcal{D}_k$ (assuming an environment with k features). In the same way, the action space A and the observation space O may also be decomposed such that each action a and observation o correspond to a joint-instantiation of variables. Such representation is called a factored representation [72, 73], which allows for the transition, observation and reward functions to be defined in terms of state variables, actions variables and observations variables (instead of simply states, actions and observations). This also prevents the need to explicitly define all states in the problem, which grow exponentially with the number of variables.

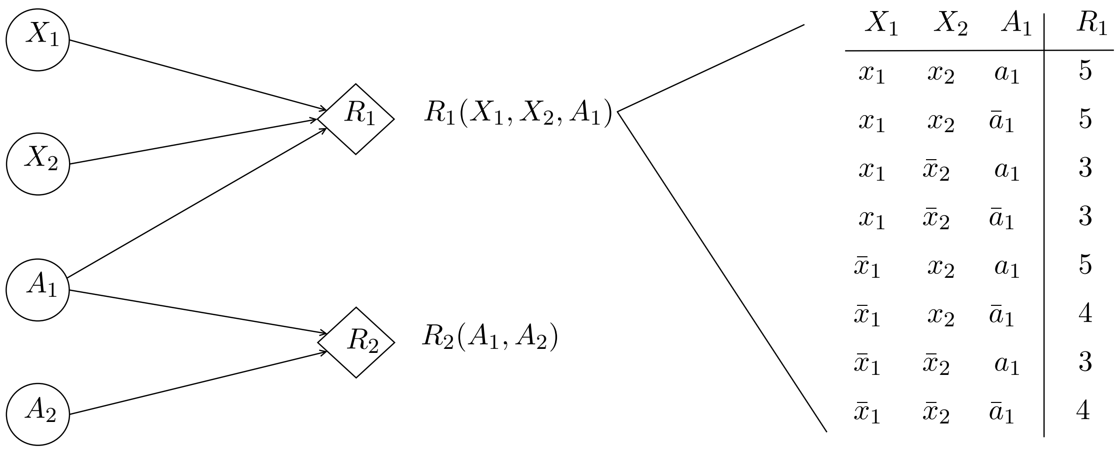


X_1	X_2	A_1	x'_1	$\bar{x}'_1$
x_1	x_2	a_1	0.4	0.6
x_1	x_2	$\bar{a}_1$	0.5	0.5
x_1	$\bar{x}_2$	a_1	0.4	0.6
x_1	$\bar{x}_2$	$\bar{a}_1$	0.3	0.7
$\bar{x}_1$	x_2	a_1	0.5	0.5
$\bar{x}_1$	x_2	$\bar{a}_1$	0.6	0.4
$\bar{x}_1$	$\bar{x}_2$	a_1	0.7	0.3
$\bar{x}_1$	$\bar{x}_2$	$\bar{a}_1$	0.4	0.6

FIGURE 2.4: Example of a Dynamic Bayesian Network which encodes the states and transitions of a MDP problem [2].

Factored transition and observation functions usually are represented as a dynamic Bayesian network (DBN) [74, 75]. DBNs take advantage and exploit conditional independence, which refers to the fact that some variables are probabilistically independent of each other when other variables values are held fixed.

Transitions and observation probabilities ($P(s'|s, a)$ and $P(o|s, a)$) are expressed using an acyclic directed graph. The graph is explicitly divided in two parts, corresponding each one to a timestep, being one immediately posterior to the other. Nodes represent state, action and observation variables and edges probabilistic dependencies. Dependencies go either from the previous to the current timestep or from the current to the same timestep. Each node in the posterior timestep has a conditional probability table (CPT) which defines the conditional probability distribution $P(X'_i|\Gamma(X'_i))$, where $\Gamma(X_i)$ is the scope of variable X_i in the DBN or, in other words, the set of parent variables of X_i. If the scope of a variable is a subset of all the state variables, then the table has a smaller dimension. Therefore, the transition function of the model may be factorized in a product of smaller conditional distributions. For instance, in the example of Figure 2.4 the scopes of state variables are $\Gamma(X_1) = \{X_1, X_2\}$ and $\Gamma(X_2) = \{X_1\}$. If we consider the state as a whole the transition function is represented as $P(X'_1, X'_2|X_1, X_2, A_1, A_2)$. However, by using



X_1	X_2	A_1	R_1
x_1	x_2	a_1	5
x_1	x_2	$\bar{a}_1$	5
x_1	$\bar{x}_2$	a_1	3
x_1	$\bar{x}_2$	$\bar{a}_1$	3
$\bar{x}_1$	x_2	a_1	5
$\bar{x}_1$	x_2	$\bar{a}_1$	4
$\bar{x}_1$	$\bar{x}_2$	a_1	3
$\bar{x}_1$	$\bar{x}_2$	$\bar{a}_1$	4

FIGURE 2.5: Example of how to encode the reward function of a MDP in a Dynamic Bayesian Network [2].

a factored representation the transition is reduced to the product of the transitions of variables X_1 and X_2: $P(X_1', X_2'|X_1, X_2, A_1, A_2) = P(X_1'|X_1, X_2, A_1) \cdot P(X_2'|X_2, A_1)$. The same procedure is used for the observation function : $P(O_1', O_2'|X_1', X_2', A_1, A_2) = P(O_1|X_1', A_1) \cdot P(O_2'|X_2', A_2)$.

The reward function may also take advantage of a factorized representation and be compactly represented exploring the concept of additive separability [76]. For example, the reward function in Figure 2.5 $R(X_1, X_2, A_1, A_2)$ may be represented as the sum of two smaller reward functions, each one depending on a smaller subset of variables: $R(X_1, X_2, A_1, A_2) = R_1(X_1, X_2, A_1) + R_2(A_1, A_2)$.

2.3.2 Algebraic Decision Diagrams

This allows for a compact model representation, but (PO)MDPs are usually a fast mixing process, which leads to complex value functions. Algebraic Decision Diagrams (ADDs) naturally capture the independencies in the Bayesian network and creates compact representations for value functions. They extend from decision trees, where nodes are labeled by state variables and edges are labeled by possible values of the state variables. To obtain the value of particular instances of states, then one should follow the respective path through the tree, until a leaf is found. Leafs contains the value for that particular state.

By avoiding explicit enumeration of the state space, ADDs have been efficiently used for MDP [77] and POMDP [2] solving. Moreover, it was shown that point-based POMDP solving can be performed more efficiently when using ADD as data representation. A number of alternative formulations for the backup operator have been proposed in the

Algorithm 5 Backup Operator with ADDs, $\mathit{backup}(b, V)$

```
$a^* \leftarrow \emptyset, v^* \leftarrow -\infty$
for each $o \in \Omega$ do $\widetilde{\alpha}^*_o \leftarrow nil$
for all $a \in A$ do
  $Q_a(b) \leftarrow 0$
  for all $o \in \Omega$ do
    $b' \leftarrow \tau(b, a, o)$
    $\alpha_o \leftarrow \text{argmax}_{\alpha \in V}\ \alpha \cdot b'$
    $Q_a(b) \leftarrow Q_a(b) + \alpha_o \cdot b'$
  if $Q_a(b) > v^*$ then
    $a^* \leftarrow a, v^* \leftarrow Q_a(b)$
    for each $o \in \Omega$ do $\alpha^*_o \leftarrow \alpha_o$
return $r_{a^*} + \gamma \sum_o g^{\alpha^*_o}_{a,o}$
```

literature. In particular, we highlight the work of Shani et al. [78], which presents point-based methods with an ADD-based implementation with a more efficient formulation for the backup operator. They note that most of the computed g_{ao} vectors are dominated by others, and their computation could be avoided if we knew a priori the maximizing action a and vector α^k. Instead, they use (2.26) to find them and only then build the output of the backup operator. It is shown that, although the complexity of each backup does not change, in practice computational times are reduced.

2.3.3 Factored linear value functions

In the context of linear value function approximation, factored models allows for efficient computations. In MDPs, a factored linear value function is defined as a linear function over the basis set H, where the scope of each h_i is restricted to some subset of variables, i.e., $V(\mathbf{x}) = \sum_{i=1}^{k} \omega_i h_i(\mathbf{c}_i)$, where $C_i \subseteq X$. The idea is that it is possible to take advantage of representing value functions as a linear combination of functions, each of which refers only to a small number of variables. Then, instead of a matrix representation it is possible to represent value functions and the value iteration process in a more compact way. The Bellman recursion using a factored representation becomes:

$$V_{n+1}(\mathbf{x}) = R(\mathbf{x}, a) + \gamma \sum_{\mathbf{x}' \in \mathbf{X}} p(\mathbf{x}'|\mathbf{x}, a) \sum_i \omega_i^n h_i(\mathbf{c}'_i) \tag{2.43}$$

$$= R(\mathbf{x}, a) + \gamma \sum_i \omega_i^n \sum_{\mathbf{x}' \in \mathbf{X}} p(\mathbf{c}'_i|\Gamma(\mathbf{c}'_i), a) h_i(\mathbf{c}'_i) \tag{2.44}$$

Here, $V_{n+1}(\mathbf{x})$ is composed of the sum of functions with restricted scope: the reward function and the backprojection of basis functions. It is assumed that the domains of rewards and basis functions are restricted to subsets of the state space. The reward function has its own scope, and the projection of each basis function has a scope restricted to the parents of its own scope.

Chapter 3

Active Perception with POMDPs

In Chapter 1 we introduced the concept of ACP, mainly in the context of NRS. In ACP, the goal is typically to increase the available information by reducing the uncertainty regarding the state of the environment, e.g., mobile sensor location, event detection). In a POMDP, if more information improves task performance, its policy will take information-gaining actions. For instance, a better self-localization estimate for a mobile sensor will improve the quality of fusing its information in the global frame with other sensors' information, or less uncertainty on the detection of an event reduces the risk of a wrong detection. However, in a traditional POMDP model, state-based rewards allow for defining many tasks, but do not explicitly reward information gain.

In this chapter we will study how to directly reward information gain. Information gain is not usually seen in POMDPs as a goal itself, but rather as a way to achieve task performance. We will present the POMDP-IR framework which can serve for ACP in NRS, based on the idea that we can reward the agent for reaching a certain level of belief regarding a state feature, at the cost of extending the action space.

3.1 Information Gain in POMDPs

Regular state-based rewards allow for defining many tasks, but do not explicitly reward information gain [22]. However, if more information improves task performance, the POMDP policy will take information-gaining actions. For instance, a better self-localization estimate can help a robot to navigate faster to a certain destination. Indeed, a key point of the POMDP framework is that it will only reduce uncertainty when it is beneficial for the task performance, but not for information gain *per se.* Araya-López et al. [79] note that "One can argue that acquiring information is always a means, not an end, and thus, a "well-defined" sequential-decision making problem with partial observability must always be modeled as a normal POMDP." They observe that in some applications, such as surveillance tasks or the exploration of a particular area, it is not clear how the information will be used by whoever is designing the system.

For instance, we may explicitly model the classification of a particular target as an objective, which induces a well-defined state-based task [80]. If we consider a more general surveillance setting, considering non-standard POMDP models could be beneficial. In particular, often no model will be available of how human operators evaluate and act upon the output of the surveillance system. Without a detailed description of all the surveillance objectives the problem cannot be cast as a well-defined POMDP. In such cases, the proposed methodology can be used to actively reduce uncertainty regarding certain important state features.

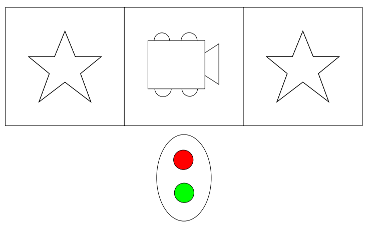
It is important to note that in many applications the user of the system might not be interested in reducing uncertainty in general, but only with respect to some features in the environment. For instance, consider the examples presented in Fig. 1.2, in which a robot provides information to either localize a target more accurately or to identify a person. In both cases, the robot's own location might be uncertain as well, but that is irrelevant to the user of the system.

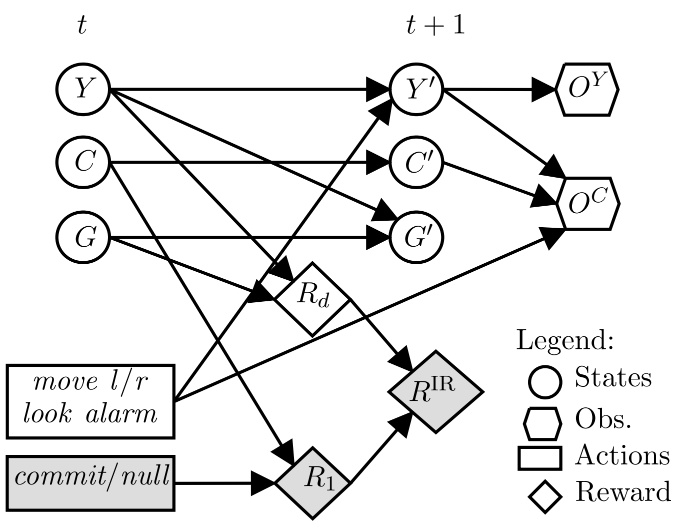
3.1.1 Running example: Patrol

To clarify the concepts presented in this section we include a small illustrative example, shown in Fig. 3.1a, dubbed Patrol. Imagine that we have a corridor environment in which a surveillance robot has to patrol between the two ends of the space (represented by stars). However, in the middle of the corridor an alarm device is present, which can be in two different configurations, *red* and *green*, where a red color means that the attention of a human operator is required.

This can be seen as a multi-objective problem, where the task of the robot is to patrol the environment while keeping track of the state of the alarm. The latter can be formalized as maintaining a low-uncertainty belief regarding the state of the alarm device. The robot can pause to observe the alarm, which delays the original patrol task. Each task is valued by its reward function and the system designer defines the balance between these objectives.

Fig. 3.1b presents a DBN representation of this problem (where the shaded nodes indicate the POMDP-IR extensions discussed in Section 3.2). This model is based on a simple robot navigation model, with a state factor $Y = \{1, 2, 3\}$ denoting the robot


(A) Environment



(B) DBN representation with information rewards

FIGURE 3.1: PATROL example. (a) Illustration of the problem, showing a corridor of length 3, a robot which has to travel between two goal nodes (marked by stars) and an alarm device in the middle. (b) Dynamic Bayesian Network representation of the problem with state factors Y = *robot position*, C = *alarm color* and G = *current goal.* The shaded nodes indicate the POMDP-IR additions to the model.

position in order from left to right, and a state factor $G = \{left, right\}$ representing the current goal. The latter represents the left or the right end of the corridor (see Fig. 3.1a), and flips when the current goal is reached. To account for the alarm, we introduce a state factor C, which models the alarm color and hence has two possible values, *red* or *green*. The initial position of the robot is the leftmost end of the environment and the initial belief regarding alarm color is uniform. The model assumes that the alarm turns *red* with probability 0.2, and once it does, it returns to *green* with probability 0.1. The action space is defined as $\{move\ left, move\ right, look\ alarm\}$ and when moving the probability of reaching the target location is 0.8 while the robot stays in the same location with probability 0.2. The reward for reaching the current goal is 0.3 for a corridor length of 3. It is not straightforward how to implement a reward function in order to perform information gain. As discussed in Section 3.1, state-based rewards do not allow for explicitly rewarding information gain. We return to this problem in Section 3.3.1 where we show how to model it in the POMDP-IR framework, which we introduce next.

3.2 POMDPs with Information Rewards

In this section we present the main contribution of this chapter, a framework for rewarding low-uncertainty beliefs without leaving the classic POMDP framework for efficiency reasons.

3.2.1 Models for Active Perception

From the perspective of active perception, as the belief is a probability distribution over the state space, it is natural to define the quality of information based on it. Now, for belief-based rewards one can define the reward ρ directly on the belief space:

$$\rho : \Delta(S) \times A \to \mathbb{R}. \tag{3.1}$$

Araya-López et al. [79] mention several convex functions that are typically used for maximizing information, such as the Kullback-Leibler divergence (also known as relative entropy) with respect to the simplex center c, i.e., a uniform belief ($D_{KL}(b||c)$),

$$\rho_{KL}(b,a) = \sum_{s} b(s) \log \left(\frac{b(s)}{c(s)} \right) = \log(|S|) + \sum_{s \in S} b(s) \log(b(s)), \tag{3.2}$$

or the distance from the simplex center (DSC)

$$\rho_{dsc}(b,a) = \|b - c\|_m \,, \tag{3.3}$$

where m indicates the order of the metric space.

We could use the belief to define a measurement of the expected information gain when executing an action. For instance, a common technique is to compare the entropy of a belief b^t at time step t with the entropy of future beliefs, for instance at $t+1$. If the entropy of a future belief b^{t+1} is lower than b^t, the robot has less uncertainty regarding the true state of the environment [81]. Assuming that the observation models are correct (e.g., unbiased) and observations are independent, this would mean we gained information. Given the models, we can predict the set of beliefs $\{b^{t+1}\}$ we could have at $t+1$, conditional on the robot's action a. If we adjust the POMDP model to allow for reward models that define rewards based on beliefs instead of states, i.e., $r(b,a)$, we can define a reward model based on the belief entropy.

However, a non-linear reward model defined over beliefs like the entropy significantly raises the complexity of planning, as the value function will no longer be piecewise linear and convex, as $V_0 = \max_{\{\alpha_0^a\}_a} b \cdot \alpha_0^a$ no longer holds, and therefore we are no longer able to apply classic solvers to such problems. Moreover, if we want to model problems with two different kinds of goals, both information gain and task performance, reward models defined only over beliefs are not convenient to model such goals.

3.2.2 Rewarding Low-Uncertainty Beliefs

We introduce a different way to reward information gain, while remaining in the classic POMDP framework with PWLC value functions, as discussed in Section 2. Instead of direct reward over beliefs, we introduce the addition of "information-reward" actions to the problem definition, which allow for rewarding the system to obtain a certain level of knowledge regarding particular features of the environment. In this way, we achieve a similar objective as defining reward functions over negative belief entropy, while remaining in the classic POMDP framework.

Hence, the goal of our work is to reward the system for having beliefs that have low uncertainty with respect to particular state factors of interest. For convenience, we assume these are the first l state factors (with $l \leq k$), and hence can be denoted $X_1, X_2, \ldots, X_i, \ldots X_l$. For simplicity, for the moment we assume that each X_i is binary, having values x_i and $\overline{x}_i$. The idea is that we can expand the action space in such a way to allow for rewarding reaching or maintaining a particular low-uncertainty belief over each X_i. In the binary case, we can model this by considering actions that assess whether $X_i = x_i$.

We now define our POMDP-IR model by extending the standard POMDP definition (as provided in Section 2.2) with information-reward actions.

Definition 3.1 (Action space)**.** *We call the original, domain-level action space of the agent A_d. For each $X_i, i \leq l$, we define*

$$A_i = \{commit, null\}.$$

The action space of the POMDP-IR is

$$A^{\mathrm{IR}} = A_d \times A_1 \times A_2 \times \ldots \times A_l.$$

Given this definition, at each time step the agent *simultaneously* chooses a regular domain-level action and an action for each state factor of interest. These actions have no effect on the state transitions nor on the observations, but do affect rewards. The *null* action is added to give the agent the option to not make any assertions regarding the information objectives. It is provided for modeling convenience, as it allows the system designer to consider the agent's task without information objectives.

In the Patrol example, as we are interested in the state of the alarm, C, we denote $X_1 = C$ and set $l = 1$. The action space is now defined as $\{move\ left, move\ right, look\ alarm\} \times \{commit, null\}$.

Definition 3.2 (Information rewards)**.** *We call the original reward function of the POMDP R_d. The POMDP-IR's reward function R^{IR} is the sum of R_d and a reward R_i for each $X_i, i \leq l$:*

$$R^{\mathrm{IR}}(X, A) = R_d(X, A_d) + \sum_{i=1}^{i=l} R_i(X_i, A_i).$$

Each $R_i(X_i, A_i)$ is defined as

$$\begin{aligned} R_i(x_i, \mathit{commit}) &= r_i^{\mathrm{correct}}, \\ R_i(x_i, \mathit{null}) &= 0, \\ R_i(\overline{x}_i, \mathit{commit}) &= -r_i^{\mathrm{incorrect}}, \\ R_i(\overline{x}_i, \mathit{null}) &= 0, \end{aligned}$$

with $r_i^{\mathrm{correct}}, r_i^{\mathrm{incorrect}} > 0$.

The upshot of this reward function is that at every time step, the agent can choose to either execute only a domain-level action ($A_i = \mathit{null}$), or in addition also receive reward for its belief over X_i ($A_i = \mathit{commit}$). Intuitively, r_i^{correct} is the reward the agent can obtain for guessing the state of X_i correctly, and $r_i^{\mathrm{incorrect}}$ is the penalty for an incorrect guess. We choose r_i^{correct} and $r_i^{\mathrm{incorrect}}$ in such a way that the agent only benefits from guessing when it is certain enough about the state of X_i. When the agent is not certain enough, it can simply choose the *null* action (in combination with any domain-level action). However, the possibility of obtaining information rewards by having a low-uncertainty belief over X_i will steer the agent's policy towards such beliefs.

3.2.3 Choosing the Information Rewards Parameters

From Definition 3.2 we can see that the expected reward for each information-rewarding action is:

$$R_i(b, \mathit{commit}) = b_i(x_i) r_i^{\mathrm{correct}} - (1 - b_i(x_i)) r_i^{\mathrm{incorrect}}, \tag{3.4}$$

$$R_i(b, \mathit{null}) = 0, \tag{3.5}$$

using a short-hand notation $b_i(x_i)$ to indicate $b_i(X_i = x_i)$. Thus, the expected reward of choosing *commit* is only higher than the *null* action when

$$b_i(x_i) r_i^{\mathrm{correct}} - (1 - b_i(x_i)) r_i^{\mathrm{incorrect}} > 0, \tag{3.6}$$

which, by rearranging, becomes:

$$b_i(x_i) > \frac{r_i^{incorrect}}{r_i^{correct} + r_i^{incorrect}}. \tag{3.7}$$

This inequality indicates the range over which the agent will execute a *commit* action, and we can see that the values of r_i^{correct} and $r_i^{\text{incorrect}}$ are decisive to determine this range.

If we assume that we want to reward the agent for having a degree of belief of at least β regarding a particular X_i, i.e., $b_i(x_i) > \beta > 0$, β is computed according to Equation (3.7), and therefore:

$$\beta = \frac{r_i^{\text{incorrect}}}{r_i^{\text{correct}} + r_i^{\text{incorrect}}}, \tag{3.8}$$

$$\Leftrightarrow r_i^{\text{correct}} = \frac{(1-\beta)}{\beta} r_i^{\text{incorrect}}. \tag{3.9}$$

Equation (3.8) tells us the relation between r_i^{correct} and $r_i^{\text{incorrect}}$. Their precise values depend on each problem, and on the insight of the system designer (just as the domain-level rewards), taking into account his knowledge of the models and the environment and calibrating it with the original reward model R_d. For instance, in the PATROL example, if the robot is rewarded too much for maintaining a low-uncertainty belief regarding the alarm, it will prefer to look at the alarm constantly, ignoring its patrol task.

The effectiveness of this scheme depends on whether in the particular POMDP reaching a particular β is possible at all, due to sensory limitations. For instance, if an agent's sensors do not observe X_i at all, or provide too noisy information, beliefs in which $b_i(x_i) > \beta$ can be unreachable given an initial belief state. In point-based POMDP methods that operate on a pre-defined belief set, this condition can be checked easily, and β be adjusted accordingly.

A second option is to define several β levels for an X_i, which reward the agent for reaching different levels of certainty. Care needs to be taken, however, to ensure that the agent will try to reach the highest β possible. How specific values for r_i^{correct} and $r_i^{\text{incorrect}}$ are obtained is up to the system designer, given that the relation between them is as in Equation (3.8). Table 3.1 defines possible ways to compute values for r_i^{correct} and $r_i^{\text{incorrect}}$ for different values of β, using different criteria. These values are taken from the given information measuring functions, by looking to the value of those functions at the specific values of β. Using this values as r_i^{correct} then $r_i^{\text{incorrect}}$ can be computed. As the potential reward for higher β is higher as well, the agent is guided to reach the highest certainty level possible.

	$D_{KL}(b\|\|c)$ (3.2)		DSC, $m=1$ (3.3)		DSC, $m=2$ (3.3)		DSC, $m=\infty$ (3.3)	
β	r_i^{correct}	$r_i^{\text{incorrect}}$	r_i^{correct}	$r_i^{\text{incorrect}}$	r_i^{correct}	$r_i^{\text{incorrect}}$	r_i^{correct}	$r_i^{\text{incorrect}}$
0.60	0.03	0.04	0.20	0.30	0.02	0.03	0.10	0.15
0.75	0.19	0.57	0.50	1.50	0.12	0.38	0.25	0.75
0.90	0.53	4.78	0.80	7.20	0.32	2.88	0.40	3.60
0.99	0.92	91.00	0.98	97.02	0.48	47.54	0.49	48.51

TABLE 3.1: Example rewards for varying β.

3.2.4 Extensions and Variations

When defining the POMDP-IR framework in Section 3.2.2, we assumed binary state factors for simplicity. Our framework can be easily generalized to multi-valued state factors, by defining one or multiple *commit* actions for a state factor X_i. In the binary case, we assign a positive reward to one element in the domain of X_i, and a negative one to the other. In the multi-valued case, a single *commit* action can result in positive reward for multiple values of X_i when each of those values is equally desirable, by trivial generalization of Definition 3.2. In the case that distinct values of X_i differ in desired reward, the definition of the information-reward action can be extended to

$$A_i = \{commit_1, commit_2, \dots, commit_j, null\}.$$

In this way, certain values of X_i can be rewarded differently, and sets of values that share the same reward can be grouped under the same $commit_j$ action.

In our framework, we considered the general case in which an agent at each time step can opt for a *commit* action or not: information rewards are obtainable at each time step. However, possible use cases of the POMDP-IR framework might consider other scenarios. For instance, it might be desirable to reward the agent only a single time for each *commit* action. Such a scenario is easily implemented by adding a Boolean bookkeeping variable that keeps track whether a *commit* action has been performed or not. By making the information reward dependent on the value of this variable, the agent will optimize its course of action to execute *commit* only once.

In a similar vein, in certain active-sensing or sensor-management scenarios an agent might have to decide whether to execute *commit* after a predefined number of steps. By encoding the time step as an extra state factor or by defining time-dependent action spaces, the POMDP-IR framework can be used in these types of scenarios as well.

3.3 Experiments

We perform an extensive experimental evaluation of the POMDP-IR framework. First, we illustrate how information rewards influence agent behavior in the Patrol example. Second, we compare the performance of POMDP-IR against another recent framework for information gain with POMDPs on a benchmark problem.

3.3.1 Information Rewards in the Patrol Example

At this point we return to the Patrol example presented in Section 3.1.1. In this example we want to solve a problem where an agent needs to patrol an environment while at the same time considering the uncertainty regarding the state of an alarm device. This illustrates the tradeoff between task performance and information gain that we want to tackle.

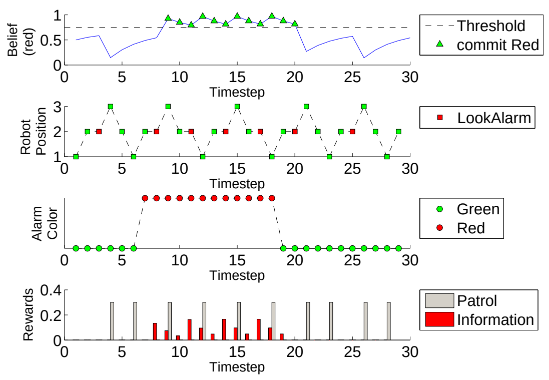
3.3.1.1 Model

The POMDP model is an extension of the model presented in Section 3.1.1 and is depicted in Fig. 3.1b. The domain-level action space is $A_d = \{$*move left*, *move right*, *look alarm*$\}$ and we extend it with an information-reward action regarding the alarm state factor C, $A_1 = \{commit, null\}$. The reward function is a sum of rewards for both types of actions, $R^{\mathrm{IR}} = R_d + R_1$, according to Definition 3.2. R_1 rewards the robot if its belief whether the alarm is red exceeds the threshold β. The reward values will depend on each particular problem, namely the environment size and the threshold defined in each case. Information reward values depends on the value of β used in each experiment, and is up to the system designer to choose a criteria for those values. In our particular case, values are taken from the first column of Table 3.1. The patrol reward encoded in R_d is 0.3 for corridor length 3 and 0.5 for corridors of length 5. We keep initial conditions and models constant for all experiments to be able to showcase the effect of the belief threshold(s).

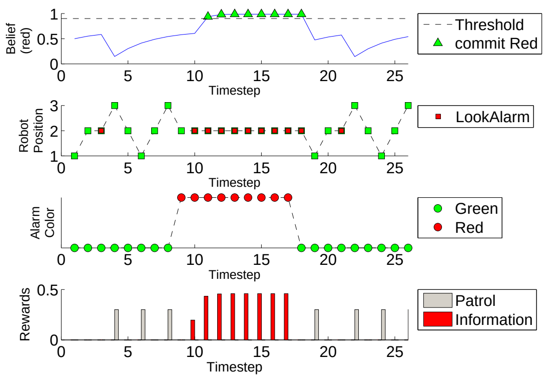
3.3.1.2 Results

Single threshold

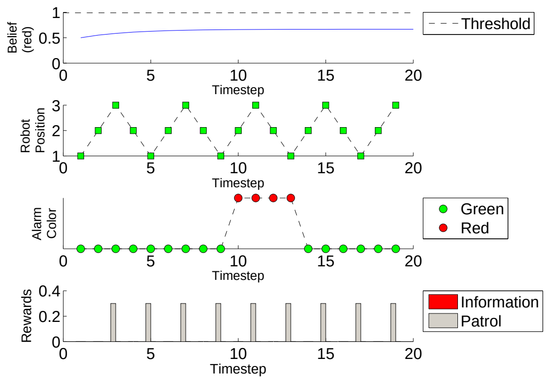
Considering that the original behavior of the robot would be to patrol the corridor up and down, we note a change in its behavior in Figs. 3.2a and 3.2b, as it continues to perform the patrolling task, but with some intermediate stops (as seen in the 2nd row of



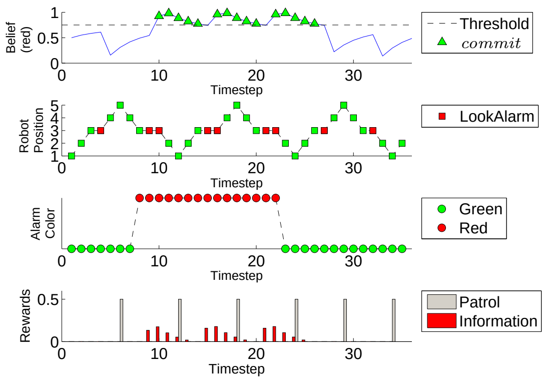
(A) Threshold $\beta = 0.75$



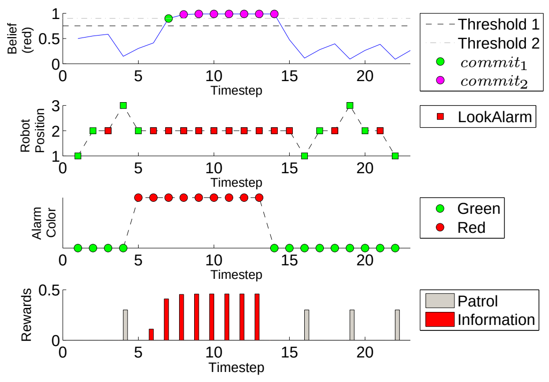
(B) Threshold $\beta = 0.9$



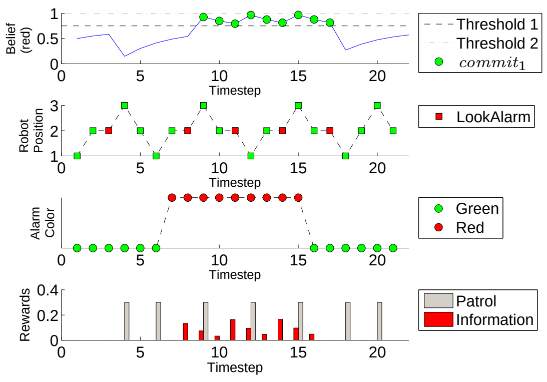
(C) Threshold $\beta = 0.99$



(D) $\beta = 0.75$, corridor of length 5



(E) Thresholds $\beta_1 = 0.75$ and $\beta_2 = 0.9$



(F) Thresholds $\beta_1 = 0.75$ and $\beta_2 = 0.99$

FIGURE 3.2: PATROL problem results. Each figure shows belief evolution (top row), robot position (2nd row), true alarm color (3rd row) and received rewards (bottom row). Corridor length is 3 except for (d).

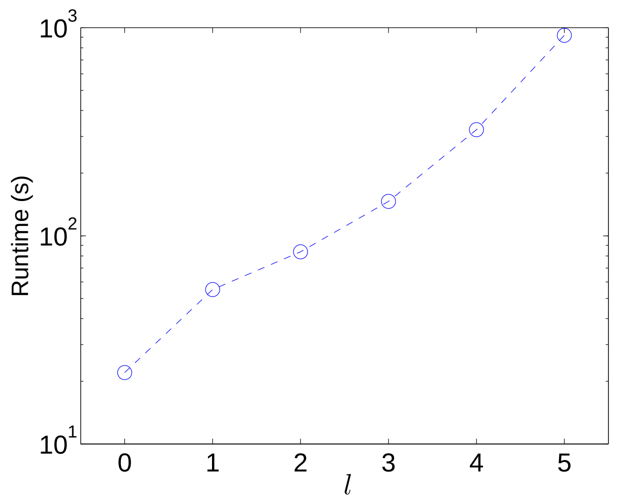


FIGURE 3.3: Performance measures for PATROL problem with 5 alarms, and l *commit* actions with $\beta = 0.9$.

plots). The bottom row details the patrol (R_d) and information reward (R_1) the robot receives. Fig. 3.2a shows results where the threshold is set to $\beta = 0.75$ and the robot clearly tries to maintain up-to-date information on the alarm, by looking at it every time it passes by. In this case the threshold is such that the robot has enough time to continue performing the original patrolling task and still return to the alarm position before its belief falls below the threshold. On the contrary, we note that with $\beta = 0.9$ (Fig. 3.2b), the threshold is higher than in the first case and thus the behavior changes slightly. Every time the robot observes a green alarm it continues performing the patrolling task. However, if it observes a red alarm, it decides to stay and keep looking at it. On the other hand, with $\beta = 0.99$ (Fig. 3.2c) the threshold is so high that even if the robot stays looking at the alarm, it will never get to levels above the threshold. Therefore, it will only focus on patrolling, illustrating the need to choose the thresholds carefully. In Fig. 3.2d we extend the simulation to a corridor length of 5, with $\beta = 0.75$. In this setting the behavior is similar to the previous results, except for the fact that when the robot decides to look at the alarm it spends 2 time steps.

Multiple thresholds

Next, we test a model with two information-reward actions, each corresponding to a different threshold. We see in Figs. 3.2e and 3.2b that the behavior of the robot is equivalent to the maximum threshold it can reach. For instance, in Fig. 3.2e with $\beta_1 = 0.75$ and $\beta_2 = 0.9$ the robot's behavior is equivalent to Fig. 3.2b ($\beta = 0.9$). On the other hand, in Fig. 3.2f the thresholds $\beta_1 = 0.75$ and $\beta_2 = 0.99$ are set, but as seen before the system will ignore the second threshold as the belief will never reach it, and therefore its behavior will be as if there were only one threshold $\beta = 0.75$.

Scalability

Our approach inflates the action space, which will lead to increased computation time. Therefore, it is also important to measure the computational impact of adding information-reward actions to problems, shown in Fig. 3.3. Note, however, that POMDP solving scales linearly with the number of actions, compared to exponentially with the number of observations. To allow for a fair comparison we must compare models with the same domain actions and a variable number of information-reward actions. We create a PATROL model which includes 5 alarms and test the planner's performance when including different number of *commit* actions. We note that the planner's performance degrades exponentially, as expected, although with more *commit* actions the system will receive higher value (not shown).

3.3.1.3 Discussion

With this toy problem we showed the advantages of our approach, by directly incorporating the concept of information gain in a classical POMDP setup. We can observe that although information-reward actions do not directly affect state transitions they have a clear effect on the robot's general behavior, as it will change its physical actions according to not only the original task to solve, but also to the uncertainty level in the system and the thresholds.

As expected, the robot faces a tradeoff between patrol task completion and information gain regarding the alarm. However, this is dependent on a number of different parameters, the desired certainty level in particular. Generally, when it is possible to reach the desired threshold the robot will interrupt patrolling to observe the alarm and get information. On the contrary, if the desired level is unreachable or if it is so low that the belief converges to a value above the threshold without the need for direct observation, the robot does not need to waste time looking at the alarm and only performs the patrol task.

3.3.2 Comparison With ρPOMDPs on the Rock Diagnosis Problem

Next, we compare our work to a POMDP-based framework for information gain, the ρPOMDP framework [3]. We present results in a scenario where information gain plays an important role: the ROCK DIAGNOSIS problem [3], which is a variation of the ROCK SAMPLING problem [68].

The approach in ρPOMDPs extends POMDPs to allow for direct belief-based rewards. It is noted that the belief-MDP formulation already accounts for a reward over beliefs, $(r(b,a) = \sum_{s\in S} b(s)R(s,a))$, but formulated in terms of state-based reward, in order to maintain the PWLC property. Therefore, ρPOMDPs generalize POMDPs in order to be able to deal with other types of belief-based rewards, under the assumption that those rewards are convex. Algorithms are modified in order to deal with rewards represented as a set of vectors. For PWLC reward functions, this is an exact representation. For non-PWLC rewards it is an approximation which improves as the number of vectors used to represent it increases. As we saw in Section 3.2 our approach also considers the belief-MDP formulation, but in a way that represents rewards for information gain in terms of state-based rewards, allowing us to use existing methods without changes.

In Araya-López [3]'s experiments an extension of point-based methods is used to accommodate this generalization and tested with different reward functions. For comparison we present here their results with two different reward functions, entropy (PB-Entropy) and linear (PB-Linear). The entropy-based reward is the Kullback-Leibler divergence with the uniform distribution as reference, while the linear-based reward is an approximation of the entropy-based reward which corresponds to the L_∞-norm of the belief.

3.3.2.1 Rock Diagnosis **problem definition**

The Rock Diagnosis is an information-gathering problem, in which a rover receives a set of rock positions and must perform sampling procedures in order to reduce uncertainty regarding the type of several rocks spread in an environment. Each rock can have a *good* or a *bad* value and we want the system to be capable of returning low-uncertainty information regarding each rock's type. Hence, our objective is to produce policies which guide the rover through the environment, allowing it to observe each rock's type. The map of the environment is considered to be a square grid with size p and there are q rocks in the environment to analyze. We refer to [3] for further details on the Rock Diagnosis problem.

In the original formulation the domain-level action space is $A_d = \{$*north*, *east*, *south*, *west*, *check$_1$*, *check$_2$*, …, *check$_q$*$\}$ and in our POMDP-IR formalization we use the extension described in Section 3.2.4 to include a set of information-reward actions for each rock: $A_i = \{commit_1, commit_2, null\}$, with $1 \leq i \leq q$. In this case, $commit_1$ and $commit_2$ encode the action that a rock is *good* or *bad*, respectively. The intuition is that the policy will try to reduce the uncertainty regarding each rock's type, in order to get a higher value, thereby achieving the problem's final objective.

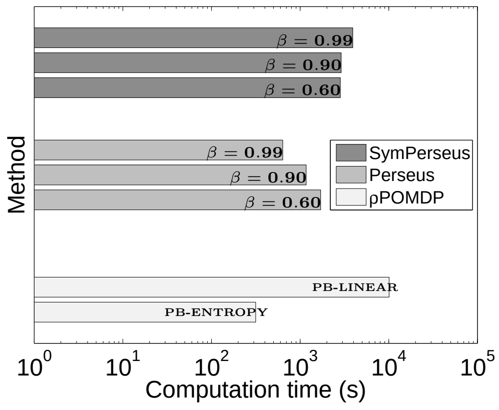
ρPOMDP		POMDP-IR (Perseus)		POMDP-IR (Symbolic Perseus)	
Method	return [$nats$]	β	return [$nats$]	β	return [$nats$]
$q = 3; p = 3$					
PB-Entropy	1.58 ± 0.25	0.6	2.065 ± 0.099	0.6	1.978 ± 0.092
PB-Linear	2.06 ± 0.03	0.9	2.079 ± 0.000	0.9	1.988 ± 0.223
		0.99	1.815 ± 0.334	0.99	2.035 ± 0.124
$q = 3; p = 6$					
PB-Entropy	0.76 ± 0.09	0.6	1.763 ± 0.463	0.6	1.386 ± 0.000
PB-Linear	0.79 ± 0.08	0.9	1.790 ± 0.417	0.9	1.580 ± 0.213
		0.99	1.662 ± 0.390	0.99	1.156 ± 0.114
$q = 5; p = 7$					
PB-Entropy	0.37 ± 0.09	0.6	—	0.6	1.580 ± 0.313
PB-Linear	0.53 ± 0.03	0.9	—	0.9	1.553 ± 0.298
		0.99	—	0.99	1.737 ± 0.456

Table 3.2: Rock Diagnosis results, comparing POMDP-IR with ρPOMDP [3].

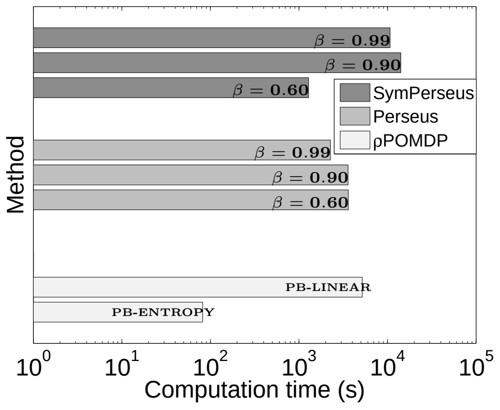
3.3.2.2 Experimental Setup

We computed policies for this problem using Perseus [67] and Symbolic Perseus [2], with $\gamma = 0.95$ and $\epsilon = 10^{-3}$, where ϵ is the convergence criterion used both by Perseus and Symbolic Perseus. We ran a set of 10 repetitions of 100 trajectories of 100 steps, re-sampling the belief set containing 5000 beliefs at each repetition. Given the information-gathering nature of this problem, the performance criterion used to evaluate the agent's behavior should also be an information measure. In particular, the Kullback-Leibler divergence between the belief distribution and the uniform distribution is used (3.2).

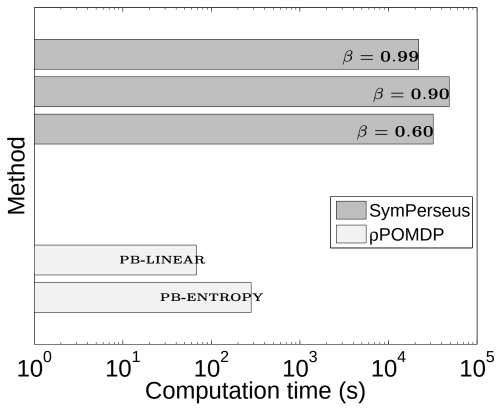
In this problem the performance criterion only considers the available information at the end of the trajectory, as the objective is to disambiguate the state of all the rocks' types. Therefore, we present results as the average of the Kullback-Leibler divergence of the belief distribution in the last time step at each trajectory. The unit used to measure information is *nat* (natural unit for information entropy), since we use natural logarithms. Also note that the target variable is the rock type, thus our information measure considers only the belief over state factors which represent each rock's type. Therefore, the maximum possible final reward is $q \log(2)$, precisely when there is no uncertainty regarding any rocks' type.



(A) Map with $q = 3; p = 3$.



(B) Map with $q = 3; p = 6$.



(C) Map with $q = 5; p = 7$.

FIGURE 3.4: ROCK DIAGNOSIS computation times. ρPOMDP timings are taken from [3] and hence cannot be compared directly.

3.3.2.3 Results

We present in Table 3.2 a set of results for different maps with varying values of β, and also, for comparison, results presented by [3]. The implementation of POMDP-IR with flat Perseus could not handle the larger map ($q = 5; p = 7$).

We can see that, in general, POMDP-IR achieves better results than the two ρPOMDP variations. The total return with ρPOMDP deteriorates with increasing problem size due to sampling a larger state space with the same number of belief points. However, we note that POMDP-IR results worsen less, which may be explained by the fact that ρPOMDPs approximate the reward function with linear vectors, imposing some error on the original reward function. This results in a loss of quality in larger problems, since the same number of points is used to approximate a higher dimensional belief space. In turn, we directly consider reward over states, thus staying in the traditional POMDP formulation. Using fewer approximations we can achieve better performance.

Our results show better performance in the smaller map, with similar results with Perseus and Symbolic Perseus. Note the particular case of Perseus with $q = 3, p = 3, \beta = 0.9$ where we always achieve perfect information about all rocks in the environment. There is a trend for Perseus to perform slightly better than Symbolic Perseus. However, we note that Symbolic Perseus implements an additional approximation by maintaining a factored belief representation which may explain the difference in results.

We may then state that, although our focus is on adding information rewards to regular POMDPs, we also improve performance on these pure information-gathering problems. The problem size can be an issue, as we see in the largest scenario that could not be handled using a flat method like Perseus. However, our focus is on factored representations and we showed that we can perform well using Symbolic Perseus.

We also include a comparison between average computation times in Fig. 3.4, where we included ρPOMDP timings reported by Araya-López [3] which precludes direct comparisons. The increase in computation time in POMDP-IR implementations is a natural consequence of the increase in the action space size (as the growth is exponential in the number of action factors), but we present a reasonable tradeoff between computation time and average value in this particular case.

3.4 Case-study: Robot-Assisted Surveillance

The main motivating application for our research is robot-assisted surveillance. In such scenarios, a robot can be seen as a mobile extension of a camera network, helping to improve confidence of detections. For instance, due to several reasons (lighting, distance to camera, capture quality, etc.) the image quality may vary from camera to camera, leading to different uncertainty rates when detecting robots, persons or different features in the environment. Also, in a surveillance system the camera network will typically not cover all the environment, leaving some blind spots. Such problems can be solved or at least improved by using mobile sensors, which can move to provide (improved) sensing in particular areas.

In our case study we are interested in a surveillance system which detects persons with a particular feature. Feature detection is achieved through image processing methods that detect persons moving in the environment. In our case, for simplicity, we consider as feature whether a person is wearing red upper body clothing. Note that our methods are rather independent of the actual feature detector used, as long as false positive and false negative rates can be estimated.

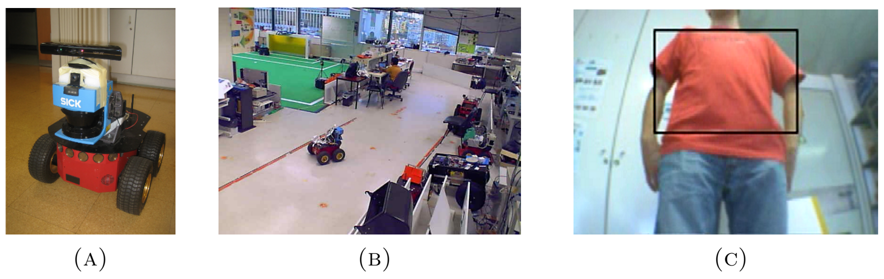


FIGURE 3.5: Experimental setup. (a) Robot Pioneer 3-AT with camera and laser range finder onboard. (b) Example of an image captured by the camera network. (c) Feature detection by the robot.

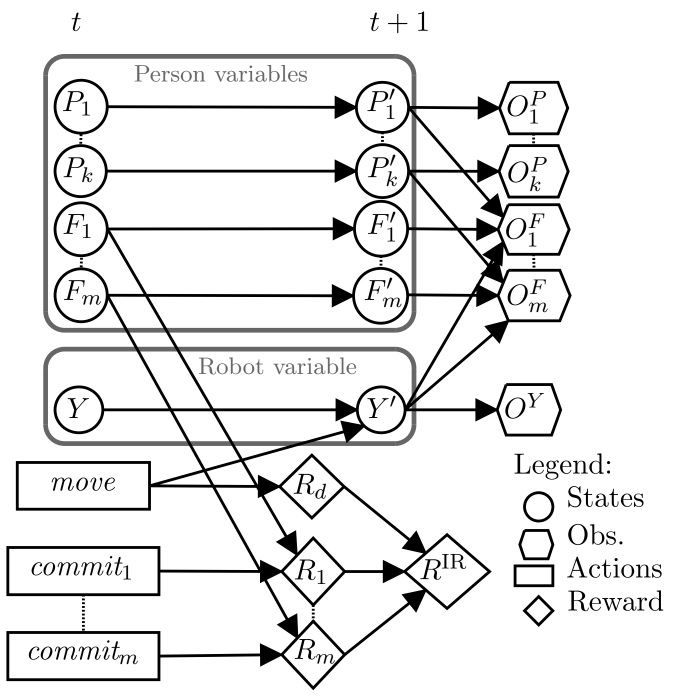


FIGURE 3.6: Two-stage dynamic Bayesian network representation of the proposed POMDP-IR model. In this figure we assume $m \equiv k$ for simplicity, i.e., each person has only one feature.

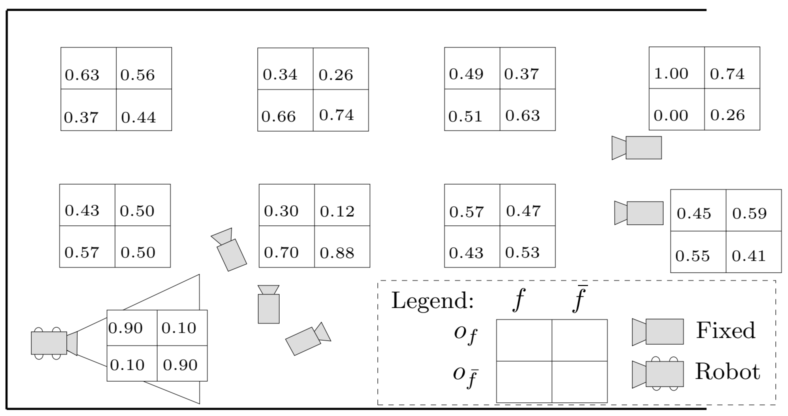


FIGURE 3.7: Environment map with observation model learned for fixed cameras and predefined model for robot camera. Each matrix is positioned at each node in the map, and represents $p(O^F|F)$, that is, probabilities of positive (o_f) or negative ($o_{\bar{f}}$) observations given the true state of the feature.

3.4.1 Model and Experimental Setup

We implemented our case study in a testbed for NRS [82], which consists of a camera network mounted on the ceiling of the lab (Fig. 3.5b), and mobile robots (Pioneer 3-AT) with onboard camera and laser range finder (Fig. 3.5a). The POMDP controllers are computed using Symbolic Perseus [2].

Figure 3.6 depicts a graphical representation of the model for time steps t and $t+1$. As before, we encode the environment in several state variables, depending on how many people and features the system needs to handle. The locations of people and robot are represented by a discretization of the environment, for instance a topological map. Graphs can be used to describe topological maps with stochastic transitions and hence sets of nodes represents the robot location Y and k people locations P_1 through P_k. In particular, we run our experiments in the lab shown in Fig. 3.5, building a discretized 8-node topological map for navigation and position identification. Besides a person's location, we represent a set of m features where each feature f is associated with a person, for instance whether it matches a visual feature. We assume each person has at least one feature, hence $m \geq k$, and features are represented by variables F_1 through F_m.

We assume a random motion pattern for each person, as we do not have prior knowledge about the person's intended path, in which the person can either stay in its current node with probability 0.6, or move to neighboring nodes with the remaining probability mass split equally among them. Such a representation allows us to model movement constraints posed by the environment (for instance, corridors, walls or other obstacles). We also need to take into account the uncertainty in the robot movement due to possible errors during navigation or unexpected obstacles present in the environment. In this model when the robot moves either it arrives at its destination with probability 0.6 or stays in the same node with probability 0.4. The value of the feature nodes $F_1, \ldots, F_m$ have a low probability of change (0.01), as it is unlikely that a particular person's characteristics changes. For each state variable we define a set of observations. Each observation $o_1^p \in O_1^P$ through $o_k^p \in O_k^P$ and $o^y \in O^Y$ indicates an observation of a person or robot close to a corresponding $p_k \in P_K$ resp. $y \in Y$. The observations $o^f \in O^F$ indicate whether a particular feature is observed.

The key to cooperative perception lies in the observation model for detecting features. The false negative and false positive rates are different at each location, depending on conditions such as the position of sensors, their field of view, lighting, etc. For detecting certain features, mobile sensors have a higher accuracy than fixed sensors, although with a smaller field of view. Therefore, the observation model differs with respect to person and robot location. In particular, if a person is observed by the robot (illustrated in Fig. 3.5c) the probability of false negatives $P(O^F = o^{\bar{f}} | F = f)$ or false positives $P(O^F = o^f | F = \bar{f})$ is low. This is an important issue in decision making, as the presence of a mobile sensor will be more valuable in areas where fixed sensors cannot provide high accuracy. In Fig. 3.7 we show the uncertainty related to observations for each node considered. Note that the observation uncertainty has been estimated for the fixed sensors but we assume a constant observation model for the mobile sensor.

This concludes the basis for the models used in our case study. In the following, we will present a formalization of this models with information-gain actions. We present simulations results using the POMDP-IR framework. A previous approach for the same case-study, but using a slightly different model can be found in Spaan et al. [20].

3.4.2 Experiments with Information Rewards

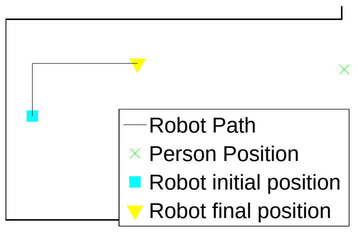
Here we apply the POMDP-IR framework to the robot-assisted surveillance problem, showcasing its behavior in larger and more realistic problem domains. In this problem there are no concurrent goals, unlike the Patrol problem, where we had to perform information gain while performing other tasks. Therefore, in this model we include two possible sets of information-reward actions, $A_1 = \{\mathit{commit}_1, \mathit{null}\}$, $A_2 = \{\mathit{commit}_2, \mathit{null}\}$.[1] A_1 encodes the action that the person is wearing red, while A_2 relates to the opposite (asserting that state factor F_1 is false).

We present in Fig. 3.8 some experiments where the threshold is $\beta = 0.75$ (Exp. A and Exp. B) and $\beta = 0.6$ (Exp. C and Exp. D). For each experiment, we show the robot's path, initial position of robot and person, cumulative reward, and the evolution of the belief over the feature of interest.

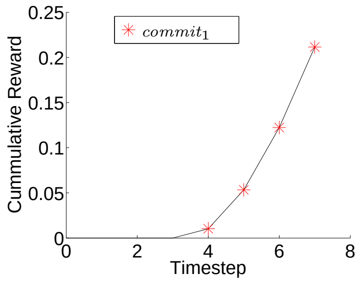
Experiment A presents a case in which a person wearing red is detected in an area with low uncertainty. We note that the belief increases rapidly, and the robot does not need to approach the person for the system to have enough information to start applying information-reward actions. The belief crosses the threshold at time step 4, and from that moment on the *commit*$_1$ action is applied, indicating that the system decides to classify this person as wearing red. The cumulative reward does not have a linear shape as the reward for the information-reward action is higher as the belief increases.

Experiment B shows a case where a person is detected not wearing red in an area with higher uncertainty. Therefore, the system decides to move the robot near the person to observe what is happening before executing *commit*$_2$ actions. In both experiments with

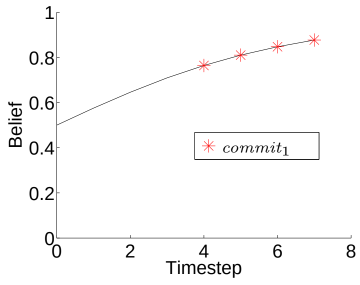
[1]An alternative option would be to implement the extension described in Section 3.2.4.



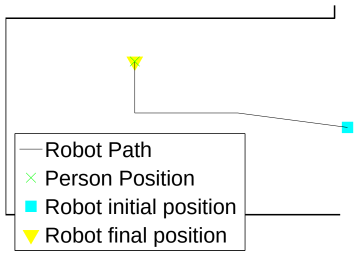
(A) Exp. A: Robot's path.



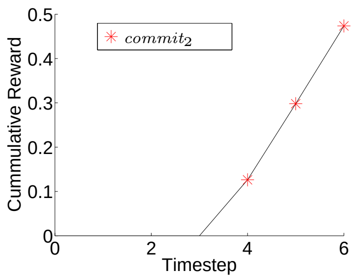
(B) Exp. A: $\sum_t \gamma^t r_t$.



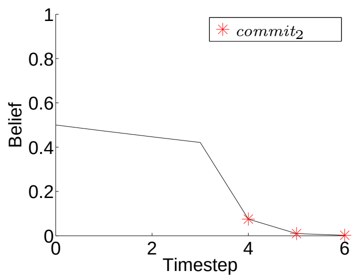
(C) Exp. A: $b_i^t(F_1 = red)$.



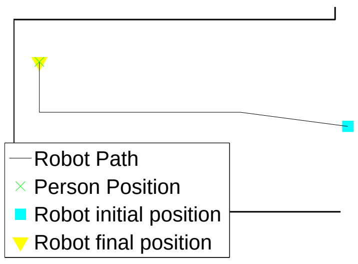
(D) Exp. B: Robot's path.



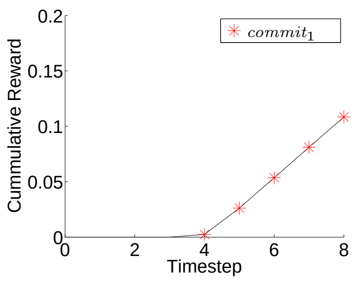
(E) Exp. B: $\sum_t \gamma^t r_t$.



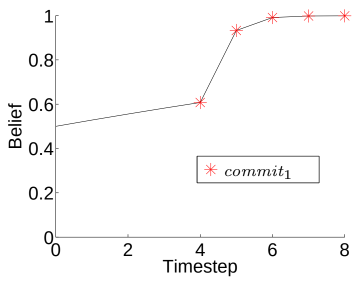
(F) Exp. B: $b_i^t(F_1 = red)$.



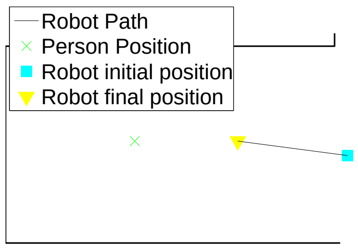
(G) Exp. C: Robot's path.



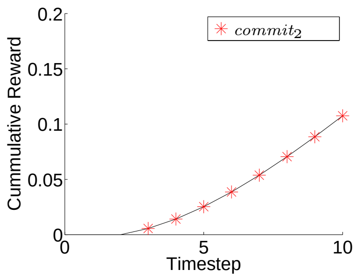
(H) Exp. C: $\sum_t \gamma^t r_t$.



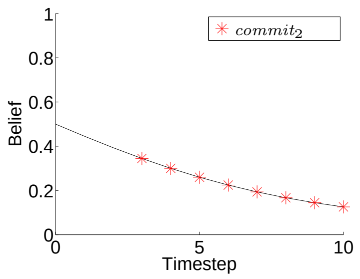
(I) Exp. C: $b_i^t(F_1 = red)$.



(J) Exp. D: Robot's path.



(K) Exp. D: $\sum_t \gamma^t r_t$.



(L) Exp. D: $b_i^t(F_1 = red)$.

FIGURE 3.8: POMDP-IR simulation results for the case study. Experiments A and B: $\beta = 0.75$. Experiments C and D: $\beta = 0.6$. Action *commit*$_1$ corresponds to asserting that the person is wearing red while action *commit*$_2$ asserts the opposite.

$\beta = 0.75$ we see that, in fact, the threshold is respected, and *commit* actions are taken only when the belief crosses the specified value.

In the case in which $\beta = 0.6$ the system takes into account the different threshold. In Exp. C a person wearing red is detected in a higher uncertainty area. The robot moves towards the person for better identification, but from the moment when the belief crosses the threshold, the action $commit_1$ is chosen. In Exp. D the system detects the person without red clothing (action $commit_2$) and the robot moves but stops half way (c.f. Exp. B).

3.4.3 Discussion

The case study presented shows an example of a real-world problem where information gain performs an important role. In general, the behavior of the system is as expected, as both in simulation as well as in the reality the system tries to optimize the robot behavior in order to detect information in the system. When the networked cameras do not provide enough information the controller asks the robot to move towards the person.

3.5 Related Work

In this section, we present an overview of the relevant literature. We discuss related work in applying POMDPs to robotic applications, followed by a discussion of how work in active sensing relates to our contributions. Finally, we discuss how the POMDP-IR framework compares to related POMDP models for rewarding information gain.

3.5.1 POMDPs in a robotic context

Techniques for decision-theoretic planning under uncertainty are being applied more and more to robotics [83]. Over the years, there have been numerous examples demonstrating how POMDPs can be used for mobile robot localization and navigation, see for example work by Roy et al. [29], Simmons and Koenig [84]. Emery-Montemerlo et al. [85] demonstrate the viability of approximate multiagent POMDP techniques for controlling a small group of robots to catch an intruder, while more recently Amato et al. [86] use similar frameworks in larger multi-robot domains to perform cooperative tasks. Messias et al. [87] models asynchronous sequential decision-making for teams of soccer robots, later extended to partially observable scenarios [88]. Capitán et al. [89] show how POMDP task auctions can be used for multi-robot target tracking applications.

These types of applications are potentially suited for the POMDP-IR framework, as they typically involve reasoning about the belief regarding the target's location.

A relevant body of work exists on systems interacting with humans driven by POMDP-based controllers. Fern et al. [90] propose a POMDP model for providing assistance to users, in which the goal of the user is a hidden variable which needs to be inferred. POMDP-based models have been applied in a real-world task for assisting people with dementia, in which users receive verbal assistance while washing their hands [91], and to high-level control of a robotic assistant designed to interact with elderly people [92, 93]. Merino et al. [94] develop a real-time robust person guidance framework using POMDP policies which includes social behaviors and robot adaptation by integrating social feedback. Interacting with humans whose goals and objectives might need to be estimated provides opportunities for the POMDP-IR approach.

3.5.2 Active sensing

Active sensing is typically defined as the problem of finding control strategies that dynamically adapt sensing parameters as the sensor interact with the environment [95]. Information gain has been studied in literature under the this framework [96], which can be formalized as acting so as to acquire knowledge about certain state variables.

A large amount of research effort has been put in approaches to robot localization using active methods. Burgard et al. [81] propose an active localization approach providing rational criteria for setting the robot's motion direction and determining the pointing direction of the sensors so as to most efficiently localize the robot, while Roy et al. [97] use environment information to minimize uncertainty in navigation. Velez et al. [98] add informative views of objects to regular navigation and task completion objectives. Another aspect which may be included in active sensing is where to position sensors to maximize the level of information of observations. Krause and Guestrin [99] and Krause et al. [100] exploit submodularity to efficiently tradeoff observation informativeness and cost of acquiring information, while Krause et al. [101] apply such methods to sensor placement in large water distribution networks. More recently, Natarajan et al. [102] maximize the observation of multiple targets in a multi-camera surveillance scenario.

Active sensing problems typically consider acting and sensing under uncertainty, hence POMDPs offer a natural solution to model such problems. Work on active sensing using POMDPs includes collaboration between mobile robots to detect static objects [103], combining a POMDP approach with information-theoretic heuristics to reduce uncertainty on goal position and probability of collision in robot navigation [14] or distributed decision processes for multirobot systems in which robots decide how to act in the environment and what to communicate with others [104]. Spaan and Lima [105] consider objectives such as maximizing coverage or improving localization uncertainty when dynamically selecting a subset of image streams to be processed simultaneously. In the field of computer vision, Atanasov et al. [106] use POMDPs for planning to detect semantically important objects and estimates their pose by actively controlling the point of view of a mobile depth camera.

3.5.3 POMDP frameworks for rewarding information gain

Multi-objective problems which include information gain have recently been proposed with POMDP formulations. Different criteria have been proposed and used for decision-making for uncertainty minimization. Mihaylova et al. [96] review some of these criteria in the context of active sensing. They consider a multi-objective setting, linearly trading off expected information extraction with expected costs and utilities. Similar to our setting, the system designer is responsible for balancing the two objectives.

Using the belief MDP formulation, it is possible to directly define a reward $r(b, a)$ over beliefs. While we may assume this function to be convex, which is a natural property for information measures, there is still the need to approximate it by a PWLC function [79]. Thus, we cannot directly model non-linear rewards as belief-based without approximating such functions.

Eck and Soh [107] introduce hybrid rewards, which combine the advantages of both state and belief-based rewards, using state-based rewards to encode the costs of sensing actions and belief-based rewards to encode the benefits of sensing. Such a formulation lies out of a traditional solver's scope. While we also consider multi-objective problems, solving our models is independent of which solver is used. We are interested in an approach that while staying in the standard POMDP framework, and thus with PWLC value functions, is still able to perform multi-objective problems, where only some tasks are related to information gain. We do that without using purely belief-based rewards, such that our framework can be plugged in any traditional solver. Also, we are able to impose thresholds on the amount of uncertainty we desire for particular features in the environment.

Similar to our *commit* actions, Williams and Young [108] propose the use of *submit* actions in slot-filling POMDPs, in which a spoken dialog system must disambiguate the internal state of the user it is interacting with. In these scenarios information gain is a mean for the system to be certain about the user's desire but there is no clear separation between domain-level actions and information-reward actions like in the POMDP-IR framework. Moreover, *submit* actions lead to an absorbing state, which indicates a final goal for the system, and do not allow for continuously monitoring of the information state in the system. Our system is intended to be parallel to normal task execution, hence the *commit* actions do not have any effect on the state of the environment. Furthermore, in contrast to their work, we provide guidance on how such actions should be rewarded.

3.6 Discussion

In this chapter we presented an approach to deal with information gain in active cooperative perception problems, which involve cooperation between several sensors and a decision maker. We based our approach on POMDPs due to their natural way of dealing with uncertainty, but we face the problem of how to build a POMDP which explicitly performs information gain.

There are several ways to reward a POMDP for information gain, such as using the negative belief entropy. Although resulting value functions are shown to remain PWLC if the reward is PWLC [79], that is not generally the case in belief-based rewards such as negative entropy. Moreover, we are interested in modelling multi-objective problems where the system must complete a set of different tasks, in which only part of them might be related to direct information gain. However, that can be cumbersome with a belief-dependent reward implementation.

We proposed a new framework, POMDP-IR, which builds on the classic POMDP framework, and extends its action space with actions that return information rewards. These rewards depend only on a particular state factor, and are defined over states, not beliefs. However, their definition ensures that the reward for information-reward actions is positive only if the belief regarding the state factor is above a threshold. In this way, we stay inside the classic POMDP framework and can use solvers that rely on PWLC value functions, but we are still able to model certain information-gain tasks.

The fact that extra actions have no effect on the transitions nor on the observations can be used to improve the efficiency of POMDP-IR. In fact, the result of a backup operation depends only on domain actions and not on information-rewarding actions. Satsangi et al. [109] extended our work and presented a decomposed maximization procedure that reduces the computational cost of extending the action space.

We illustrated our framework on a toy problem and included a case study on robot surveillance, demonstrating how our approach behaves in such environments. Besides the fact that they represent realistic scenarios, they are challenging as they include multiple sensors, both active and passive ones, in a multi-objective scenario.

We compared against the ρPOMDP model in a benchmark domain and showed that we compare favorably. ρPOMDPs extend the POMDP formalism to allow for belief-based rewards. In practice, this rewards are approximated by a set of tangent vectors. This ensures that the actual reward remains PWLC and existing solvers can be used with minor modifications. In a later extension to our work, Satsangi et al. [109] established the mathematical equivalence of the ρPOMDP and POMDP-IR approaches. They note that, given a POMDP model in one of the frameworks, it can be translated to the other. This conclusion show the flexibility of our approach.

Chapter 4

Value Function Approximation

In the previous chapter we have introduced a framework for active perception under a decision-theoretic framework, in particular by using POMDPs. However, POMDPs are hard to solve, and become intractable with larger problems. Several steps have been taken forward in the last few years with new solvers which can manage larger problem domains, but there is plenty of room for innovations. Thus, this chapter focus on scaling up POMDP solving by using linear value function approximation. This approximation scheme has been popular in MDPs but under-explored in POMDPs. We will review methods used in MDPs and explore theirs applicability in POMDPs.

The main contributions of this chapter are the application of linear value function approximation in POMDPs, a new approach for automatic basis function generation, and an empirical evaluation of its use in POMDP solving.

4.1 Related Work

As its name suggests, linear value function approximation fits in this family of approximations. As the state space increases, also the size of each of the vectors composing the value function increases. Mostly developed in MDPs, this approximation family tackles this problem by defining vectors over features instead of states, assuming that a small set of features is sufficient to model the system's behavior. Our work presents contributions for for this approximation field, in particular on its extension to POMDPs.

4.1.1 Linear Value Function Approximation

The idea of approximating value functions through a linear change of basis was proposed for dynamic programming [38, 110] and then extended to reinforcement learning methods [111–114]. It was also shown that linear value function approximation in reinforcement learning is equivalent to a form of linear model approximation [115].

In the context of fully observable MDP, alternatives for value iteration [40], policy iteration [116] and linear programming [117] solvers with value function approximation were proposed, which exploit factored structures to restrict the scope of basis functions. Improvements to such approaches have been later presented, which proposed alternative projection methods to compute basis function weights [42] and extensions to multiagent planning [118]. The improvement of value function approximation methods lies in two subproblems: weight optimization and basis construction. In the context of MDPs, several projection methods for weight optimization have been studied, taking into account its influence in the algorithm convergence properties [119].

Outside the fully observable framework, Guestrin et al. [120] proposed an extension to partially observable settings. However, no results are provided for their implementation and they focus on exact algorithms, such as incremental pruning.

The major line of research looks to improve POMDP solving by scaling in the state space. However, a few authors also look into large observation spaces. Mostly by building schemes to reduce its dimension, either by aggregating observations which are indistinguishable from the decision maker point of view [121], by applying data summarization techniques to obtain a reduced observation space with an arbitrarily size chosen by the user [122], by pruning some observations while minimizing the loss of value [123].

4.1.2 Basis Function Construction

The problem of basis construction is, in many cases, left for the system's designer. However, even if the designer has good knowledge of the problem's dynamics this is not necessarily an easy task. Thus, automated methods have been presented which look for sets of basis functions that represent the value function more accurately. We review existing literature on this topic for the fully and partially observable MDPs.

4.1.2.1 MDPs

Although a few simple rules arise from experience [42] there were some attempts to obtain sets of basis functions automatically and update them online. Patrascu et al.

Algorithm 6 Krylov Iteration (R,$T^{a,z}$)

```
$H \leftarrow \emptyset$
$H \leftarrow R$
for each $H_i \in H$ do
  for each $a \in A$, $o \in \Omega$ do
    if $T^{a,o}H_i$ is linearly independent of all $H_j \in H$ then
      $H \leftarrow H \cup T^{a,o}H_i$
return $H$
```

[124] greedily searches for basis functions which reduce the approximation error in approximated linear programming methods. Poupart and Boutilier [125] uses a similar procedure for incremental construction of suitable basis functions for linear approximation in weakly couple models. In turn, Parr et al. [41] derives new basis functions from the Bellman error of the previous set of basis functions.

4.1.2.2 POMDPs

In the POMDP context, given the little work on linear value function approximation, few approaches may compare to ours. The most similar is the use of Krylov iteration, which is part of a group of methods which attempt to construct basis functions by dilation of the reward function, i.e., they compute sets of bases by expanding (by multiplication) the reward function through the transition matrix operator. This group of methods, in turn, are part of a wider range of methods which compute invariant spaces of linear operators [126, 127]. Krylov iteration is particularly interesting as it is one of the few methods used in model compression in POMDPs which provided good results. In particular, it was used as a mean to obtain linear mappings which define a subspace in which compressed value functions lie [128]. In practice, this mapping defines a set of basis functions for the compressed space, thus may be used for automatic basis function construction. We present in Algorithm 6 a procedure to obtain a set H of basis functions with Krylov iteration. This algorithm starts by adding the set of reward vectors into the set of basis vectors. Then, it expands each element in the set of basis by multiplying it with the transition-observation matrix $T^{a,o}$, an $|S| \times |S|$ matrix corresponding to the probability of transition between states, given that action a is executed and observation o is received. The newly computed vector is added to the set of basis if it is linearly independent of all vectors in the set.

An alternative with early stopping conditions was later proposed [129], dubbed truncated Krylov iteration (Algorithm 7). With this formulation, Krylov iteration is ran, retaining only the first k vectors. By retaining basis vectors which are far from being linear

Algorithm 7 Truncated Krylov Iteration (R,$T^{a,z}$,k,ϵ)

```
$H \leftarrow \emptyset$
$basisSet \leftarrow R$
while $basisSet \neq \emptyset$ and $n_h < k$ do
    remove vector $x$ from $basisSet$ with largest error $\|x - \sum_i c_i H_i\|_2$
    $H \leftarrow H \cup x$
    for each $a \in A$, $o \in \Omega$ do
        $basisSet \leftarrow basisSet \cup T^{a,z} x$
return $H$
```

combination of prior vectors, the algorithm tries to increase the space spanned by basis vectors as most as possible.

Given its iterative nature, both Krylov iteration versions can be computational demanding. We will propose an alternative approach to basis function construction which is less demanding, as is not an iterative procedure.

4.2 Linear Value Function Approximation Extension to POMDPs

Despite various works on fully observable MDPs, there was little attention to extending this framework into the partially observable setting. In particular, only Guestrin et al. [120] proposed to adapt this idea to POMDPs but did not report any experimental results.

Recall that a value function in an infinite-horizon POMDP can be approximated arbitrarily well by a finite set of α-vectors. The key idea on linear value function approximation in POMDPs is that we can approximate each vector as a linear combination of basis functions. Thus, α-vectors are approximated by $\widetilde{\alpha}$-vectors, written as a linear combination of basis functions h_i:

$$\widetilde{\alpha}(s) = \sum_{i=1}^{n_h} \omega_{\alpha,i} h_i(s). \tag{4.1}$$

For some intuition, we present in Figure 4.1 an illustrative example of representing an $\widetilde{\alpha}$-vector in a two-state model with a set of two basis functions, $H = \{\boldsymbol{h}_1, \boldsymbol{h}_2\}$. Consider the state space of the problem as $S = (s_1, s_2)$, then each basis function is represented as a vector $\boldsymbol{h}_i = [h_i(s_1)\ h_i(s_2)]$. In this case basis functions are individual indicators for each state:

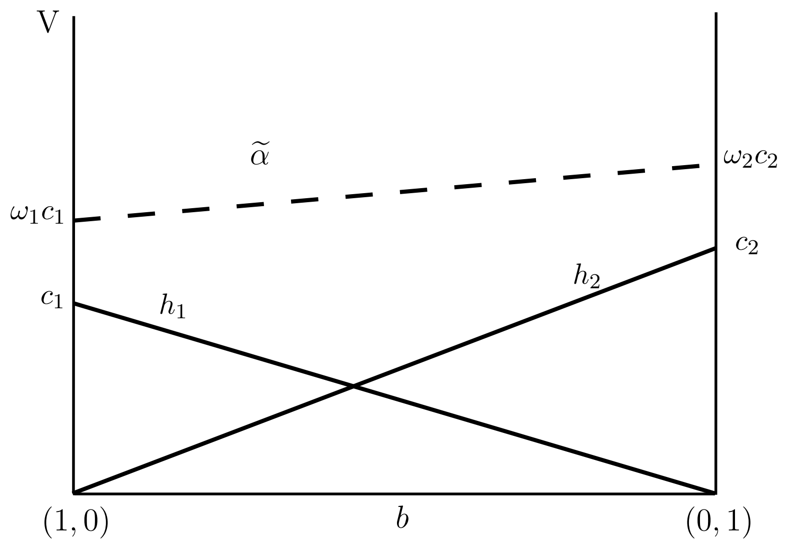


FIGURE 4.1: Example of a vector representation with basis functions with two states.

$$\begin{cases} \boldsymbol{h}_1 = [c_1 \; 0]; \\ \boldsymbol{h}_2 = [0 \; c_2]. \end{cases} \tag{4.2}$$

Then, by weighting on each basis function the final vector is obtained:

$$\widetilde{\boldsymbol{\alpha}} = \omega_1 \mathbf{h}_1 + \omega_2 \mathbf{h}_2 \tag{4.3}$$

$$= \omega_1 \begin{bmatrix} c_1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ c_2 \end{bmatrix} \tag{4.4}$$

$$= \begin{bmatrix} \omega_1 c_1 \\ \omega_2 c_2 \end{bmatrix} \tag{4.5}$$

Each vector values can be set by controlling the weights of each basis function. Note that in this simple example the set of bases fully spans the state space, thus we call it a complete set of bases, and controlling the weights is equivalent to controlling the full vector. Typically this will not be the case and the set of weights which best represent a full vector must be obtained.

As introduced in Section 2.3.3 factored models allow for more efficient computation and basis function representation because, in the general case, basis function have smaller scopes. In this representation $\widetilde{\alpha}$-vectors with factored states are written as a linear combination of basis functions restricted to to a subset of state variables $C_i \subseteq X$:

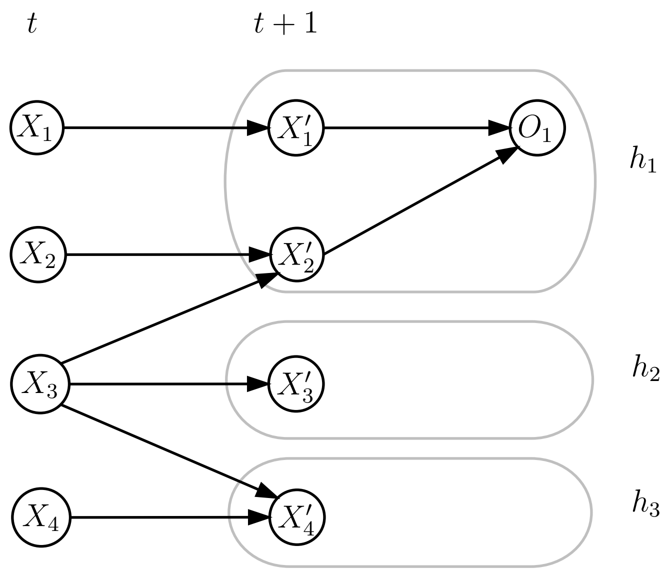


FIGURE 4.2: Example DBN with basis.

$$\widetilde{\alpha}(\mathbf{x}) = \sum_{i=1}^{n_h} \omega_{\alpha,i} h_i(\mathbf{c}_i). \tag{4.6}$$

For illustration, Figure 4.2 shows an example of a choice of basis function for a factored model, with its respective DBN. In this example each basis function has a different scope, in particular:

- $C_1 = \Gamma(h_1) = \{X_1, X_2\}$,
- $C_2 = \Gamma(h_2) = \{X_3\}$,
- $C_3 = \Gamma(h_3) = \{X_4\}$.

The most important information to retrieve from basis functions is their scope and relation between their values, rather than the specific mapping from bases to state values. For instance, in our two-state example we can choose any parameters c for the basis functions that the weights would balance to accurately represent any vector. It is more important to know how it ranks with the other values in this vector, i.e., what is the relative importance of each state at each basis function.

Finally, this is a general formulation such that any classic algorithm to solve POMDPs can be adapted to work on the subspace $\mathcal{H}$. At each value iteration step we can compute all new α-vectors which make up the new value function V^{n+1} and then project those onto $\mathcal{H}$. Keeping that in mind, we will focus on the use of linear value function approximation in a particular class of solvers, point-based algorithms.

4.3 Point-Based Methods With Linear Value Function Approximation

Recall that the original formulation for point-based POMDP methods [62, 67, 71] states that we can update the maximizing vector at each belief point b with the operator $\texttt{backup}(b)$ (Section 2.2.5):

$$\texttt{backup}(b) = \operatorname*{argmax}_{\{g_a^b\}_{a\in A}} b \cdot g_a^b, \quad \text{with} \tag{4.7}$$

$$g_a^b = R(s,a) + \gamma \sum_o \operatorname*{argmax}_{\{g_{ao}^k\}} b \cdot g_{ao}^k, \text{ and} \tag{4.8}$$

$$g_{ao}^k(s) = \sum_{s'} p(o|s',a)p(s'|s,a)\alpha_n^k(s'). \tag{4.9}$$

In order to accommodate linear value function approximation two key some steps must be rewritten. We start with the g_{ao} computation, obtained by replacing (4.6) in (4.9):

$$\begin{aligned} \widetilde{g}_{ao}^k(\mathbf{x}) &= \sum_{\mathbf{x}'} p(o|\mathbf{x}',a)p(\mathbf{x}'|\mathbf{x},a)\widetilde{\alpha}(\mathbf{x}) && (4.10)\\ &= \sum_{\mathbf{x}'} p(o|\mathbf{x}',a)p(\mathbf{x}'|\mathbf{x},a)\sum_{i=1}^{n_h}\omega_{k,i}h_i(\mathbf{c}_i') && (4.11)\\ &= \sum_{i=1}^{n_h}\omega_{k,i}\sum_{\mathbf{x}'} p(o|\mathbf{x}',a)p(\mathbf{x}'|\mathbf{x},a)h_i(\mathbf{c}_i'). && (4.12) \end{aligned}$$

Note that the newly computed $\widetilde{g}_{ao}$ vectors might not be in the space spanned by the basis functions. Thus, if at this point we project these vectors back into the subspace $\mathcal{H}$, we end up with a vector of weights $\boldsymbol{\omega}_{ao}^k$ which represent the linear approximation of $\widetilde{g}_{ao}$ vector:

$$\boldsymbol{\omega}_{ao}^k = \Pi\widetilde{g}_{ao}^k \tag{4.13}$$

Given this, we can perform the rest of the backup operator in the reduced subspace. First, we can reformulate (4.8) to accommodate an approximated formulation:

Algorithm 8 Point-Based Backup With Linear Value Function Approximation

```
for all $a \in A$ do {independent of $b$, can be cached}
  $\boldsymbol{\omega}_R \leftarrow$ Project $R(s,a)$ vectors to $\mathcal{H}$.
  for all $o \in O$ do
    Compute $g^k_{ao}$ vectors using (4.12).
    $\boldsymbol{\omega}^k_{ao} \leftarrow$ Project $g^k_{ao}$ vectors to $\mathcal{H}$.
for all $a \in A$ do
  Compute $\boldsymbol{\omega}^b_a$ vectors using (4.19).
Find the maximizing vector backup($b$) using (4.17).
```

$$\widetilde{g}^b_a = \widetilde{R} + \gamma \sum_o \operatorname*{argmax}_{\widetilde{g}^k_{ao}} b \cdot \widetilde{g}^k_{ao} \tag{4.14}$$

$$= \widetilde{R} + \gamma \sum_o \operatorname*{argmax}_{\widetilde{g}^k_{ao}} b \cdot \sum_{i=1}^{n_h} \omega^k_{ao,i} \mathbf{h_i} \tag{4.15}$$

At this point, we can also reformulate two key operations to use approximate representations: the inner product between vectors and belief points:

$$b \cdot \widetilde{\alpha} = \sum_{\mathbf{x}\in\mathbf{X}} \sum_{i=1}^{n_h} \omega_{\alpha,i} h_i(\mathbf{x}) b(\mathbf{x}) \tag{4.16}$$

$$= \sum_{i=1}^{n_h} \omega_{\alpha,i} \sum_{\mathbf{c}_i \in \mathbf{C}_i} h_i(\mathbf{c}_i) b_i(\mathbf{c}_i). \tag{4.17}$$

and the sum of two vectors represented by basis functions:

$$\sum_{i=1}^{n_h} \omega_{1,i} h_i(\mathbf{x}) + \sum_{j=1}^{n_h} \omega_{2,j} h_j(\mathbf{x}) = \sum_{i=1}^{n_h} (\omega_{1,i} + \omega_{2,i}) h_i(\mathbf{x}). \tag{4.18}$$

Therefore, computing the g^b_a vectors (4.8) can be performed using solely a reduced representation with weight vectors:

$$\boldsymbol{\omega}^b_a = \boldsymbol{\omega}_R + \gamma \sum_o \operatorname*{argmax}_{\boldsymbol{\omega}^k_{ao}} \left(\sum_{i=1}^{n_h} \omega^k_{ao,i} \sum_{\mathbf{c}_i \in \mathbf{C}_i} h_i(\mathbf{c}_i) b_i(\mathbf{c}_i) \right). \tag{4.19}$$

Here, $\boldsymbol{\omega}_R$ and $\boldsymbol{\omega}^k_{ao}$ represent the projections onto the space spanned by the basis functions of, respectively, the immediate reward $R(s,a)$ and g^k_{ao} vectors which result from (4.12). The inner product is performed using an approximated representation as in (4.17), and summation of weight vectors as in (4.18).

Algorithm 9 Point-Based Backup With Linear Value Function Approximation, Optimized For ADDs

```
$a^* \leftarrow \emptyset, v^* \leftarrow -\infty$
**for each** $o \in \Omega$ **do** $\widetilde{\alpha}^*_o \leftarrow nil$
**for all** $a \in A$ **do**
   $Q_a(b) \leftarrow 0$
   **for all** $o \in \Omega$ **do**
      $b' \leftarrow \tau(b, a, o)$
      $\widetilde{\alpha}_o \leftarrow \text{argmax}_{\widetilde{\alpha} \in V} \widetilde{\alpha} \cdot b'$
      $Q_a(b) \leftarrow Q_a(b) + \widetilde{\alpha}_o \cdot b'$
   **if** $Q_a(b) > v^*$ **then**
      $a^* \leftarrow a, v^* \leftarrow Q_a(b)$
      **for each** $o \in \Omega$ **do** $\widetilde{\alpha}^*_o \leftarrow \widetilde{\alpha}_o$
**return** $\Pi(r_{a^*} + \gamma \sum_o g^{\widetilde{\alpha}^*_o}_{a,o})$
```

Algorithm 8 summarizes the steps needed to perform the backup step for each belief point using a general point-based method with linear value function approximation. Note that in this formulation the projection operator is called $|A||O|k$ times. Depending on the size of the problem and on the complexity of this operator this might be computational expensive. The efficiency of the backup operator also depends on the data representation used for the factored vectors.

In respect to data representation, an ADD-based implementation (Section 2.3.2) is a natural choice for factored basis functions and linear value functions. It is useful to exploit model structure in factored linear value functions given that basis functions will often be restricted to a subset of the whole state space. Thus, ADDs naturally capture and take advantage of the factored nature on most of the operations with factored linear value functions. Moreover, as noted in Section 2.3.2, the backup operator can be more efficiently computed. In Algorithm 5 we present the optimized backup operator from Algorithm 5 slightly modified to deal to factored linear value functions. Here, the inner products can be efficiently performed according to (4.17). In this formulation, the only vector which is not in the space spanned by basis functions is the final result of the backup. Therefore, there is only one projection operator per backup, at the end of the process.

Besides being based on a more efficient formulation, both on its process and on data representation, Algorithm 9 has the particular advantage of reducing the number of projections needed to compute at each backup. This can result in large computational gains, given the complexity of projections. We will further study the usage of projection operators in POMDPs in Section 4.4.

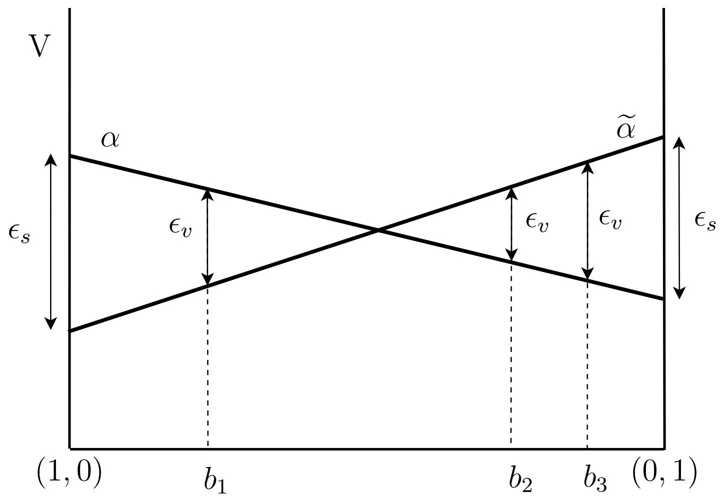


FIGURE 4.3: Projection Errors example. α is the original vector and $\widetilde{\alpha}$ the approximated one. ϵ_s is the error minimized in state-based projection, while ϵ_v is the error minimized in belief-based projections.

4.4 Belief-based Projection Methods

In Section 2.1.3 the projection operator Π is introduced in the context of MDPs. This operator finds the best weighted combination of basis functions, which induces the closest value function in $\mathcal{V}$ according to a pre-defined mapping.

In terms of its representation, a vector in a POMDP value function is equivalent to a value function in a MDP, i.e., both are represented in memory as vectors over the state space, with size $|S|$. Therefore, projection methods used for value functions in MDPs can be translated to projecting vectors in POMDPs. In the remaining we will name those as state-based projections, as a reference to the fact that they minimize the projection error of vectors over states.

State-based projections offers a practical way to project vectors back to the space spanned by the set of basis functions. However, note that value functions in POMDPs are defined over the belief space. Therefore, projections may be performed in an alternative way, which in turn minimizes the induced error in the value function. We name it belief-based projections.

Definition 4.1. A belief-based projection operator is a mapping $\Pi : \Re^{|S|} \to \mathcal{H}$. Π is said to be a projection operator w.r.t. a norm $\|\cdot\|$ if: $\widetilde{\boldsymbol{\alpha}} = \Pi\boldsymbol{\alpha} = H\mathbf{w}^*$ such that $\mathbf{w}^* \in \operatorname{argmin}_{\mathbf{w}} \left\|V^{\widetilde{\boldsymbol{\alpha}}} - V^{\boldsymbol{\alpha}}\right\|$. For any vector $\boldsymbol{\beta}$, $V^{\boldsymbol{\beta}}$ represents the value of following the policy associated with $\boldsymbol{\beta}$ for each belief in the belief set.

This projection method is best suited to POMDP solvers which work in a subset of belief points such as point-based methods. Figure 4.3 shows a graphical explanation for both projection methods and the respective errors they try to minimize. In the case in which the original vector can be approximated with no error, i.e., it is inside the subspace $\mathcal{H}$, then the projection result is the same for both methods. The complexity of a state-based projection is dependent on the size of the state space. In turn, the complexity of belief-based projections is dependent on the size of the belief space. Note that both projections are related:

$$\|\epsilon_v\| = \left\|V^{\widetilde{\boldsymbol{\alpha}}} - V^{\boldsymbol{\alpha}}\right\| = \tag{4.20}$$

$$\|B\widetilde{\boldsymbol{\alpha}} - B\boldsymbol{\alpha}\| = \tag{4.21}$$

$$\|B\left(\widetilde{\boldsymbol{\alpha}} - \boldsymbol{\alpha}\right)\| = \tag{4.22}$$

$$\|B\epsilon_s\| \tag{4.23}$$

where B is a matrix with all belief points. Each row of the matrix represents one belief point.

In the particular case that the belief set is composed of all the extreme points in the belief simplex, then both projections are equivalent. The projection error with belief-based projections is bounded by the error of state-directed projection: from (4.20) it follows that $\|B\left(\widetilde{\boldsymbol{\alpha}} - \boldsymbol{\alpha}\right)\|_p \leq \|B\|_p \|\widetilde{\boldsymbol{\alpha}} - \boldsymbol{\alpha}\|_p$. When $p = \infty$, then $\|B\|_\infty = 1$. The most used norms in the context of state-based projections are the $\mathcal{L}_2$ and $\mathcal{L}_\infty$ norms. We derive similar formulations for both norms, in the context of belief-based projections.

L-2 norm Following the least squares solution for parameter estimation previously presented (2.16), it directly follows for belief-based projections:

$$\boldsymbol{\omega} = \underset{\boldsymbol{\omega}}{\operatorname{argmin}} \|B\widetilde{\alpha} - B\alpha\|_2 \tag{4.24}$$

$$= \underset{\boldsymbol{\omega}}{\operatorname{argmin}} \|BH\boldsymbol{\omega} - B\alpha\|_2 \tag{4.25}$$

$$= ((BH)^T(BH))^{-1}(BH)^T B\alpha \tag{4.26}$$

L-∞ norm Similarly, we can apply the same reasoning of (2.20) to the belief-based equation, $\boldsymbol{\omega} = \operatorname{argmin}_{\boldsymbol{\omega}} \|B\widetilde{\alpha} - B\alpha\|_\infty$, and solve it with the following linear program:

$$
\begin{array}{ll}
\text{variables} & \omega_1, \dots, \omega_n, \phi \\
\text{minimize} & \phi \\
\text{subject to} & \phi \geq \sum_{i=1}^{n_h} \omega_i \mathbf{b}_k \cdot \boldsymbol{h}_i - \mathbf{b}_k \cdot \boldsymbol{\alpha}, \\
& \phi \geq \mathbf{b}_k \cdot \boldsymbol{\alpha} - \sum_{i=1}^{n_h} \omega_i \mathbf{b}_k \cdot \boldsymbol{h}_i, \; k = 1, \dots, |B|
\end{array}
\tag{4.27}
$$

The linear program for the $\mathcal{L}_\infty$ norm has $n_h + 1$ variables but $2|B|$ constraints, which offers an alternative for state-based projections which has $2|S|$ constraints. Comparing to state-based projections this formulation has a few more operations (inner products), whose computational demand can be alleviated when using ADD to represent data. Nonetheless, in some problems, in which we plan for a subset of beliefs, it may happen that it is possible to achieve good policy quality with a small belief set. In these cases the gains obtained in the optimization due to fewer constraints can be larger than the loss incurred.

4.5 Automatic Basis Function Construction

When using linear value function approximation the choice of basis functions is crucial to the success of such approximation schemes. In this section we will study the dynamics of value iteration in POMDPs and from its structure derive a good set of basis functions.

In the MDP framework some approaches were presented to dynamically build a good set of basis functions, either previous to problem solving, or by updating over each iteration. Although POMDP methods may be inspired by those, their direct application is not straightforward, given the increased complexity in POMDP value functions. In this section we introduce an approach for basis function generation by exploiting model structure.

We propose to take advantage of factored models to automatically search for a good set of basis functions, i.e., a set of bases which takes the approximated value function, lying in the subspace $\mathcal{H}$, as close as possible to the original value function.

First, we combine equations (4.8) and (4.9) and rewrite the result in a matrix form:

$$\mathbf{g}_a^b = \mathbf{r}_a + \gamma \sum_o \underset{\{\mathbf{g}_{ao}^k\}}{\operatorname{argmax}} b \cdot \mathbf{g}_{ao}^k \tag{4.28}$$

$$\mathbf{g}_{ao}^k = \mathbf{P}^{ao} \boldsymbol{\alpha}_n^k \tag{4.29}$$

where $\mathbf{P}^{ao}$ is a transition-observation $|S| \times |S|$ matrix, with each element $P^{ao}(s', s) = p(o|s', a)p(s'|s, a)$.

Replacing α-vectors in (4.29) by their approximated version we obtain:

$$\begin{aligned} \mathbf{g}_{ao}^k &= \mathbf{P}^{ao} \tilde{\boldsymbol{\alpha}}_n^k && (4.30) \\ &= \mathbf{P}^{ao} \sum_{i=1}^{n_h} w_i^k \mathbf{h}_i && (4.31) \\ &= \sum_{i=1}^{n_h} w_i^k \mathbf{P}^{ao} \mathbf{h}_i && (4.32) \end{aligned}$$

We note here that g_a^b vectors are formed by a sum of two groups of vectors: the reward vector and the sum of a set of g_{ao} vectors, which are backprojected copies of α_n^k for each a and o. Intuitively, this gives a hint on how to choose a good set of basis functions. First, the reward should be representable by the set of bases. Second, note that each g_{ao} vector is the result of a linear transformation, therefore, when using an approximate representation, we want the result of this transformation to be as close as possible to the space spanned by the set of basis vectors. In this case we can say that the set of basis vectors should span the space in the direction of the eigenvectors of each matrix P^{ao}, for each a and o. Note, however, that the space spanned by the eigenvectors of transition-observation matrix is a subspace of that spanned by the eigenvectors of the transition matrix, as those have the same dimension. Given that the transition matrix only depends on the action and not in the observation, this reduces the computations needed at this step:

$$\mathbf{g}_{ao}^k = \sum_{i=1}^{n_h} w_i^k \mathbf{T}^a \boldsymbol{\Omega}^{ao} \mathbf{h}_i \tag{4.33}$$

where $\mathbf{T}^a$ is the $|S| \times |S|$ transition matrix, and $\boldsymbol{\Omega}^{ao}$ is a diagonal matrix whose diagonal is the observation function.

Using a full matrix representation brings no clear advantage unless we find the transition matrix to be rank deficient. However, we may take advantage of transition representations to more efficiently represent the set of basis functions.

First, we rewrite the g_{ao} vectors by replacing (4.6) in (4.9):

$$\begin{aligned} g^k_{ao}(\mathbf{x}) &= \sum_{\mathbf{x}'} p(o|\mathbf{x}',a)p(\mathbf{x}'|\mathbf{x},a)\alpha^k_n(\mathbf{x}') && (4.34)\\ &= \sum_{i=1}^{n_h} \omega_{k,i} \sum_{\mathbf{x}'} p(o|\mathbf{x}',a)p(\mathbf{x}'|\mathbf{x},a)h_i(\mathbf{x}') && (4.35)\\ &= \sum_{i=1}^{n_h} \omega_{k,i} g^i_{ao}(\mathbf{x}) && (4.36) \end{aligned}$$

Given that we are dealing with factored models, we may rewrite part of the equation taking into account independence between features of the model. First, remember that the scope of each basis function can be restricted to a subset C_i, thus we rewrite $h_i(\mathbf{x}') = h_i(\mathbf{c}'_i)$. Also, we can replace both observation and transition functions by their factored representation. Note that we can split dependencies in two cases: state factors which are in the observations scope, and those which are not. Let us name those sets X_o and $X_{\bar{o}}$ (mathematically, $X_o \subseteq \Gamma(O)$ and $X_{\bar{o}} \nsubseteq \Gamma(O)$), respectively, and exploit independence between state transitions to separate summations, noting that we can rewrite a sum over the state space as a series of sums over different state components.

$$\begin{aligned} g^i_{ao}(\mathbf{x}) &= \sum_{\mathbf{x}'_{\bar{o}}} \sum_{\mathbf{x}'_o} p(\mathbf{x}'_{\bar{o}}|\Gamma(\mathbf{x}'_{\bar{o}}),a)p(o|\mathbf{x}'_o,a)p(\mathbf{x}'_o|\Gamma(\mathbf{x}'_o),a)h_i(\mathbf{c}'_i) && (4.37)\\ &= \sum_{\mathbf{x}'_1} \cdots \sum_{\mathbf{x}'_l} \sum_{\mathbf{x}'_o} p(\mathbf{x}'_1|\Gamma(\mathbf{x}'_1),a) \times \ldots \times p(\mathbf{x}'_l|\Gamma(\mathbf{x}'_l),a) \times \\ &\quad \times p(o|\mathbf{x}'_o,a)p(\mathbf{x}'_o|\Gamma(\mathbf{x}'_o),a) \times h_i(\mathbf{c}'_i) && (4.38) \end{aligned}$$

The backprojection of a basis function can be split into two sub-projections. Given the context, it would be interesting if we could exploit basis functions' scope according to independence in transitions. In the following, we depict two cases, one where basis functions have the same scope as observations ($C'_i = X'_o$), and other where its scope is not in that set ($C'_i = X'_{1 \leq i \leq l}$). Here we consider that X_o can be any set of state variables and each $X_l \subset X_o$ is an individual variable. By replacing the scope of basis functions and rearranging the equations for each case it is possible to rewrite them. In both cases, the transition for each state component is under the summation over that component, with the observation function under the summation over its scope. In the first case basis functions are under the summation over the state components on the observations scope, while in the second case they are under the summation over each state variable not on that scope.

$$g^i_{ao}(\mathbf{x}) = \begin{cases} \sum_{\mathbf{x}'_1} \dots \sum_{\mathbf{x}'_l} p(\mathbf{x}'_1|\Gamma(\mathbf{x}'_1), a) \dots p(\mathbf{x}'_l|\Gamma(\mathbf{x}'_l), a) \times \\ \times \sum_{\mathbf{x}'_o} p(o|\mathbf{x}'_o, a) p(\mathbf{x}'_o|\Gamma(\mathbf{x}'_o), a) h_i(\mathbf{x}'_o) & \text{if } C'_i = X'_o \\ \sum_{\mathbf{x}'_i} p(\mathbf{x}'_i|\Gamma(\mathbf{x}'_i), a) h_i(\mathbf{x}'_i) \\ \times \sum_{\mathbf{x}'_o} p(o|\mathbf{x}'_o, a) p(\mathbf{x}'_o|\Gamma(\mathbf{x}'_o), a) & \text{if } C'_i = X'_{1<i<l} \end{cases} \tag{4.39}$$

Taking into account that $\sum_{\mathbf{x}'_o} p(o|\mathbf{x}'_o, a)p(\mathbf{x}'_o|\Gamma(\mathbf{x}'_o), a) = K$ is constant and independent of i, and, for any j, $\sum_{\mathbf{x}'_j} p(\mathbf{x}'_j|\Gamma(\mathbf{x}'_j), a) = 1$, then (4.39) can be simplified:

$$g^i_{ao}(\mathbf{x}) = \begin{cases} \sum_{\mathbf{x}'_o} p(o|\mathbf{x}'_o, a) p(\mathbf{x}'_o|\Gamma(\mathbf{x}'_o), a) h_i(\mathbf{x}'_o) & \text{if } C'_i = X'_o \\ K \sum_{\mathbf{x}'_i} p(\mathbf{x}'_i|\Gamma(\mathbf{x}'_i), a) h_i(\mathbf{x}'_i) & \text{if } C'_i = X'_{1<i<l} \end{cases} \tag{4.40}$$

Again, at this step we can switch to a matrix notation and follow the same reasoning as with full matrices.

$$\mathbf{g}^i_{ao} = \begin{cases} \mathbf{T}^a_{\mathbf{x}'_o} \mathbf{\Omega}^{ao}_{\mathbf{x}'_o} \mathbf{h}_i & \text{if } C'_i = X'_o \\ K \mathbf{T}^a_{\mathbf{x}'_i} \mathbf{h}_i & \text{if } C'_i = X'_{1<i<l} \end{cases} \tag{4.41}$$

where $T_{\mathbf{x}'_j} : p(\mathbf{x}'_j|\Gamma(\mathbf{x}'_j), a)$ and $\Omega_{\mathbf{x}'_j} : p(o|\mathbf{x}'_j, a)$. Both cases represent a linear transformation from $\mathbb{R}^{|C'_i|}$ to $\mathbb{R}^{|\Gamma(C'_i)|}$.

As stated before, to minimize the projection error every g^i_{ao} vector should be as close as possible to the subspace spanned by the basis functions. That is to say that each vector resulting from the linear transformation (4.41) should point in the same direction of the respective basis vector $\mathbf{h}_i$. Given that often these factored transition matrices might be rectangular we choose to use the right singular vectors of $\mathbf{T}^a$ to build the set of bases for the transformation.

We use the right singular vectors, given that for any $m \times n$ transformation matrix, the right singular vectors form an orthogonal basis for $\mathbb{R}^n$, which is the subspace we are interested to span in each transformation.

Algorithm 10 Automatic Construction of Basis Functions

```
$H \leftarrow \emptyset$
for all $a \in A$ do
  $X'_o \leftarrow \Gamma(O_a)$
  $USV^T \leftarrow svd(T_{\mathbf{x}'_o})$
  $H \leftarrow H \cup V$
  for all $X'_j \notin X'_o$ do
    $USV^T \leftarrow svd(T_{\mathbf{x}'_j})$
    $H \leftarrow H \cup V$
  $H \leftarrow H \cup R_a$
return linearly independent columns of $H$
```

Our methodology is summarized in Algorithm 10, which returns the set of basis functions H. This procedure is dependent on the structure of the model, in particular on the dependencies of observation factors. However, in problems with structure which allows to explore independencies it is possible to find a compact space to represent the value function, as we will see in the following.

4.6 Exploiting Factored Observation Spaces

In most of the literature on improving POMDP solvers, the focus is on scaling the state space. Although this is a very important issue in scalability and application in real systems we can argue that the size of the observation space is also becoming the limit to scalability. State and observation spaces influence scalability in different ways. Larger state spaces increase the complexity in computations and the memory needs during the solving process. Backups in value iteration are composed of a sequence of mathematical operations, involving mostly inner products and sums of vectors. Each vector is $|S|$-dimensional, thus the complexity of those operators and the size of each vector increases directly with the state space. On the other hand, the observation space influences the number of new vectors computed at each iteration, as well as the scope of each g_{ao} vector. Although the increased complexity with the state space has been tackled in different ways, there has been little work considering the complexity of observation spaces, in particular by exploiting factored observation variables. We will exploit the use of factored linear value functions to take advantage of independencies between variables.

Let us now consider that we have an observation space represented in a factored way. Also, assume that we are able to write the factored state space S as the cross product of subsets of variables U_i. Each subset U_i contains the parents of observation i in the DBN (that is, $\mathbf{u}'_i \in \Gamma(O_i)$). We will also assume that all subsets are disjoint ($\forall\ i, j\ :\ U'_i \cap U'_j = \emptyset$). The key idea is that, since basis functions are defined over a subset of the state space, we may exploit similarities between the scope of basis functions and the structure of the DBN, although imposing some restrictions on the choice of basis functions.

Looking at Equation 4.12 we can replace both observation and transition functions by their factored, thus independent, representation. Additionally, we will assume that for each problem we define n_O basis functions, each one with domain U_i, that is, the domain of the basis functions is the same as of the observation variables. From this, we get the following:

$$\begin{aligned} g_{ao}^k(\mathbf{x}) = \sum_{i=1}^{n_O} \omega_{k,i} \sum_{\mathbf{u}'_1 \in \Gamma(O_1)} \cdots \sum_{\mathbf{u}'_{n_O} \in \Gamma(O_{n_O})} & \\ & p(o_1|\mathbf{u}'_1, a) \times \ldots \times p(o_{n_O}|\mathbf{u}'_{n_O}, a) \times \\ & \times p(\mathbf{u}'_1|\Gamma(U'_1), a) \times \ldots \times p(\mathbf{u}'_{n_O}|\Gamma(U'_{n_O}), a) \\ & \times h_i(\mathbf{u}'_i). \end{aligned} \tag{4.42}$$

Due to the independence between variables we can rewrite (4.42) as follows:

$$\begin{aligned} g_{ao}^k(\mathbf{x}) = \sum_{i=1}^{n_O} \omega_{k,i} & \left(\sum_{\mathbf{u}'_1 \in \Gamma(O_1)} p(o_1|\mathbf{u}'_1, a) p(\mathbf{u}'_1|\Gamma(U'_1), a) \right) \times \\ \ldots \times & \left(\sum_{\mathbf{u}'_{n_O} \in \Gamma(O_{n_O})} p(o_{n_O}|\mathbf{u}'_{n_O}, a) p(\mathbf{u}'_{n_O}|\Gamma(U'_{n_O}), a) \right) \\ & \times h_i(\mathbf{u}'_i). \end{aligned} \tag{4.43}$$

In this representation, at iteration i all terms are constant except for term i, which includes the corresponding value of the basis function. Therefore, we can consider a set of auxiliary vectors defined as:

$$d_{ao_i}(\mathbf{x}) = \sum_{\mathbf{u}'_i \in \Gamma(O_i)} p(o_i|\mathbf{u}'_i, a) p(\mathbf{u}'_i|\Gamma(U'_i), a), \tag{4.44}$$

$$f_{ao_i}(\mathbf{x}) = \sum_{\mathbf{u}'_i \in \Gamma(O_i)} p(o_i|\mathbf{u}'_i, a) p(\mathbf{u}'_i|\Gamma(U'_i), a) h_i(\mathbf{u}'_i), \tag{4.45}$$

and replace them in (4.43):

$$g_{ao}^k(\mathbf{x}) = \sum_{i=1}^{n_O} \omega_{k,i} \prod_{\substack{j=1\\ j\neq i}}^{n_O} f_{ao_i}(\mathbf{x}) d_{ao_j}(\mathbf{x}) \tag{4.46}$$

$$= \sum_{i=1}^{n_O} \omega_{k,i} f_{ao_i}(\mathbf{x}) \prod_{\substack{j=1\\ j\neq i}}^{n_O} d_{ao_j}(\mathbf{x}). \tag{4.47}$$

Given that we can precompute these auxiliary vectors prior to performing value iteration it is possible to reduce the number of numerical operations needed during computation of g_{ao}^k vectors, thereby reducing the complexity of this step. In the original formulation ((2.29)) it takes $|S|^2$ operations to compute new vectors while in this formulation it takes $|S| \times n_O \times (n_O - 1)$ operations. Therefore, if the number of basis functions n_O is small in comparison to the size of the state space

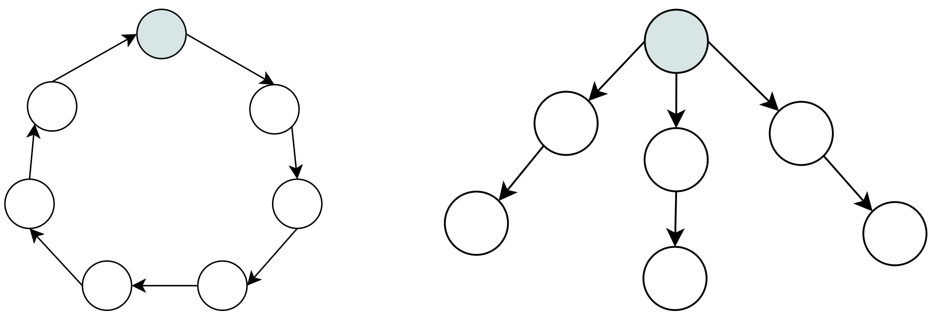
This technique can help in practice to alleviate some of the computational cost involved in g_{ao} computation, while imposing some assumptions on the model. Although may be seen as too limitative, assumptions on the models are not rare. For instance, similar to ours but in the context of mixed-observable MDPs, Drougard et al. [130] also assume that an observation variable is available for each hidden state variable. Moreover, they note that if that is not the case of the original model it is possible to reconstruct the model in a way that original observations can be equivalently modeled as a Cartesian product of observations, each one depending on one state variable.

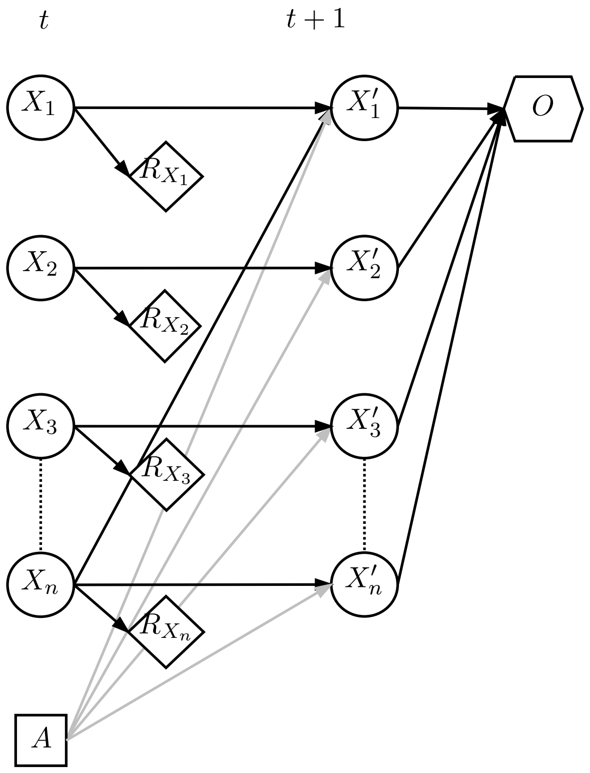
4.7 Experiments

To test the viability of the contributions in this chapter, we performed a series of illustrative experiments. We start by a description of the models used in our benchmark problems and provide empirical evaluations of our methods using linear value function approximation.

4.7.1 Model Description

We test our method on the network management problem [2] and a variation of the fire fighting problem [131]. We will briefly describe the models used for each problem.


(A) cycle (B) 3legs

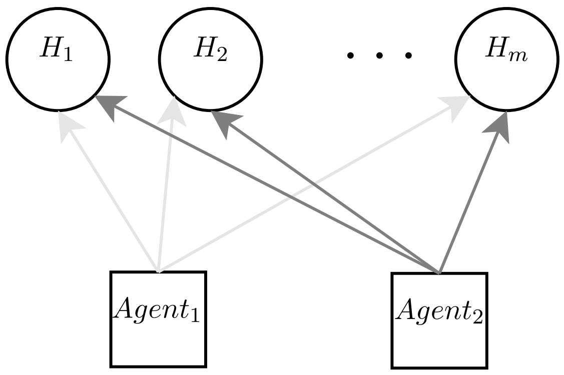


(C) Dynamic Bayesian network.

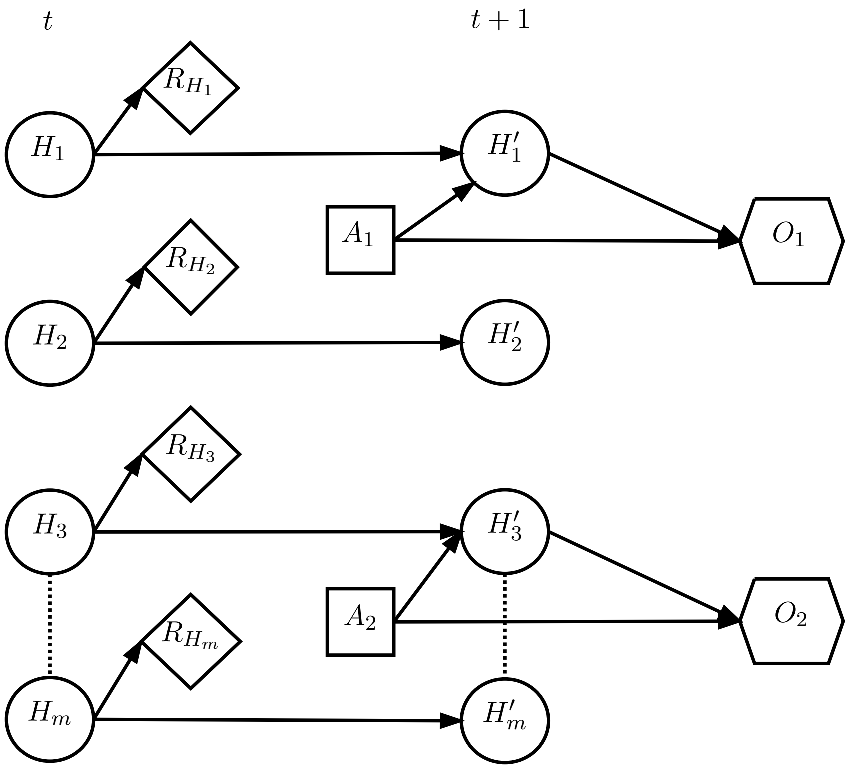
FIGURE 4.4: Network management problem. (a)(b) Illustration of the problem, showing different configurations. Shaded node represents the server machine and arrows represent connections in the network.(c) Dynamic Bayesian network representation of the problem. State factors X_i represent the state of each machine i.

4.7.1.1 Network Management

In the network management problem a system administrator must maintain a network of machines. At any stage each machine can be in a state *up* or *down* and the available options for the system administrator are to *reboot*, *ping* any machine or *do nothing*. An observation is received every time a machine is pinged: the agent observes the correct state of the pinged machine with probability 0.95.. The agent receives a positive reward for each working machine (1 for all machines and 2 for a particular machine assigned as server), a small 0.1 cost for pinging machines, and a higher 2.5 cost for rebooting.



(A) Problem illustration.



(B) Dynamic Bayesian network.

FIGURE 4.5: Fire fighting problem.(a) Illustration of the problem. Squares represent agents, while circles represent houses. Each agent can act in one single house per timestep, but it can move freely between houses. (b) Dynamic Bayesian network representation of the problem. State factors H_i represent the fire level at each house i. For visualization purposes we show the dynamics of the problem for a single action $A = (1, 3)$, meaning that Agent 1 goes to house 1 and agent 2 goes to house 3.

When a machine is down it stays down with high probability (0.95). If it is *up*, then it transitions to *down* with probability 0.333 if the parent is also *down*, or 0.1 if parent is *up*. We tested on this problem with two configurations: *3legs* and *cycle* (Figure 4.4)

Network, n machines				Fire fighting, m houses			
n	$\|S\|$	$\|A\|$	$\|O\|$	m	$\|S\|$	$\|A\|$	$\|O\|$
4	16	9	2	3	27	6	4
7	128	15	2	4	81	10	4
10	1024	21	2	5	243	15	4
13	8192	27	2	6	729	21	4
16	65536	33	2	7	2187	28	4

TABLE 4.1: Problem sizes. $|S|$, $|A|$, $|O|$, represent, respectively, the number of states, actions and observations for each problem.

4.7.1.2 Fire Fighting

The fire fighting problem models a team of agents that have to extinguish fires in a number of houses, controlled by a centralized decision maker. We consider a variation of the problem with a fixed number of 2 agents, and increasing number of m houses. Each house can be in one of 3 fire levels (fl_0, fl_1, fl_2). Each agent can go to any of the houses and observes whether there are flames or not at its actual location. Flames are observed with probability 0.275, 0.5 and 0.775 if the fire at that location is, respectively, in level 0, 1 or 2. If any fire fighter is present at a house, its fire level will lower one level with probability 0.8, otherwise it will increase with probability 0.6. If both firefighters go to the same house, its fire level will certainly decrease to the lowest level, i.e., with probability 1. Agents receive a negative reward equivalent to the level of the fire at each house, $R(H_i) = [0 \ -1 \ -2]$.

4.7.1.3 Experimental Setup

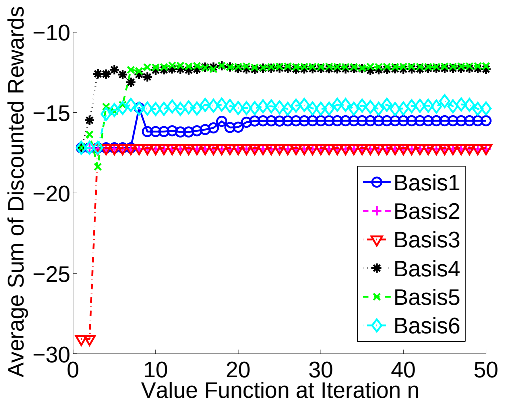
Table 4.1 shows the dimension of the models we tested with. Unless mentioned otherwise, all network management models are run with a belief set of 1000 points and fire fighting models with 500 belief points. Value iteration is run for 50 iterations, and results are averaged over 100 runs of 50 steps each.

All results are obtained with an implementation of the point-based POMDP solver Symbolic Perseus [2], adapted to use linear value functions. For better analysis of our results we introduce some terms used through this section:

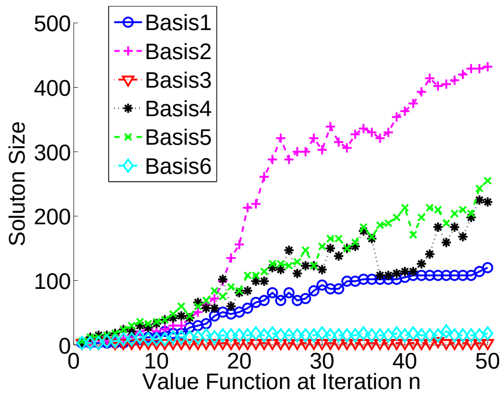
- *Average sum of discounted rewards*: the sum of all rewards obtained by the agent averaged over all runs of the problem, computed as $\frac{\sum_i^N \sum_t \gamma^t R_{i,t}}{N}$, where N is the total number of runs.

#	$h_i(X_i)$
1	$[-1 \quad 1 \quad -1]$
2	$[0 \quad -1 \quad -2]$
3	$[2 \quad 1 \quad 0]$
4	$[2 \quad 0 \quad -1]$
5	$[5 \quad -2 \quad -5]$
6	$[3 \quad 1 \quad -1]$

(A) Basis functions



(B) Average sum of discounted rewards



(C) Total # of values stored

FIGURE 4.6: Fire fighting problem. (a) Basis functions tested. (b), (c) Comparison of different basis functions.

- *Solution size*: a measure of the space needed to store the value function in memory. It is measured as the total of values stored to represent the value function. For a flat problem representation, that would be $|S||V|$, where $|V|$ is the number of vectors in the value function. For a value function representation using ADDs that means the total number of leafs, summed over all vectors. Finally, with linear value functions it is measured as $n_h|\widetilde{V}|+|H|$, where $|\widetilde{V}|$ is the number of vectors in the approximated value function and $|H|$ is the number of leafs used to represent the set of basis functions as ADDs.

4.7.2 Comparing Basis Functions

Our first test consists of running the algorithm with several different basis functions in the fire fighting problem with 3 houses. In these experiments, we considered a fixed number of basis functions equal to the number of state factors (the number of houses), that is, the set of basis function at each trial have 3 bases, $H = \{h_1, h_2, h_3\}$. Each basis function is a vector with the same size as the number of possible fire levels at each house, valuating each level of that state factor, $h_i = [h_i(fl_0)\ h_i(fl_1)\ h_i(fl_2)]$. Thus, each basis' scope is restricted to each house: $\Gamma(h_1) = \{X_1\}$; $\Gamma(h_2) = \{X_2\}$; $\Gamma(h_3) = \{X_3\}$. Basis for all houses are equal at each trial ($h_i = h_j,\ i \neq j$) and we experiment with different mappings from basis to state values, i.e., how much the basis function valuates each value of a state factor. The table in Figure 4.6a shows the different mappings tested.

We ran value iteration for each mapping, and computed the average sum of discounted rewards for the policy obtained at each iteration. In Figure 4.6b we present the average sum of discounted rewards for each mapping over all iterations, and in Figure 4.6c the total solution size needed to represent the value function, for each iteration step. Our results show that the choice of bases influences the method's performance. In this experiments, we are approximating the space in which vectors in the original problem are represented (an $\Re^{27}$ dimensional space) by an $\Re^3$ subspace (given by 3 basis functions). Thus, different mappings for basis functions end in different quality of approximation. If we remember that the reward function for this problem (with 2 agents and 3 houses) is $R_i(X_i) = [0\ -1\ -2]$, we can compare the performance of each basis function with its similarities to the reward function. As Guestrin et al. [132] stress for the MDP case, the success of the technique depends on the ability to capture the most important structure in the value function with a good choice of basis functions.

In our experiments, we distinguish between those bases which are related to the reward function, and those which are not. For instance, all bases except for basis 1 assign a higher value to lower fire levels, and decreasing values for increasing fire levels. Indeed, we notice that the performance of basis 1 is one of the worst in the test in terms of policy quality. It is also important to be aware of numerical issues resulting from the choice of basis functions. In contrast to expectation, bases 2 and 3 do not perform well, when comparing with others in the test. In particular, basis 2 is a direct translation of the reward function to a basis function. These results show how important the choice of basis functions is for the success of linear value function approximation, highlighting how beneficial it is to have fully automated methods to construct basis functions, such as the one we presented in Section 4.5.

4.7.3 Factored Observations

We introduced a method to speed up computation of g^k_{ao} by exploiting a factored representation of the observation model, and tested with a modified model of the fire fighting problem. This model includes one observation factor for each state factor, as we assumed in Section 4.6, and has 2 agents and 3 houses. Figure 4.7 shows the respective DBN for this modified model. In Figure 4.8 we compare the average computation time for each g^k_{ao} vector when using (4.12) and (4.47). The implementation of this test is based on the adapted fire fighting problem, represented in the DBN shown in Figure 4.7. We

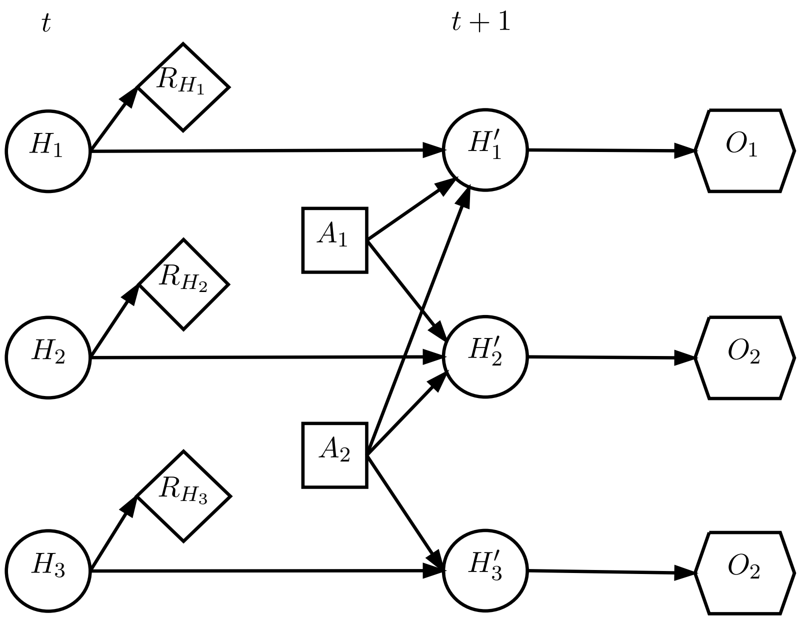


FIGURE 4.7: Two-stage DBN for the modified fire fighting problem with one observation factor for each state factor, where $X_i = \{0, 1, \ldots, k-1\}$; $O_i = \{0, 1, \ldots, l-1\}$.

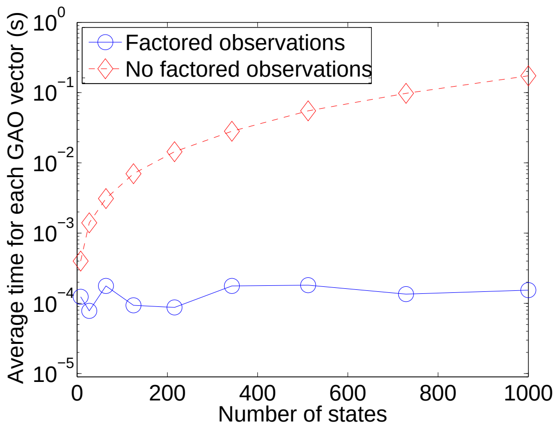
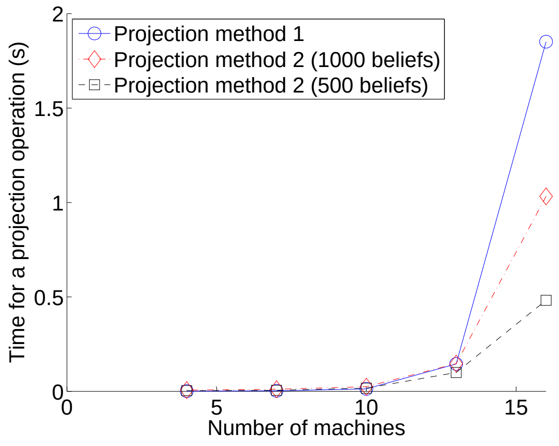


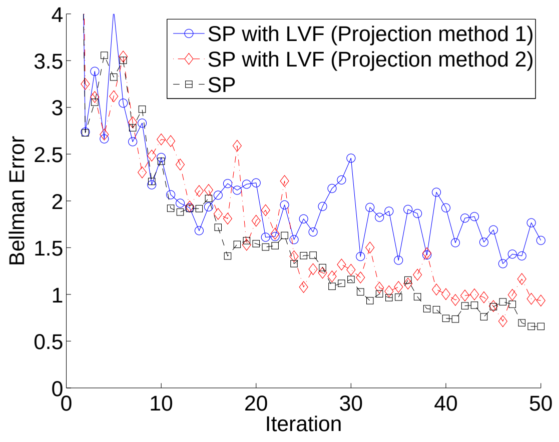
FIGURE 4.8: Factored observations (log-scale y-axis)

compare different instances with a constant number of observations (2 observation levels per house), while we increase the number of states (by increasing one fire level at the time).

The average time of computing g_{ao} vectors does not increase while exploiting factored observations (using (4.47)), in contrast to what happens in the traditional way using (4.12). We conclude that there is an excellent speedup in this step with an increasing number of states, confirming our hypothesis. Since we reduce the number of operations over the state space, its size is the most significant parameter that affects the efficiency of speeding up this step.



(A) Average time to perform a projection operation with n machines. Projections 1 and 2 mean, respectively,state-based projection and belief-based projection



(B) Bellman Error evolution in the network management problem with 13 machines. SP stands for Symbolic Perseus. Projections 1 and 2 mean, respectively, state-based projection and belief-based projection

4.7.4 Comparing Projection Methods

In Figure 4.9a we compare the time it takes to project a vector into the subspace $\mathcal{H}$ in the network management problem, with a $3legs$ configuration, with increasing number of machines. We test our belief-based projection with two sets of belief points with 500 and 1000 beliefs against a state-based projection. Also, in Figure 4.9b we compare the evolution of the Bellman error, $\|V_{n+1} - V_n\|$, over the steps of value iteration for the network problem with a $3legs$ configuration with 13 machines. Here we compare the performance of plain Symbolic Perseus against its extension with linear value function approximation with state and belief-based projections.

We conclude that we can achieve a speedup in the projection operation by using a belief-based method, while performing better in terms of convergence. The smaller the belief set size, the larger the gain in computation time. This is as expected given that, although we need to perform additional inner products to form the set of constraints, our linear program also takes less constraints, therefore performing faster. We also note that the evolution of the Bellman error is similar to the one without linear value function approximation. This shows that our projection method adapts better to point-based methods, as approximates the value function on this set.

4.7.5 Basis Function Construction

In this section we compare our automated procedure using Krylov iteration methods as the baseline. Krylov iteration finds basis functions by expanding the reward function through the transition matrix. Used in the POMDP framework for model compression, we choose it as a baseline due to the similarities with our work. A detailed description of this method may be found in Section 4.1.2.2. Table 4.2 shows the dimension of the sets of basis functions obtained with our procedure for different models of the problem domains we are testing, and we compare our performance in terms of problem solving against that of Krylov basis, in particular with the truncated version of the method.

In Figure 4.9 we compare policy quality for the network problem with 7 machines in a $3legs$ configuration. Our method proves to be competitive with Krylov iteration, by maintaining a good policy quality. Krylov basis yield better results when the set of basis functions grows, but can only provide better results than our method for sets with 20 or more basis vectors. Note that, for this problem size, we obtain a set with 8 basis vectors.

We also compare, for the same model with 7 machines, the computational time needed by both methods to generate sets of basis functions, Table 4.3. Krylov basis are generated by an iterative method, involving several matrix multiplications. Even for a small problem like the one we tested this makes a difference to our method, which involves less complexity.

This shows that we can obtain a set of basis which provides good results with less computational cost which, in turn, leads to further computational savings in problems solving because reduces the complexity of the projection operators.

Network, n machines						Fire fighting, m houses					
n	4	7	10	13	16	m	3	4	5	6	7
n_h	5	8	9	14	17	n_h	19	33	51	73	99

TABLE 4.2: Size n_h of set of basis functions found automatically with our method.

SVD	Truncated Krylov Iteration	
$t(s)$	n_h	$t(s)$
0.255	5	14.16
	10	97.45
	15	266.36
	20	489.18

TABLE 4.3: Time comparison with Krylov Iteration in *3legs* network problem with 7 machines (128 states). SVD Basis is our method with fixed number of basis functions (8 basis for this problem)

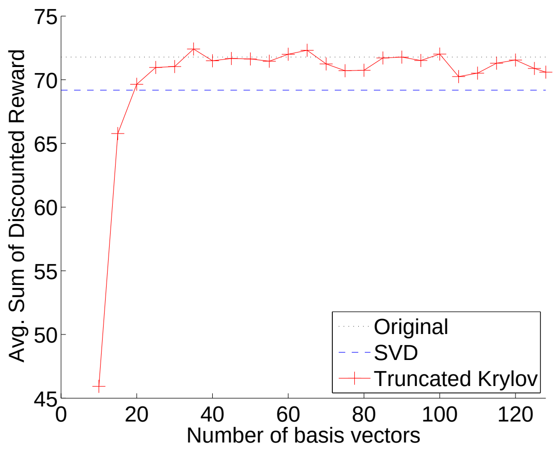


FIGURE 4.9: Policy quality comparison with Krylov Iteration in *3legs* network problem with 7 machines (128 states). SVD Basis is our method with fixed number of basis functions (8 basis for this problem)

4.7.6 Scaling POMDP Solving

In this set of experiments we will see how our value function approximation scheme scales up in POMDP solving.

First, we present in Figure 4.10 the evolution of the average and standard deviation time to perform a backup operation in Symbolic Perseus, with and without value function approximation, in the network management problem with increasing number of machines. When scaling up problem size we also can achieve computational gains in this operation. In our backup operator we perform one additional step, the projection

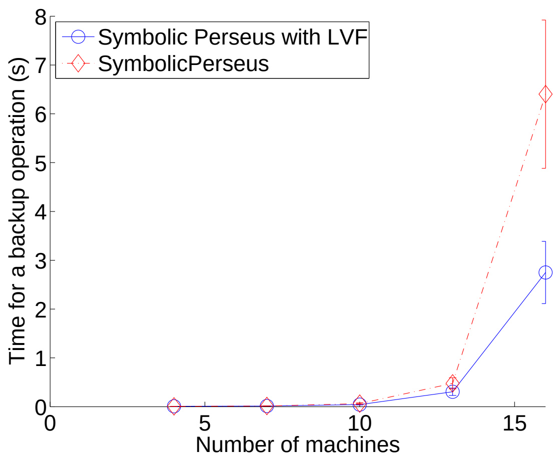


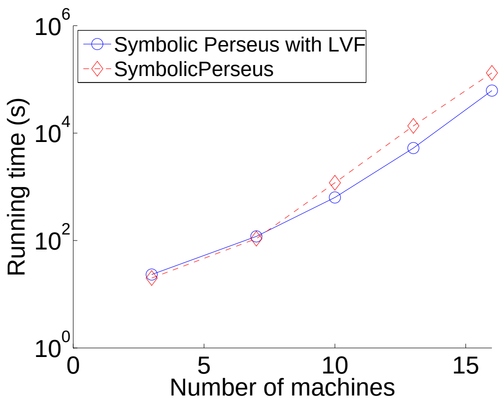
FIGURE 4.10: Average and standard deviation time to perform a backup operation (Algorithm 5) in the network management problem with n machines.

of vectors. However, it is important to note that some steps inside this operator, such as inner products, can be efficiently performed with factored vectors.

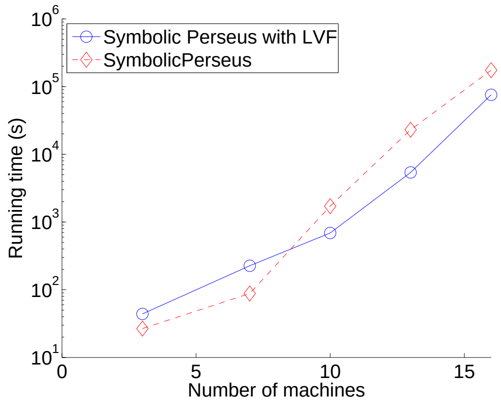
Finally, we compared both methods directly in terms of value iteration computation times, policy quality and size. Figure 4.11 and 4.12 show, respectively, results for the network management problem and the fire fighting problem. We report the running time of 50 iterations, the average summed discount reward and solution sizes (computed as the total numbers of values needed to store the value function).

Network Management

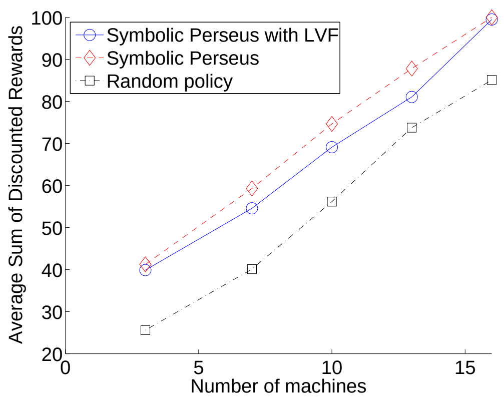
In both configurations of the network management problem a couple of notes are in order. The most noticeable, for any problem size we can reduce solution size by orders of magnitude, which can go up to 3 in the problem with 16 machines. Remember that in this domain the set of basis function is very compact which can explain such a difference in representing the value function. This is due to the structure of the problem, as each state factor (each machine) is only connected to its parent in the network. There is a small decrease in the policy performance, which would be expected given the approximation nature of our method. However, we can consider that there is a good tradeoff between a small decrease in policy quality against a huge decrease in solution size. Moreover, we can achieve a reduction in the computation time. For visualization purposes we show plots in a logarithmic scale, but we note that for larger problems we can cut computation time in half, which is already an important gain.



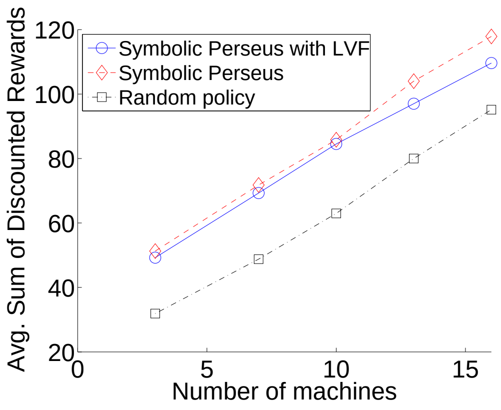
(A) Running times



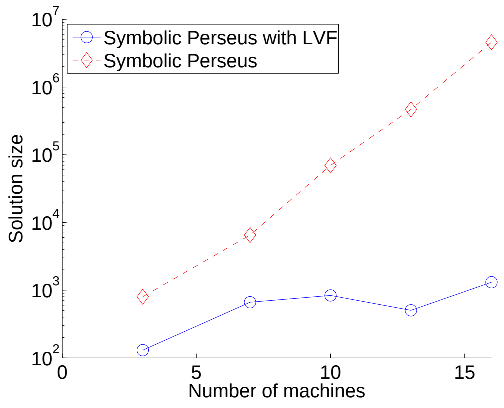
(B) Running times



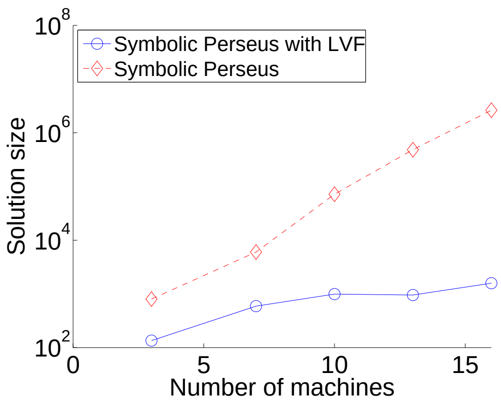
(C) Average sum of discounted rewards



(D) Average sum of discounted rewards



(E) Total # of values stored

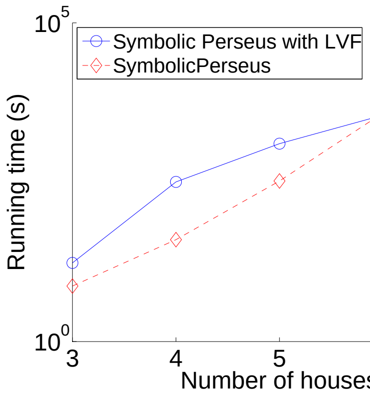


(F) Total # of values stored

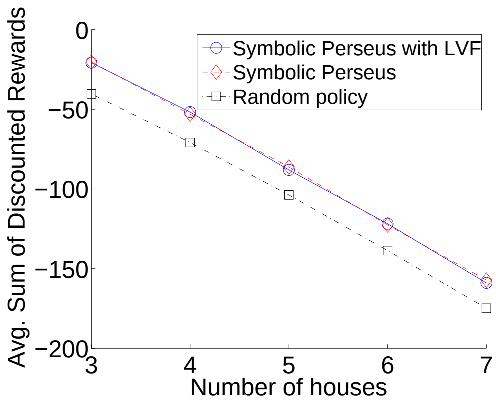
FIGURE 4.11: Network management problem with increasing number of machines and *cycle* (a),(d),(f) and *3legs* (b),(c),(e) topologies. Comparison with Symbolic Perseus and its extension with linear value function approximation.

Fire Fighting

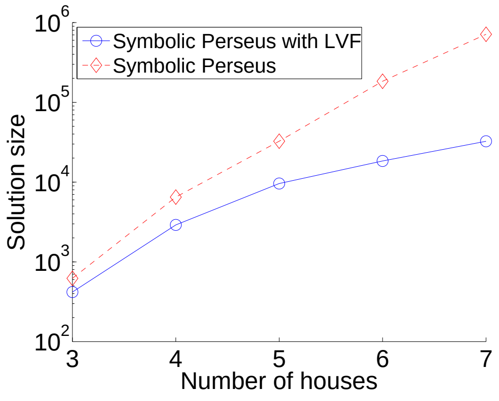
The fire fighting problem has a more complex structure. We note the higher number of basis functions (see Table 4.1), when comparing with the network problem. Therefore,



(A) Running times



(B) Average sum of discounted rewards



(C) Total # of values stored

FIGURE 4.12: Fire fighting problem with 2 agents and increasing number of houses. Comparison with Symbolic Perseus and its extension with linear value function approximation.

it should not be surprising that gains for this problem are smaller than in the previous problem. This shows how important the structure of the problem is when automatically capturing a good set of basis functions. Also, for smaller problems we perform worse in terms of computation time. Nonetheless, when scaling up this problem, the computation time of our method grows slower, leading to best results in larger problems. We can also reduce the solution size for all problem sizes, with almost 2 orders of magnitude difference in the larger problem. In this problem, the average reward is similar for the approach with or without linear value function approximation.

4.8 Discussion

In this chapter we covered linear value function approximation techniques applied to POMDP solving. As previously formulated, this has been a dimensional reduction technique popular in the MDP framework, but under-explored in POMDPs. Bringing such techniques to POMDPs seems a natural step, which had not been taken so far. We believe the reason for that gap, as we found, is that direct application of MDP based linear value function approximation techniques is not straightforward in POMDPs. Therefore, we identified points where we can use POMDP properties to improve linear value function approximation in this framework. Methods for compact representation of value function can, in principle, be plugged in to any solver. We focus in point-based methods, given the good performance they have shown in literature.

We conclude that approximating value functions through a set of basis functions is feasible in practice. We are able to improve POMDP solving by reducing the size of solutions, while being competitive in running times. In approximate value iteration algorithms, there are steps that take the value function outside the space spanned by the basis functions. Therefore, one important operation is the projection of vectors onto this space, which is a key step in these approximation schemes. The complexity of this step is dependent on the complexity of the set of basis function used. Moreover, a good set of basis functions (i.e., a set which captures the most important structure of the problem) is also important to reduce the difference between the original and approximated value functions.

Poupart [2] states that their compressed POMDPs, obtained by Krylov iteration, can be solved more effectively if the POMDP has good myopic policies. This has been supported by experimental results presented by Pajarinen et al. [133], which compared several methods including *Perseus* with truncated Krylov iteration. Similarly, our linear value function approximation approach is also better suited for for problems with good myopic policies. Intuitively, our method to generate basis functions focus on the properties of one iteration, in particular on the dependencies of the DBN. After many iterations the scope of dependence grows and our approximate representation should be less effective.

An interesting issue is how can we be able to generalize the use of linear value function approximation for a broader set of problems. Usually problems can be viewed from different perspectives, allowing for different model designs. Finding sets of rules which allow for a problem to be redesigned in order to have good myopic policies can be a point of future research.

In the MDP framework, error bounds and convergence guarantees have been derived for solutions with linear value function approximation. Another point of direction for future research would be to derive it for the POMDP case, provided that bounds for projections can be found. However, once again, it is not possible to directly use the same reasoning applied in MDPs to the partially observable case.

Chapter 5

Conclusions and Future Work

To conclude, in this chapter we review the main contributions of this work and provide future research directions.

5.1 Conclusions

In this thesis we studied task planning under uncertainty for intelligent agents. An agent is a general concept which may include from autonomous robots to intelligent computer programs or, in resume, any intelligent actor which interacts with the environment. We are particularly interested in the problem of active cooperative perception, i.e., the problem of (possibly) multiple agents actively acting in the system in order to improve the general situational awareness. For instance, in a scenario with static and mobile sensors where the system's goal is to detect some features in the environment, mobile sensors can change its situation in order to help the static sensors to better observe changes.

Generalizing, we can frame this as a problem of information-gathering, i.e., a problem in which the agent's goal is not to achieve some physical state but rather to reduce the uncertainty about its knowledge of the environment. Task planning problems under uncertainty are naturally framed in the partially observable Markov decision process paradigm, as it provides a solid mathematical framework for the decision-making process and allows for directly modelling the uncertainties in the system. However, classic POMDP formulation is not optimized for information-gathering. On the contrary, objectives are defined over states, thus agents will reduce uncertainty only if that helps to better perform some physical task.

To overcome this limitation, in Chapter 3 we introduce the POMDP-IR framework, for rewarding low-uncertainty beliefs. This is achieved at the cost of extending the actions space with information-reward actions, which do not affect transitions or observations, but only rewards. With this property, the agent can choose at every timestep, to perform a domain-level action (which influences the environment) simultaneously with an information-rewarding action. In the long run, the influence of the information-rewards in the value of policies will influence the agent's behavior towards low-uncertainty beliefs.

We illustrated the properties of POMDP-IR with a multi-objective toy problem, in which a surveillance robot needs to patrol a corridor and balance between task performance and information gain, and a case study on robot surveillance. Though implemented in simulation, the experimental setup, in particular the observation model, is retrieved from a real scenario. Moreover, we compared our information-gain framework against the ρPOMDP model and conclude that, with the same number of belief points, POMDP-IR achieves better results.

The formal properties of POMDPs come at a cost. In general, computing exact solutions for POMDPs is an intractable problem, making POMDP application in realistic scenarios impractical. This led to several approximation methods, which balance between computing sub-optimal solutions, but still with good policy quality. Most common approaches attempt to reduce the complexity of the model, search for policies in a reduced policy space, or by computing sub-optimal value functions. Most value function approximation approaches attempt to reduce the number of vectors used to represent the value function. In turn, linear value function approximation attempts to reduce the representation size for each vector. In Chapter 4 we introduce linear value function approximation applied to POMDP solving. With this formulation, vectors in the value function are represented as a linear combination of basis functions, which can be compactly represented with factored models (if each basis has a scope that is a subset of the state space). So far, this approximation technique have been mostly developed and applied in the context of MDPs (both in planning and reinforcement learning). We highlight the practical implications of translating it to the POMDP context.

With linear value function approximation methods, every time an operator in the value iteration takes the value function outside the space spanned by the set of basis functions the algorithm needs to project it back to that space. This implies one additional operator, the projection operator. Besides introducing additional computational complexity, this operator adds some error in the value function. In POMDP solving we must call this operator after each vector backup while in MDP we call it only after each iteration. Thus, the number of projections in POMDP is much higher. Nonetheless, our experiments show that the added number of operations may be balanced by the reduced complexity on other operations.

The success of this technique is highly dependent on the ability of capturing the most important structure of the problem with good sets of basis functions. Based on factored models and the ability to decouple parts of equations in the backup procedure, according to variables' scopes in the model, we derive an automated procedure to automatically construct sets of basis functions. We demonstrated in benchmark problems that this method works in practice, and we can obtain large gains in solution representation, while sacrificing little policy quality.

5.2 Future work

In this thesis, in particular in the active perception scenarios, we considered several cooperating sensors in the environment, but only one is an active sensor. In practice, this is formalized as a single-agent decision-making problem. However, we can easily imagine problems which extend to multiple decision makers, e.g., surveillance scenarios with several robots or active cameras. A requirement for treating (parts of) the system as a centralized POMDP is fast and reliable communication, as cameras and robots need to share local observations [134]. When communication delays are limited and potentially stochastic, the problem can be modeled as a multiagent POMDP with delayed communication [135, 136]. Finally, when no communication channel is present, the problem can be modeled as a decentralized POMDP [137, 138]. Extending the POMDP-IR framework to any of these multiagent models is promising.

Despite the improvements in POMDP solving with different approximation techniques found in literature, its application in realistic, rich scenarios is not yet straightforward. There is still space for improvement, in particular in what concerns exploiting the model structure. Most methods which exploit factored structure focus on conditional independence in the state space, but the observation space is becoming the bottleneck for POMDP solving. In our work, we trace some steps towards mitigating this problem, by noting that factored basis functions in linear value function approximation allows for decoupling some mathematical steps. Some strategies that assume to reduce the complexity of the observation space have been proposed, such as clustering of observations [122], or asynchronous formulation of multiagent problems [139].

Innovative ideas for new approximation techniques for POMDPs keep appearing in literature. One interesting alternative approach is to drop the α-vector representation for the value function and approximate it with a continuous function over the belief space.

Exploiting the property that the value function must be a convex function in the belief state, an approximation based on a quadratic function can be derived [140]. This approach has the promising property that it allows for parametrized reward functions defined over the belief space, thus allowing to directly reward low-uncertainty beliefs. Other approaches include online planning [141] that, instead of planning for all possible scenarios, tries to determine the optimal action for the current belief. Though scalable for large POMDPs [142], online algorithms can be improved for highly uncertain domains [143] where information-gathering actions are essential for task performance. A combinations of these approaches with information-rewards could allow for efficient and scalable planning algorithms for real applications of active cooperative perception.

Bibliography

[1] Matthijs T. J. Spaan. Partially observable Markov decision processes. In Marco Wiering and Martijn van Otterlo, editors, *Reinforcement Learning: State of the Art*, pages 387–414. Springer Verlag, 2012.

[2] Pascal Poupart. *Exploiting Structure to Efficiently Solve Large Scale Partially Observable Markov Decision Processes*. PhD thesis, University of Toronto, 2005.

[3] Mauricio Araya-López. *Des algorithmes presque optimaux pour les problémes de décisione séquentielle à des fins de collecte d'information*. PhD thesis, University of Lorraine, 2013.

[4] Guy Shani, Joelle Pineau, and Robert Kaplow. A survey of point-based pomdp solvers. *Autonomous Agents and Multi-Agent Systems*, 27(1):1–51, July 2013. ISSN 1387-2532.

[5] Druzdzel M.J. and Flynn R.R. Decision support systems. In *Encyclopedia of Library and Information Sciences*. Taylor & Francis, Inc, 2003.

[6] W. Burgard, A.B. Cremers, Dieter Fox, D. Hhnel, G. Lakemeyer, D. Schulz, W. Steiner, and Sebastian Thrun. The interactive museum tour-guide robot. In *Proc. of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, 1998.

[7] M. Veloso, J. Biswas, B. Coltin, S. Rosenthal, T. Kollar, C. Mericli, M. Samadi, S. Brandao, and R. Ventura. Cobots: Collaborative robots servicing multi-floor buildings. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5446–5447, Oct 2012.

[8] Jim Blythe. Artificial intelligence today. chapter An Overview of Planning Under Uncertainty, pages 85–110. Springer-Verlag, Berlin, Heidelberg, 1999. ISBN 3-540-66428-9.

[9] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003. ISBN 0137903952.

[10] Robert Duncan Luce and Howard Raffa. *Games and decisions : introduction and critical survey*. Wiley, New York, 1957. ISBN 0-471-55341-7.

[11] Jerome A. Feldman and Robert F. Sproull. Decision theory and artificial intelligence ii: The hungry monkey. *Cognitive Science*, 1(2):158 – 192, 1977.

[12] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101 (1-2):99–134, 1998.

[13] Richard D. Smallwood and Edward J. Sondik. The Optimal Control of Partially Observable Markov Processes Over a Finite Horizon. *Operations Research*, 21(5): 1071–1088, 1973.

[14] S. Candido and S. Hutchinson. Minimum uncertainty robot navigation using information-guided pomdp planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 6102–6108, May 2011.

[15] R. Gosangi and R. Gutierrez-Osuna. Active temperature programming for metal-oxide chemoresistors. *Sensors Journal, IEEE*, 10(6):1075–1082, June 2010. ISSN 1530-437X.

[16] Shaohui Ma and Hao Zhang. A dynamic crm model based on pomdp. In *Fuzzy Systems and Knowledge Discovery, 2008. FSKD '08. Fifth International Conference on*, volume 2, pages 55–59, Oct 2008.

[17] A. R. Cassandra. A survey of POMDP applications. In *Working Notes of AAAI 1998 Fall Symposium on Planning with Partially Observable Markov Decision Processes*, pages 17–24, 1998.

[18] Alberto Sanfeliu, Norihiro Hagita, and Alessandro Saffiotti. Network robot systems. *Robot. Auton. Syst.*, 56(10):793–797, October 2008. ISSN 0921-8890.

[19] A. Saffioti, P. Lima, H. Levent Akin, A. Birk, A. Bonarini, G. Kraetzschmar, D. Nardi, E. Pagello, M. Reggiani, A. Sanfeliu, and M. Spaan. Two "hot issues" in cooperative robotics: Network robot systems, and formal models and methods for cooperation. EURON Special Interest Group on Cooperative Robotics, 2008.

[20] Matthijs T. J. Spaan, Tiago S. Veiga, and Pedro U. Lima. Active cooperative perception in network robot systems using POMDPs. In *Proc. of International Conference on Intelligent Robots and Systems*, pages 4800–4805, 2010.

[21] Alberto Sanfeliu, Juan Andrade-Cetto, Marco Barbosa, Richard Bowden, Jesús Capitán, Andreu Corominas, Andrew Gilbert, John Illingworth, Luis Merino,

Josep M. Mirats, Plinio Moreno, Aníbal Ollero, João Sequeira, and Matthijs T. J. Spaan. Decentralized sensor fusion for ubiquitous networking robotics in urban areas. *Sensors*, 10(3):2274–2314, 2010.

[22] Matthijs T. J. Spaan. Cooperative active perception using POMDPs. In *AAAI 2008 Workshop on Advancements in POMDP Solvers*, July 2008.

[23] Christos Papadimitriou and John N. Tsitsiklis. The complexity of markov decision processes. *Math. Oper. Res.*, 12(3):441–450, August 1987. ISSN 0364-765X.

[24] Omid Madani, Steve Hanks, and Anne Condon. On the undecidability of probabilistic planning and related stochastic optimization problems. *Artif. Intell.*, 147 (1-2):5–34, July 2003. ISSN 0004-3702.

[25] Joelle Pineau, Geoffrey Gordon, and Sebastian Thrun. Anytime point-based approximations for large pomdps. *J. Artif. Int. Res.*, 27(1):335–380, November 2006.

[26] Nevin L. Zhang and Wenju Liu. A model approximation scheme for planning in partially observable stochastic domains. *J. Artif. Int. Res.*, 7(1):199–230, November 1997. ISSN 1076-9757.

[27] Thomas Dean, Robert Givan, and Sonia Leach. Model reduction techniques for computing approximately optimal solutions for markov decision processes. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, pages 124–131, 1997.

[28] Milos Hauskrecht. Value-function approximations for partially observable markov decision processes. *Journal of Artificial Intelligence Research*, 13:33–94, 2000.

[29] N. Roy, G. Gordon, and S. Thrun. Finding approximate POMDP solutions through belief compression. *Journal of Artificial Intelligence Research*, 23:1–40, 2005.

[30] Matthijs T.J. Spaan, Tiago S. Veiga, and Pedro U. Lima. Decision-theoretic planning under uncertainty with information rewards for active cooperative perception. *Autonomous Agents and Multi-Agent Systems*, pages 1–29, 2014. ISSN 1387-2532.

[31] Tiago S. Veiga, Matthijs T. J. Spaan, and Pedro U. Lima. Point-based POMDP solving with factored value function approximation. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 2512–2518, 2014.

[32] Tiago S. Veiga, Matthijs T. J. Spaan, and Pedro U. Lima. Improving value function approximation in factored pomdps by exploiting model structure. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 1827–1828, 2015. Extended abstract.

[33] Richard Bellman. A markovian decision process. *Indiana Univ. Math. J.*, 6:679–684, 1957. ISSN 0022-2518.

[34] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994. ISBN 0471619779.

[35] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[36] Richard Bellman, Robert Kalaba, and Bella Kotkin. Polynomial approximation–a new computational technique in dynamic programming: Allocation processes. *Mathematics of Computation*, 17(82):pp. 155–161, 1963. ISSN 00255718.

[37] Carlos Guestrin. *Planning Under Uncertainty in Complex Structured Environments*. PhD thesis, Stanford University, 2003.

[38] John N. Tsitsiklis and Benjamin van Roy. Feature-based methods for large scale dynamic programming. *Mach. Learn.*, 22(1-3):59–94, January 1996. ISSN 0885-6125.

[39] Daniel J. Lizotte. Convergent fitted value iteration with linear function approximation. In J. Shawe-Taylor, R.S. Zemel, P.L. Bartlett, F. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2537–2545. Curran Associates, Inc., 2011.

[40] Daphne Koller and Ronald Parr. Computing factored value functions for policies in structured mdps. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 1332–1339. Morgan Kaufmann, 1999.

[41] Ronald Parr, Christopher Painter-Wakefield, Lihong Li, and Michael L. Littman. Analyzing feature generation for value-function approximation. In *Proceedings of the Twenty-Fourth International Conference on Machine Learning (ICML-07)*, page 737744, 2007.

[42] C. Guestrin, D. Koller, and R. Parr. Max-norm projections for factored MDPs. In *Seventeenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 673–680, Seattle, Washington, August 2001.

[43] Eduard Stiefel. Note on jordan elimination, linear programming and chebyshev approximation. *Numerische Mathematik*, pages 2:1–17, 1960.

[44] Edward J. Sondik. The optimal control of partially observable markov processes over the infinite horizon: Discounted costs. *Operations Research*, 26(2):282–304, 1978. ISSN 0030364X, 15265463.

[45] George E. Monahan. A Survey of Partially Observable Markov Decision Processes: Theory, Models, and Algorithms. *Management Science*, 28(1):1–16, 1982.

[46] Chelsea C. White, III. A survey of solution techniques for the partially observed markov decision process. *Annals of Operations Research*, 32(1):215–230, 1991.

[47] Anthony Cassandra, Michael L. Littman, and Nevin L. Zhang. Incremental pruning: A simple, fast, exact method for partially observable markov decision processes. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, pages 54–61. Morgan Kaufmann Publishers, 1997.

[48] Michael L. Littman, Anthony R. Cassandra, and Leslie Pack Kaelbling. Readings in agents. chapter Learning Policies for Partially Observable Environments: Scaling Up, pages 495–503. Morgan Kaufmann Publishers Inc., 1998. ISBN 1-55860-495-2.

[49] Anthony R. Cassandra, Leslie Pack Kaelbling, and James A. Kurien. Acting under uncertainty: Discrete bayesian models for mobile robot navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1996.

[50] Thomas Dean, Robert Givan, and Sonia Leach. Model reduction techniques for computing approximately optimal solutions for markov decision processes. In *Thirteenth Conference on Uncertainty in Artificial Intelligence*, pages 124–131. Morgan Kaufmann, 1997.

[51] Milos Hauskrecht. Value-function approximations for partially observable markov decision processes. *Journal of Artificial Intelligence Research*, 13:33–94, 2000.

[52] Michael L. Littman. Memoryless policies: Theoretical limitations and practical results. In *Proceedings of the Third International Conference on Simulation of Adaptive Behavior : From Animals to Animats 3: From Animals to Animats 3*, SAB94, pages 238–245, Cambridge, MA, USA, 1994. MIT Press. ISBN 0-262-53122-4.

[53] Satinder P. Singh, Tommi Jaakkola, and Michael I. Jordan. Learning without state-estimation in partially observable Markovian decision processes. In William W. Cohen and Haym Hirsch, editors, *Proceedings of the Eleventh International Conference on Machine Learning (ICML 1994)*, pages 284–292, San Francisco, CA, USA, 1994. Morgan Kauffman. ISBN 1-55860-335-2.

[54] III Chelsea C. White and William T. Scherer. Finite-memory suboptimal design for partially observed markov decision processes. *Operations Research*, 42(3):439–455, 1994.

[55] R. Andrew McCallum. Instance-based utile distinctions for reinforcement learning with hidden state. In Armand Prieditis and Stuart J. Russell, editors, *In Proceedings of the Twelfth International Conference on Machine Learning (ICML 1995)*, pages 387–395, San Francisco, CA, USA, 1995. Morgan Kauffman. ISBN 1-55860-377-8.

[56] Pascal Poupart and Craig Boutilier. Bounded finite state controllers. In *Advances in Neural Information Processing Systems 16*, pages 823–830. 2004.

[57] Peters J. and Schaal S. Policy gradient methods. In *Springer Encyclopedia of Machine Learning*. Springer, 2010.

[58] Leemon Baird and Andrew Moore. Gradient descent for general reinforcement learning. In *Advances in Neural Information Processing Systems 11*, pages 968–974. MIT Press, 1999.

[59] Jonathan Baxter and Peter L. Bartlett. Infinite-horizon policy-gradient estimation. *J. Artif. Int. Res.*, 15(1):319–350, November 2001. ISSN 1076-9757.

[60] Douglas Aberdeen, Olivier Buffet, and Owen Thomas. Policy-gradients for psrs and pomdps. In *Proceedings of the 11th International Conference on Artificial Intelligence and Statistics*, 2007.

[61] Ronen I. Brafman. A heuristic variable grid solution method for pomdps. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Conference on Innovative Applications of Artificial Intelligence*, pages 727–733, 1997.

[62] Joelle Pineau, Geoff Gordon, and Sebastian Thrun. Point-based value iteration: an anytime algorithm for pomdps. In *Proceedings of the 18th international joint conference on Artificial intelligence*, IJCAI'03, pages 1025–1030, 2003.

[63] I. R. Nourbakhsh, R. Powers, and S. Birchfield. DERVISH - an office-navigating robot. In *AI Magazine*, pages 16(2):53–60, 1995.

[64] Michael Littman, Anthony R. Cassandra, and Leslie Pack Kaelbling. Learning policies for partially observable environments: Scaling up. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 362–370, 1995.

[65] Eric A. Hansen. Solving POMDPs by Searching in Policy Space. In *Fourteenth International Conference on Uncertainty In Artificial Intelligence (UAI-98)*, pages 211–219, 1998.

[66] William S. Lovejoy. Computationally feasible bounds for partially observed markov decision processes. *Oper. Res.*, 39(1):162–175, February 1991.

[67] Matthijs T. J. Spaan and Nikos Vlassis. Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research*, 24:195–220, 2005.

[68] Trey Smith and Reid Simmons. Heuristic search value iteration for POMDPs. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, UAI '04, pages 520–527, 2004.

[69] Joelle Pineau and Geoff Gordon. POMDP planning for robust robot control. In *in: The Twelveth International Symposium on Robotics Research*, 2005.

[70] Guy Shani, Ronen I. Brafman, and Solomon E. Shimony. Forward search value iteration for pomdps. In *Proc. Int. Joint Conf. on Artificial Intelligence*. IJCAI, 2007.

[71] Hanna Kurniawati, David Hsu, and Wee Sun Lee. Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In *In Proc. Robotics: Science and Systems*, pages 65–72, 2008.

[72] Craig Boutilier and David Poole. Computing optimal policies for partially observable decision processes using compact representations. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*, pages 1168–1175, 1996.

[73] Eric A. Hansen and Zhengzhu Feng. Dynamic programming for pomdps using a factored state representation. In *Proceedings of the Fifth International Conference on AI Planning Systems*, pages 130–139, 2000.

[74] Thomas Dean and Keiji Kanazawa. A model for reasoning about persistence and causation. *Comput. Intell.*, 5(3):142–150, 1990.

[75] Kevin Patrick Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, University of California, 2002.

[76] J.A. Tatman and R.D. Shachter. Dynamic programming and influence diagrams. *IEEE Transactions on Systems, Man and Cybernetics*, 20(2):365–379, 1990.

[77] Jesse Hoey, Robert St-aubin, Alan Hu, and Craig Boutilier. Spudd: Stochastic planning using decision diagrams. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 279–288, 1999.

[78] Guy Shani, Pascal Poupart, Ronen I. Brafman, and Solomon E. Shimony. Efficient ADD operations for point-based algorithms. In *Int. Conf. on Automated Planning and Scheduling*, 2008.

[79] Mauricio Araya-López, Olivier Buffet, Vincent Thomas, and François Charpillet. A pomdp extension with belief-dependent rewards. In *Advances in Neural Information Processing Systems 23*, pages 64–72, 2010.

[80] AnYuan Guo. Decision-theoretic active sensing for autonomous agents. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1002–1003, 2003.

[81] Wolfram Burgard, Dieter Fox, and Sebastian Thrun. Active mobile robot localization by entropy minimization. In *Proc. of the Second Euromicro Workshop on Advanced Mobile Robotics*, pages 155–162, 1997.

[82] Marco Barbosa, Alexandre Bernardino, Dario Figueira, José Gaspar, Nelson Gonçalves, Pedro U. Lima, Plinio Moreno, Abdolkarim Pahliani, José Santos-Victor, Matthijs T. J. Spaan, and João Sequeira. ISRobotNet: A testbed for sensor and robot network systems. In *Proc. of International Conference on Intelligent Robots and Systems*, pages 2827–2833, 2009.

[83] Nikos Vlassis, Geoff Gordon, and Joelle Pineau. Planning under uncertainty in robotics. *Robotics and Autonomous Systems*, 54(11), 2006. Special issue.

[84] Reid Simmons and Sven Koenig. Probabilistic robot navigation in partially observable environments. In *Proc. Int. Joint Conf. on Artificial Intelligence*, 1995.

[85] Rosemary Emery-Montemerlo, Geoff Gordon, Jeff Schneider, and Sebastian Thrun. Game theoretic control for robot teams. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2005.

[86] Christopher Amato, George Konidaris, Gabriel Cruz, Christopher A. Maynor, Jonathan P. How, and Leslie Pack Kaelbling. Planning for decentralized control of multiple robots under uncertainty. In *ICAPS-14 Workshop on Planning and Robotics*, 2014.

[87] João V. Messias, Matthijs T. J. Spaan, and Pedro U. Lima. GSMDPs for multi-robot sequential decision-making. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, pages 1408–1414, 2013.

[88] João Messias. *Decision-Making under Uncertainty for Real Robot Teams.* PhD thesis, Universidade de Lisboa, 2014.

[89] Jesús Capitán, Matthijs T. J. Spaan, Luis Merino, and Aníbal Ollero. Decentralized multi-robot cooperation with auctioned POMDPs. *International Journal of Robotics Research*, 32(6):650–671, 2013.

[90] Alan Fern, Sriraam Natarajan, Kshitij Judah, and Prasad Tadepalli. A decision-theoretic model of assistance. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, pages 1879–1884, 2007.

[91] Jennifer Boger, Pascal Poupart, Jesse Hoey, Craig Boutilier, Geoff Fernie, and Alex Mihailidis. A decision-theoretic approach to task assistance for persons with dementia. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pages 1293–1299, 2005.

[92] J. Pineau, M. Montemerlo, M. Pollack, N. Roy, and S. Thrun. Towards robotic assistants in nursing homes: Challenges and results. *Robotics and Autonomous Systems*, 42(3–4):271–281, 2003.

[93] N. Roy, G. Gordon, and S. Thrun. Planning under uncertainty for reliable health care robotics. In *Proc. of the Int. Conf. on Field and Service Robotics*, 2003.

[94] Luis Merino, Joaquín Ballesteros, Noé Pérez-Higueras, Rafael Ramón-Vigo, Javier Pérez-Lara, and Fernando Caballero. Robust person guidance by using online POMDPs. In *ROBOT2013: First Iberian Robotics Conference*, volume 253 of *Advances in Intelligent Systems and Computing*. Springer, 2014.

[95] Ruzena Bajcsy and Ruzena Bajcsy. Active perception. In *Proc IEEE, 76:996–1005*, 1988.

[96] L. Mihaylova, T. Lefebvre, H. Bruyninckx, K. Gadeyne, and J. De Schutter. A comparison of decision making criteria and optimization methods for active robotic sensing. In I. Dimov, I. Lirkov, S. Margenov, and Z. Zlatev, editors, *Numerical Methods and Applications*, volume 2542 of *LNCS*, pages 316–324. Springer, 2003.

[97] N. Roy, W. Burgard, D. Fox, and S. Thrun. Coastal navigation – mobile robot navigation with uncertainty in dynamic environments. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1999.

[98] Javier Velez, Garrett Hemann, Albert S. Huang, Ingmar Posner, and Nicholas Roy. Planning to perceive: Exploiting mobility for robust object detection. In *Int. Conf. on Automated Planning and Scheduling (ICAPS)*, Freiburg, Germany, Jun. 2011.

[99] Andreas Krause and Carlos Guestrin. Near-optimal observation selection using submodular functions. In *Proceedings of the 22nd National Conference on Artificial Intelligence - Volume 2*, pages 1650–1654, 2007.

[100] Andreas Krause, Ajit Singh, and Carlos Guestrin. Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research (JMLR)*, 9:235–284, February 2008.

[101] Andreas Krause, Jure Leskovec, Carlos Guestrin, Jeanne Vanbriesen, and Christos Faloutsos. Efficient sensor placement optimization for securing large water distribution networks. *Journal of Water Resources Planning and Management*, 134(6): 516–526, 2008.

[102] Prabhu Natarajan, Trong Nghia Hoang, Kian Hsiang Low, and Mohan Kankanhalli. Decision-theoretic approach to maximizing observation of multiple targets in multi-camera surveillance. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, pages 155–162, 2012.

[103] Shiqi Zhang and Mohan Sridharan. Active visual sensing and collaboration on mobile robots using hierarchical pomdps. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, pages 181–188, 2012.

[104] Jennifer Renoux, Abdel-Illah Mouaddib, and Simon LeGloannec. Distributed decision-theoretic active perception for multi-robot active information gathering. In Vicen Torra, Yasuo Narukawa, and Yasunori Endo, editors, *Modeling Decisions for Artificial Intelligence*, volume 8825 of *Lecture Notes in Computer Science*, pages 60–71. Springer International Publishing, 2014.

[105] Matthijs T. J. Spaan and Pedro U. Lima. A decision-theoretic approach to dynamic sensor selection in camera networks. In *Int. Conf. on Automated Planning and Scheduling*, pages 279–304, 2009.

[106] N. Atanasov, B. Sankaran, J. Le Ny, G.J. Pappas, and K. Daniilidis. Nonmyopic view planning for active object classification and pose estimation. *Robotics, IEEE Transactions on*, 30(5):1078–1090, Oct 2014.

[107] Adam Eck and Leen-Kiat Soh. Evaluating pomdp rewards for active perception. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 3*, pages 1221–1222, 2012. Extended abstract.

[108] Jason D. Williams and Steve Young. Partially observable Markov decision processes for spoken dialog systems. *Computer Speech and Language*, 21(2):393–422, 2007.

[109] Yash Satsangi, Shimon Whiteson, and Matthijs T. J. Spaan. An analysis of piecewise-linear and convex value functions for active perception POMDPs. Technical Report IAS-UVA-15-01, University of Amsterdam, Informatics Institute, 2015.

[110] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, 1st edition, 1996. ISBN 1886529108.

[111] Benjamin Van Roy. *Learning and Value Function Approximation in Complex Decision Processes.* PhD thesis, Massachussetts Institute of Technology, 1998.

[112] Steven J. Bradtke, Andrew G. Barto, and Pack Kaelbling. Linear least-squares algorithms for temporal difference learning. In *Machine Learning*, pages 22–33, 1996.

[113] J. N. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. In *IEEE Transactions on Automatic Control*, volume 42, pages 674–690, 1997.

[114] Francisco S. Melo, Sean P. Meyn, and M. Isabel Ribeiro. An analysis of reinforcement learning with function approximation. In *Proceedings of the 25th International Conference on Machine Learning*, pages 664–671, 2008.

[115] Ronald Parr, Lihong Li, Gavin Taylor, Christopher Painter-Wakefield, and Michael L. Littman. An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *Proceedings of the 25th International Conference on Machine Learning*, pages 752–759, 2008.

[116] Daphne Koller and Ronald Parr. Policy iteration for factored mdps. In *In Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI-00*, pages 326–334, 2000.

[117] Dale Schuurmans and Relu Patrascu. Direct value-approximation for factored mdps. In *Advances in Neural Information Processing Systems 14*, pages 1579–1586. 2002.

[118] Carlos Guestrin, Daphne Koller, and Ronald Parr. Multiagent planning with factored mdps. In *Advances in Neural Information Processing Systems 14*, pages 1523–1530. 2002.

[119] Istvn Szita and Andrs Lrincz. Factored value iteration converges. In *Acta Cybernetica*, pages 18(4):615–635, 2008.

[120] Carlos Guestrin, Daphne Koller, and Ronald Parr. Solving factored pomdps with linear value functions. In *In IJCAI-01 workshop on Planning under Uncertainty and Incomplete Information*, 2001.

[121] Jesse Hoey and Pascal Poupart. Solving pomdps with continuous or large discrete observation spaces. In *Proc. Intl. Joint Conf. on Artificial Intelligence*, pages 1332–1338, 2005.

[122] Amin Atrash. Efficient planning and tracking in pomdps with large observation spaces. In *AAAI Workshop on Statistical and Empirical Approaches for Spoken Dialogue Systems*, 2006.

[123] Alan Carlin and Shlomo Zilberstein. Value-based observation compression for dec-pomdps. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, pages 501–508, 2008.

[124] Relu Patrascu, Pascal Poupart, Dale Schuurmans, Craig Boutilier, and Carlos Guestrin. Greedy linear value-approximation for factored markov decision processes. In *Proceedings of the 18th National Conference on Artificial Intelligence*, pages 285–291, 2002.

[125] Pascal Poupart and Craig Boutilier. Piecewise linear value function approximation for factored mdps. In *Proceedings of the Eighteenth National Conference on AI*, pages 292–299, 2002.

[126] Sridhar Mahadevan. Learning representation and control in markov decision processes: New frontiers. *Foundations and Trends in Machine Learning*, 1(4):403–565, 2008. ISSN 1935-8237.

[127] Sridhar Mahadevan. Representation discovery in sequential decision making. In *AAAI Conference on Artificial Intelligence*, 2010.

[128] Pascal Poupart and Craig Boutilier. Value-directed compression of pomdps. In *Advances in Neural Information Processing Systems 15*, pages 1579–1586. 2003.

[129] Pascal Poupart and Craig Boutilier. Vdcbpi: an approximate scalable algorithm for large pomdps. In *Advances in Neural Information Processing Systems 17*, pages 1081–1088. 2005.

[130] Nicolas Drougard, Florent Teichteil-Knigsbuch, Jean-Loup Farges, and Didier Dubois. Structured possibilistic planning using decision diagrams. In *In 28th AAAI Conference on Artificial Intelligence*, pages 2257–2263, 2014.

[131] Frans A. Oliehoek, Matthijs T. J. Spaan, Shimon Whiteson, and Nikos Vlassis. Exploiting locality of interaction in factored Dec-POMDPs. In *Proc. of Int. Conference on Autonomous Agents and Multi Agent Systems*, pages 517–524, 2008.

[132] Carlos Guestrin, Daphne Koller, Ronald Parr, and Shobha Venkataraman. Efficient solution algorithms for factored mdps. *J. Artif. Int. Res.*, 19(1):399–468, October 2003.

[133] Joni Pajarinen, Jaakko Peltonen, Ari Hottinen, and MikkoA. Uusitalo. Efficient planning in large pomdps through policy graph based factorized approximations. In *Machine Learning and Knowledge Discovery in Databases*, volume 6323 of *Lecture Notes in Computer Science*, pages 1–16. 2010.

[134] David V. Pynadath and Milind Tambe. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of Artificial Intelligence Research*, 16:389–423, 2002.

[135] Frans A. Oliehoek and Matthijs T. J. Spaan. Tree-based solution methods for multiagent POMDPs with delayed communication. In *Proc. of the AAAI Conference on Artificial Intelligence*, pages 1415–1421, 2012.

[136] Matthijs T. J. Spaan, Frans A. Oliehoek, and Nikos Vlassis. Multiagent planning under uncertainty with stochastic communication delays. In *Int. Conf. on Automated Planning and Scheduling*, pages 338–345, 2008.

[137] Daniel S. Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, 27(4):819–840, 2002.

[138] Frans A. Oliehoek, Matthijs T. J. Spaan, and Nikos Vlassis. Optimal and approximate Q-value functions for decentralized POMDPs. *Journal of Artificial Intelligence Research*, 32:289–353, 2008.

[139] João V. Messias, Matthijs T.J. Spaan, and Pedro U. Lima. Multiagent pomdps with asynchronous execution. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, pages 1273–1274, 2013.

[140] Tao Wang, Pascal Poupart, Michael Bowling, and Dale Schuurmans. Compact, convex upper bound iteration for approximate POMDP planning. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI)*, pages 1245–1251, 2006.

[141] Stéphane Ross, Joelle Pineau, Sébastien Paquet, and Brahim Chaib-draa. Online planning algorithms for pomdps. *J. Artif. Int. Res.*, 32(1):663–704, July 2008.

[142] David Silver and Joel Veness. Monte-carlo planning in large pomdps. In *Advances in Neural Information Processing Systems 23*, pages 2164–2172. 2010.

[143] Adam Eck and Leen-Kiat Soh. Online heuristic planning for highly uncertain domains. In *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-agent Systems*, pages 741–748, 2014.