Planning under Uncertainty in Constrained and Partially Observable Environments

Planning under Uncertainty in Constrained and Partially Observable Environments

Planning under Uncertainty in Constrained and Partially Observable Environments

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof. dr. ir. T. H. J. J. van der Hagen,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op maandag 27 mei 2019 om 12:30 uur

door

Erwin Martinus Petrus WALRAVEN

ingenieur in de computerwetenschappen,
Technische Universiteit Delft, Nederland,
geboren te Lisse, Nederland.

The research reported in this dissertation has been funded by the Netherlands Organisation for Scientific Research (NWO), as part of the Uncertainty Reduction in Smart Energy Systems (URSES) program.

SIKS Dissertation Series No. 2019-15
The research reported in this dissertation has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

ISBN: 978-94-6384-034-7
Printing: Ridderprint BV

Contents

1 Introduction **1**

1.1 Planning and decision making 2

1.2 Planning in smart distribution grids. 3

1.3 Modeling of uncertainty and constraints 5

1.4 Contributions of the dissertation 8

1.5 Dissertation overview 11

2 Planning under uncertainty **13**

2.1 Markov Decision Processes 13

2.1.1 States, actions, rewards and policies. 13

2.1.2 Infinite-horizon problems 15

2.1.3 Finite-horizon problems. 16

2.1.4 Linear programming formulations 17

2.1.5 Constrained problems 19

2.1.6 Planning for multiple agents 20

2.2 Partial observability 21

2.2.1 Augmenting MDPs with observations. 21

2.2.2 Belief states and belief updates 22

2.2.3 Infinite-horizon problems 23

2.2.4 Finite-horizon problems. 23

2.2.5 Vector-based value functions and backups 24

2.2.6 Exact value iteration. 26

2.2.7 Vector pruning. 28

2.2.8 Point-based value iteration 32

2.2.9 Representing policies as policy graphs. 33

3 Accelerating optimal planning for POMDPs **35**

3.1 Constraint generation for vector pruning 35

3.1.1 Analysis of the LP formulation 36

3.1.2 Benders decomposition method for linear programs 38

3.1.3 Derivation of a constraint generation procedure 40

3.1.4 Theoretical analysis 44

3.2 Bootstrapping for constraint generation 48
3.2.1 Convergence of value iteration 48
3.2.2 Bootstrapping of linear program constraints 48
3.2.3 Integrating bootstrapping in vector pruning 51
3.3 Experiments 53
3.3.1 Constraint generation experiments 53
3.3.2 Bootstrapping experiments 61
3.4 Related work 69
3.5 Conclusions 71

4 Approximate planning for finite-horizon POMDPs 75
4.1 Planning horizons and discounting 75
4.2 Strategies for solving finite-horizon problems 76
4.3 Discarding the discount factor 78
4.4 Finite-horizon point-based value iteration 80
4.4.1 Time-dependent value functions and backups 81
4.4.2 Time-dependent value upper bounds and bound updates . . 82
4.4.3 Algorithm description of FiVI 83
4.4.4 Belief points and convergence of the algorithm 86
4.5 Backup and update heuristics 88
4.5.1 Perseus Belief Selection (PBS) 89
4.5.2 Dependency-Based Bound Updates (DBBU) 89
4.6 Experiments 91
4.6.1 Performance of FiVI with PBS and DBBU 92
4.6.2 Number of backups executed by PBS 95
4.6.3 Dependency graph construction in DBBU 97
4.6.4 Comparison with alternative methods 98
4.7 Conclusions 101

5 Approximate planning for Constrained POMDPs 103
5.1 Introduction 104
5.1.1 Contributions 106
5.1.2 Chapter outline 107
5.2 Constrained POMDPs 107
5.2.1 Multi-agent Constrained POMDPs 108
5.3 Column Generation for Constrained POMDPs 109
5.3.1 Exact column generation for Constrained POMDPs 110
5.3.2 Analysis of exact column generation 112
5.4 Approximate algorithms for subproblems 117
5.4.1 Policy graphs as policy representation 119
5.4.2 Creating and evaluating a policy graph 120
5.4.3 Adapted column generation algorithm 122

5.4.4 Analysis of the policy graph construction 125
5.5 Experiments 130
5.5.1 Single-agent planning: robot navigation domains 130
5.5.2 Multi-agent planning: condition-based maintenance 132
5.5.3 Multi-agent planning: online advertising 135
5.5.4 Translation of vectors into policy graph 137
5.5.5 Additional details benchmark domains 139
5.6 Related work 140
5.7 Conclusions 142

6 Constrained planning under uncertainty in smart grids 145
6.1 Introduction 146
6.1.1 Contributions 147
6.1.2 Chapter outline 148
6.2 Applications of AI in smart grids 148
6.2.1 Control of flexible loads 149
6.2.2 Energy markets, tariffs and trading 150
6.2.3 Grid control and integration of renewables 151
6.2.4 Load forecasting and smart meter data 152
6.2.5 Grid constraints in optimization algorithms. 153
6.2.6 Summary 154
6.3 Power flows in distribution grids 154
6.4 Congestion management using CMMDPs. 158
6.4.1 Algorithm requirements for congestion management. . . . 158
6.4.2 Constrained Multi-agent Markov Decision Processes 159
6.4.3 Agent goals and guarantees 160
6.4.4 Integrating grid constraints in Constrained MMDPs 161
6.4.5 Preventing violations of constraints: preallocation 163
6.4.6 Reducing violations using empirical bounding 164
6.5 Experiments 169
6.5.1 Experimental setup 169
6.5.2 Uncontrollable flows 172
6.5.3 Uncoordinated electric vehicle charging 173
6.5.4 Constraints on expected power flows 174
6.5.5 Reducing violations using empirical bounding 175
6.5.6 Flexibility of electric vehicles 177
6.5.7 Preallocation of grid capacity 178
6.5.8 Multiple binding grid constraints 179
6.6 Conclusions 181

7 Conclusions **189**
7.1 Contributions and implications 189
7.1.1 Exact value iteration for POMDPs 189
7.1.2 Finite-horizon planning for POMDPs 190
7.1.3 Approximate algorithm for Constrained POMDPs 190
7.1.4 Constrained planning in smart distribution grids 191
7.2 Directions for future work 192

Bibliography **195**

Summary **211**

Samenvatting **213**

List of publications **215**

Acknowledgments **217**

SIKS dissertation series **219**

1 Introduction

The design and analysis of intelligent decision making systems is a major area in computer science and artificial intelligence. These systems perceive their environment and decide autonomously how to act in order to perform a task as well as possible. Intelligent decision making is not only an active topic of academic research. It is also used in several applications and systems that affect our society.

Existing applications based on decision making include control of traffic lights at road intersections, which involves deciding whether certain lanes get priority in order to prevent long queues and congestion (Yousef, Al-Karaki, and Shatnawi, 2010). Another application can be found in elevators in buildings, which decide autonomously how to operate in order to move people to the right floor (Koehler and Ottiger, 2002). Both examples illustrate how algorithms have been applied to solve control and decision making problems. In the near future new applications arise which require more sophisticated intelligent decision making algorithms. A first example is autonomous driving, where a vehicle needs to reason about signs in order to participate in traffic (Levinson et al., 2011). Another example is the development of smart distribution grids in the residential area (Ramchurn et al., 2012), in which decisions need to be made for a large number of consumers, while reasoning about the distribution grid conditions as well as uncertain behavior of, e.g., electric vehicles that require charging. Both applications create algorithmic challenges related to the scalability of algorithms, as well as challenges regarding the ability to reason about uncertain events that occur in the environment.

Driven by the future applications of intelligent decision making, this dissertation focuses on a specific type of intelligent decision making problems in which uncertainty and constraints on resource consumption need to be considered while making decisions. These characteristics are conceptually easy to understand, but from a technical point of view they can make it surprisingly difficult to solve decision making problems. In this introductory chapter we provide an overview of decision making under uncertainty subject to constraints on resource consumption, and we describe the contributions and structure of the dissertation.

1.1. Planning and decision making

We consider decision making problems in which it is required to decide how to act in order to get the best possible performance when performing a certain task. The system that perceives its environment and acts in this environment is known as a software agent, or simply agent, and it performs its task autonomously without human intervention. An agent executes a plan, which tells the agent what to do depending on the things it has perceived in the environment. We are interested in computing such a plan for the agent, which we refer to as *planning*. Execution of plans has a sequential nature, because typically it is required to execute a sequence of actions in multiple subsequent time steps in order to reach a particular goal. For example, heating systems in smart distribution grids decide sequentially how the heating system should be controlled over time in order to maintain a given temperature. It is important to emphasize that planning typically involves deciding *what* an agent does, whereas the field of scheduling focuses on the question *when* an agent needs to do something. In practice planning and scheduling can be seen as complementary and sometimes even mixed problems. This is especially the case when it is necessary to decide what needs to happen during a sequence of multiple time steps, such as the sequential navigation decisions made by a vehicle in the aforementioned autonomous driving application.

Multiple types of planning problems and plans can be distinguished in the planning field. The plans considered in this dissertation can be used to decide during plan execution what an agent should do, depending on things perceived in the environment of the agent. Another type of planning involves classical planning, which focuses on computing a static sequence of actions which ensures that the agent reaches its goal when executing the action sequence (McDermott et al., 1998). Planning problems in general do not only affect individual agents in isolation. In several settings there are multiple agents which potentially influence each other, which needs to be considered when computing plans for the agents. In this dissertation we consider planning problems with multiple cooperative agents, in which the agents aim to achieve a common goal by executing a sequence of actions. Although multiple agents may execute an action at the same time, plan execution can still be seen as a sequential process.

Computing plans for agents may become computationally difficult due to various characteristics of the planning task and the environment. In this dissertation we focus on two characteristics that are often present in real-world decision making

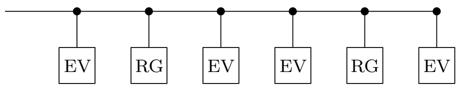


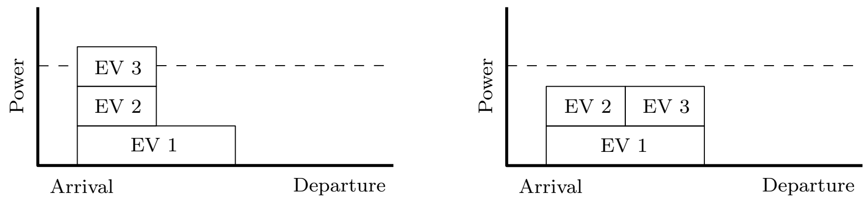
Figure 1.1: Multiple electric vehicles connected to a power line

problems. First, we focus on planning tasks in which the agent faces uncertainty while executing the plan. For example, one can think about uncertainty regarding the travel time to reach a destination. Second, we consider planning tasks which naturally include resource constraints, such as capacity limitations of road networks. Throughout this dissertation we refer to such planning tasks as constrained planning tasks. In the next section we provide a practical motivation which shows how uncertainty and resource constraints arise in a real-world planning problem, based on the smart grid application domain that we briefly introduced before.

1.2. Planning in smart distribution grids

In order to illustrate the application of intelligent decision making in the real world, we consider the development of smart distribution grids in our society. This section provides an introduction to the application domain, which also illustrates the practical relevance of research on planning algorithms. Additionally, the application domain that we describe aligns with the research goals of this dissertation, which we discuss in Section 1.4.

We consider the power distribution grid that provides power to neighborhoods and cities. The power distribution grid serves as a backbone of our society and it is the crucial infrastructure for daily needs such as communication, health care, and transportation. The current grid has been designed decades ago and it faces major changes in the upcoming years due to new developments. In particular, renewable generators such as solar panels become increasingly popular, and there is an increase in the number of electric vehicles. These developments introduce two major problems. First, there is increased uncertainty in the power grid due to the uncertain availability of renewable power and uncertainty in the charging behavior of electric vehicle owners. Second, electric vehicles require a significant amount of power for charging, which creates problems because grids have limited capacity and they have not been designed for large-scale charging. If loads connected to the grid require more power than the grid can accomodate, then the grid becomes congested. In this example the power grid can be seen as a resource that is used by the vehicles while they are charging. Grid congestion can be prevented by increasing the capacity of the grid, but this requires significant infrastructural work and it is considered expensive. An alternative approach relies on planning and scheduling of power consumption using algorithms that consider both uncertainty and resource constraints, in such a way that congestion does not arise.



(a) Constraint violation due to charging (b) Shifting of charging load

Figure 1.2: Charging without planning and charging with a planner that exploits flexibility

We illustrate how planning algorithms can be used for prevention of congestion using a high-level example. We consider electric vehicles (EVs) and renewable generators (RGs) connected to a line in a distribution grid, as visualized in Figure 1.1. Due to the limited capacity of the line it is typically impossible to charge all the vehicles at the same time, because this would lead to power consumption that exceeds the capacity of the line. This is illustrated in Figure 1.2a, in which the height of each block represents the power consumption of the vehicle and the width represents the charging duration. As can be seen, 3 vehicles start to charge immediately upon arrival, leading to a violation of the capacity limit.

Rather than increasing the grid capacity represented by the dashed line, it is possible to plan when vehicles charge their batteries by exploiting their temporal flexibility. The notion of flexibility is illustrated in Figure 1.3, which shows a horizontal timeline representing the arrival and departure time of an individual vehicle. The width of the block indicates the duration of charging, and this charging period can be temporally shifted in case the time required for charging is less than the total time available. The difference between the charging duration and the amount of time available corresponds to the flexibility of an electric vehicle. Shifting the charging periods of electric vehicles by exploiting flexibility can contribute to a decrease of the peak power consumption of the vehicles, such that line capacities are not violated, as illustrated in Figure 1.2b. In the example there is one vehicle that starts charging later in time, which resolves the violation of the capacity limit.

In practice automated planning for shifting of charging periods is difficult due to uncertainties in arrival time, uncertainties in the intended departure time and uncertainty regarding the amount of time required for charging the battery. In addition, there can be many vehicles connected to the same line in a distribution grid, which means that many different vehicles need to be considered when deciding which vehicle needs to shift its charging period. Finally, when shifting loads it is required to consider the capacity limits of multiple lines in the distribution grid.

Based on the application in this section we have seen that uncertainties and resource constraints can naturally arise in real-world planning problems. The next section provides a more general characterization of both concepts, which gives us

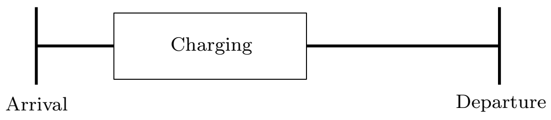


Figure 1.3: Notion of flexibility of electric vehicles

better understanding about the requirements for planning algorithms that solve these problems.

1.3. Modeling of uncertainty and constraints

In the previous sections we observed that uncertainty may arise in planning problems, and we observed that uncertainty potentially affects the decisions that can be made. However, so far we did not specify what types of uncertainty can be distinguished. Below we discuss these uncertainty types in more detail, and we explain why they can make decision making more complicated. Furthermore, we provide a more elaborate introduction to resource constraints, and we explain how characterizations of uncertainty can be combined with constrained planning.

State uncertainty From a planning point of view it is important to know the current state of the environment before decisions can be made. However, in several settings it is difficult for the agent to actually determine this state prior to making a decision regarding the action to take. In such planning problems the environment is called *partially observable*, and the agent has to infer information about the current state based on observations it gets. These observations are related to the state of the environment, but they do not always reveal the actual state of the environment completely. For example, autonomous vehicles have to reason about the current distance to other vehicles (i.e., the current state) when controlling their behavior, but the embedded sensors and cameras may not provide a completely accurate measure of the actual distance. Another example is smart metering in a distribution grid. If a utility company communicates once a week with the smart meter, then the current meter readings are partially observable during the week.

Transition uncertainty A second source of uncertainty comes from the fact that it is not always known how actions taken by an agent influence the environment. For example, an environment may behave stochastically and in that case its state transitions stochastically in response to an action executed by the agent. We can illustrate this using a dishwasher connected to a smart distribution grid, which finishes its program within an hour in 90 percent of the cases, while it runs a bit longer in 10 percent of the cases. When turning on the washing machine, then the total duration of the program is not deterministic, and there is transition uncertainty regarding the state of the washing machine after an hour. Another example is an agent which controls the room temperature, for which the state description only contains the current room temperature. When turning on the heating system, the resulting temperature increase is not deterministic due to, e.g., doors and windows that may be open.

Model uncertainty Decision making systems use a model that is an abstraction of the real world, and unfortunately there is an inherent mismatch between the model and the real environment of the agent. In other words, formalizing the environment using a model brings uncertainty, because the real environment may have characteristics that have not been incorporated in the model. As a result, an intelligent decision making system that operates in an environment that does not correspond the model that was used to construct the system can potentially exhibit undesired behavior. For example, an autonomous device connected to a power grid may introduce violations of grid capacities in case it is unaware of the presence of other agents that require power at the same time. Another example is an autonomous vacuum cleaner that was built for a specific type of environment, which may behave completely different if the actual environment is slightly different during deployment of the system. In this dissertation we do not consider model uncertainty, and we only focus on computing plans based on a given model of the environment. However, it is important to mention that model uncertainty starts to receive increased attention because it becomes more relevant when building robust AI-based systems in the real world (Amodei et al., 2016; Grau-Moya et al., 2016).

Exogenous and endogenous uncertainty The final types of uncertainty that we discuss are exogenous uncertainty and endogenous uncertainty, which represent two classes within transition uncertainty that require some special attention. Exogenous uncertainty can be seen as transition uncertainty that is not influenced by the actions taken by the agent. For example, the decisions made for electric vehicles do not influence the uncertain weather in the next few days. Endogenous uncertainty, on the other hand, is transition uncertainty that is influenced by the actions executed by the agent. For example, charging decisions for electric vehicles may have influence on the uncertain charging demand in the next days. The distinction between both types is important to mention, because some planning algorithms only support exogenous uncertainty and they cannot be used for the other (Defourny, Ernst, and Wehenkel, 2012). The algorithms considered in this dissertation support both types.

In this dissertation we consider problems with state uncertainty and transition uncertainty. We focus on solving planning problems based on a given model, and we do not consider model uncertainty because this type of uncertainty does not directly affect the behavior of agents. Instead, it can be seen as a modeling challenge for control systems in general. Besides sources of uncertainty it may be necessary to consider constraints on consumption of resources while computing and executing a plan. For instance, execution of an action by an agent may require a money investment or usage of equipment. Limited availability of money and equipment imposes constraints on the actions that can be executed by the agent. Throughout this dissertation a constrained planning problem refers to a planning problem which includes resource constraints. Two types of these resource constraints can be distinguished: budget constraints and instantaneous constraints, which we discuss below in more detail.

Budget constraints Constraints on budget can be used to model situations in which there is a finite amount of resources available during the execution of a plan. An intuitive example is a setting in which each decision made by the decision making system requires investing a certain amount of money. If there is a finite amount of money available, then the money budget imposes restrictions on the actions that can be executed during the entire plan execution. If the agent uses the available budget by executing an action, then it decreases the budget that is available to the agent in subsequent steps. This means that the budget cannot be used anymore once it becomes depleted.

Instantaneous constraints The second type of constraint that we consider is the instantaneous constraint, which can be used to model resource constraints that need to be respected during a specific time step. This is different compared to the budget constraint, which models a situation where a constraint holds across multiple time steps. As an example we consider line capacity constraints in a distribution grid, which should be respected at all times. The usage of line capacity at 2 PM does not reduce the capacity of the line at 3 PM, and therefore the capacity constraint should be modeled using multiple instantaneous constraints that are dependent on time. Instantaneous constraints can be used for resources that are renewable, which means that current usage of the resource does not affect its availability later in time.

Planning problems which include uncertainty can be modeled using Markov Decision Processes (Puterman, 1994), which provide a mathematical model for encoding environment states, actions of agents and stochastic state transitions. The model supports both exogenous transition uncertainty and endogenous transition uncertainty. A Partially Observable Markov Decision Process extends a Markov Decision Process with state uncertainty, in which an agent needs to infer information about the environment state based on the observations it receives. Both models have been studied extensively in artificial intelligence literature, and the models have been applied in several domains for solving planning problems which include uncertainty. Furthermore, the models have been extended with additional constraints in order to account for limited availability of resources during planning (Altman, 1999; Isom, Meyn, and Braatz, 2008). In some sense planning with constraints can be seen as planning with multiple objectives, in which the first objective is related to the task performance, and the second objective is related to a constraint that needs to be respected. However, in this dissertation we do not view this problem as a multi-objective optimization problem, and we only focus on the integration of constraints in planning algorithms.

1.4. Contributions of the dissertation

Based on the practical and theoretical motivations in the previous sections, we can formulate the main research goal of the dissertation as follows:

Advancing the state of the art in constrained multi-agent planning under uncertainty, and thereby improving the applicability of AI-based planning in domains such as smart distribution grids with resource constraints.

The current state of the art in planning can be used for constrained planning in uncertain environments, but there are several practical aspects which currently prevent the application of planning in applications such as smart distribution grids. This dissertation aims to present specific advancements which bring us closer to constrained planning under uncertainty in these real-world applications. To be more specific, we achieve the main research goal by focusing on the following open research challenges:

1. Exact planning algorithms for problems with state uncertainty are computationally demanding. Computing optimal solutions is intractable in many domains, which means that it represents an open challenge that requires additional research.

2. Planning problems in smart distribution grids typically involve a finite time horizon. The state of the art in approximate planning under uncertainty typically considers an infinite horizon. Existing methods cannot be applied to solve finite-horizon problems, because they assume that there is an infinite horizon with discounting of reward. This means that finite-horizon problems require new tailored algorithms.

3. Planning algorithms for problems with state uncertainty have been extended with constraints, but the resulting algorithms have limited scalability. Furthermore, the algorithms do not support multiple agents, which is required

in domains such as smart distribution grids. Solving multi-agent planning problems with constraints and state uncertainty efficiently is an open research challenge.

4. Algorithms for constrained planning under uncertainty consider multiple constraints, but adapting them to a specific application requires additional work. In smart distribution grids this is an open problem, since it is unclear how power grid constraints translate to resource constraints supported by the planning algorithms. Furthermore, the type of constraints considered by planning algorithms does not match the constraints that are required in smart distribution grids.

We advance the state of the art in constrained multi-agent planning under uncertainty by presenting algorithmic techniques which address the research challenges that we have identified. These techniques also make the applicability in domains such as smart distribution grids closer to reality. As a result, we obtain a collection of algorithmic techniques which achieve the main research goal of the dissertation. Below we provide a more detailed overview of our individual contributions, which describes how this dissertation addresses the research challenges.

Accelerated exact value iteration for POMDPs For research challenge 1 we focus on exact value iteration, which is an algorithm that can be used to compute an optimal POMDP solution. The existing state of the art is the incremental pruning algorithm (Cassandra, Littman, and Zhang, 1997). This algorithm computes a large number of so-called alpha vectors that represent a value function, and subsequently it executes a pruning subroutine which discards the vectors that are dominated by others. The traditional pruning subroutine relies on solving many linear programs, and we show that the running time of solving these linear programs can be reduced by applying a constraint generation scheme. As a result, we obtain the fastest exact pruning-based value iteration algorithm for POMDPs.

Approximate algorithm for finite-horizon POMDP planning For research challenge 2 we consider approximate algorithms for finite-horizon POMDPs. We first argue why existing POMDP algorithms are not suitable for solving these problems effectively. The main contribution of the chapter is FiVI, a point-based value iteration algorithm for solving finite-horizon POMDPs. FiVI unifies multiple ideas from existing point-based value iteration algorithms for infinite-horizon POMDPs. Furthermore, it contains two strategies to enhance the efficiency of point-based backups and the efficiency of value upper bound updates. A series of experiments shows that FiVI is an effective method for solving finite-horizon POMDPs. FiVI is also used in our new algorithm for Constrained POMDPs, which we discuss next.

	Unconstrained	**Constrained**
MDP		Congestion management (Ch. 6)
POMDP	Exact VI (Ch. 3), FiVI (Ch. 4)	Column generation (Ch. 5)

Table 1.1: Overview of the contributions in the research field

Approximate algorithm for Constrained POMDP planning To address the third research challenge we present a novel approximate algorithm for solving Constrained POMDPs, which is fundamentally different compared to existing algorithms in the literature. Until now, research on Constrained POMDPs has focused on two types of methods. The first type of methods adds additional constraints to traditional unconstrained POMDP algorithms. The second type augments algorithms for Constrained MDPs with partial observability. We propose a new type of solution algorithms, which enables us to solve a Constrained POMDP as a sequence of unconstrained POMDPs. Our algorithm is based on column generation for linear programming and it has shown to outperform the current state of the art. Furthermore, it is the first algorithm that supports multi-agent variants of this problem, in which multiple independent agents in a partially observable environment share global constraints.

Planning algorithm for congestion management in smart grids For research challenge 4 we consider a practical application of constrained planning under uncertainty which is directly relevant for the development of smart distribution grids in our society. Distribution grids face significant changes in the upcoming years due to increased demand (e.g., electric vehicles) and uncertain production from renewables. These developments increase the risk of grid congestion, and it makes it more difficult to control demand and supply. We show that Constrained MDPs can be used to build a congestion management scheme which takes sources of uncertainty into account. To this end, we show how realistic power grid constraints can be integrated in Constrained MDPs, and we present methods to ensure that these constraints are respected during policy execution. Experiments based on a realistic IEEE distribution grid demonstrate the effectiveness of our approach. From a more general point of view, our results show that constrained planning under uncertainty can be potentially used to address problems our society is facing in the near future.

Our individual contributions address the research challenges and contribute to achieving the main research goal. From a more general point of view, the individual contributions of this dissertation improve the state of the art in multiple areas of the sequential decision making under uncertainty research field. In Table 1.1 this field is visualized based on two criteria: presence of state uncertainty and presence of additional constraints. For each individual contribution we indicate where it can be positioned within the field, which shows that we cover constrained decision making problems for both Markov Decision Processes and Partially Observable Markov Decision Processes. For unconstrained planning problems we present two methods for planning problems with partial observability. It is important to note that one of these methods is exact while the other method is approximate.

1.5. Dissertation overview

The structure of this dissertation is based on the individual contributions listed in the previous section. In Chapter 2 we start with an overview of background material. In Chapter 3 we describe techniques for accelerating exact value iteration. In Chapter 4 we present the FiVI algorithm for finite-horizon planning. In Chapter 5 we describe our approximate algorithm for Constrained POMDPs. In Chapter 6 we focus on constrained planning under uncertainty in smart distribution grids. We summarize our contributions in Chapter 7, which also provides an overview of research directions that can be expanded in the future.

2

Planning under uncertainty

The field of planning under uncertainty provides well-grounded models and algorithms for making sequential decisions in uncertain environments. In particular, Markov Decision Processes (MDPs) enable modeling of agents which fully observe their surroundings, and Partially Observable Markov Decision Processes (POMDPs) enhance this model with the ability to reason about imperfect information. In this chapter we formally introduce both models, as well as commonly used solution algorithms and model extensions.

2.1. Markov Decision Processes

Markov Decision Processes (Puterman, 1994) provide a mathematical framework for modeling sequential decision making problems which involve uncertainty. It models an agent that interacts with an uncertain environment by executing actions sequentially, in such a way that it performs well on a given task. For example, in the context of robotics one can model a robot which needs to reach a designated goal while reasoning about uncertain outcomes of the actions. The planning problem consists of finding a conditional action sequence which performs as well as possible.

2.1.1. States, actions, rewards and policies

In this dissertation we mostly focus on problems with a finite planning horizon, in which an agent executes a predefined number of actions. Finite-horizon problems differ slightly from the traditional infinite-horizon MDP model that is typically introduced in academic literature. In order to facilitate a general introduction, we first describe the basic components of the model. In the next two sections we discuss specific concepts for infinite-horizon problems and finite-horizon problems.

Formally, an MDP M is defined using a tuple $M = \langle S, A, T, R, s_1 \rangle$. The set S contains all possible environment states, and the set A contains all possible actions that can be executed by the agent. Our description focuses on problems with a finite number of states and actions. State descriptions can be factored such that a state is defined by multiple separate state variables (Guestrin et al., 2003), but unless stated otherwise it is assumed that we are dealing with flat representations. The state $s_1 \in S$ denotes the initial state of the system. The state transitions of the environment are represented by the transition function $T : S \times A \times S \rightarrow [0,1]$. When executing action $a \in A$ in state $s \in S$, then the environment state changes stochastically to state s' with probability $T(s,a,s') = P(s'|s,a)$. The Markov property entails that the stochastic transition to a successor state only depends on the current state, and not on states encountered in the past. In the context of a robot navigation problem states can represent the current location of the robot in a grid, and actions would correspond to the directions in which the robot can move. If there is uncertainty associated with these moves, then the transition function defines the uncertain nature of the action outcomes.

The reward function $R : S \times A \rightarrow \mathbb{R}$ implicitly defines the goal to be reached or the task to be fulfilled. To be more specific, when the agent executes action $a \in A$ in state $s \in S$, it receives a reward $R(s,a)$. The reward function can also be defined as $R : S \times A \times S \rightarrow \mathbb{R}$, where rewards are also dependent on the successor state. Both representations can be used interchangeably since they can be easily converted in the other form[1], but in the remainder of this dissertation we refer to the former. Typically we are interested in maximizing the reward collected over time. For example, in the context of the robot navigation problem one can think about a positive reward for moving in the direction of the goal state. It should be noted, however, that the reward function also allows for modeling of the cost to be minimized, by treating them as negative rewards.

Figure 2.1 visualizes the agent that interacts with the environment. The environment state s transitions to state s' after executing action a, and the agent perceives both the new state s' and the reward $R(s,a)$. This can be repeated multiple times.

For infinite-horizon MDPs a solution is typically expressed as a deterministic stationary policy $\pi : S \rightarrow A$, which defines the action $\pi(s)$ to be executed in each state s. For finite-horizon problems the policy becomes non-stationary and hence it can be dependent on time. In some cases we use a stochastic policy $\pi : S \times A \rightarrow [0,1]$, which defines the probability $\pi(s,a)$ to execute action a in state s. Policies are defined in such a way that the agent maximizes the reward it collects over time. More details about optimality criteria and the computation of policies are

[1]Under the expected reward optimality criterion, as considered in this dissertation, we can take a weighted average over successor states s' to compute the expected immediate reward $R(s,a)$ after executing a.

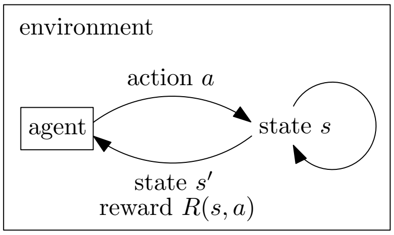


Figure 2.1: MDP agent interacting with the environment

presented in the next two sections for both infinite-horizon problems and finite-horizon problems.

2.1.2. Infinite-horizon problems

MDPs with an infinite horizon assume that the agent maximizes the reward collected in an infinite number of steps, where reward collected in the future is discounted according to a discount factor $0 \leq \gamma < 1$. Formally, this optimality criterion can be stated as follows:

$$E\left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t\right], \tag{2.1}$$

where r_t represents the reward collected at time t. Note that the discount factor γ ensures that the sum becomes finite, and it ensures that rewards early reward contributes more than reward received much later in time.

Given the expected reward optimality criterion, the quality of a policy π can be quantified using a value function $V^{\pi} : S \rightarrow \mathbb{R}$. The value $V^{\pi}(s)$ is defined as the expected reward collected by the agent when executing policy π starting from state s:

$$V^{\pi}(s) = E\left[\sum_{k=1}^{\infty} \gamma^{k-1} r_{t+k} \,\middle|\, s_t = s\right]. \tag{2.2}$$

It can be shown that we can recursively define the equation in terms of a Bellman equation (Bellman, 1957):

$$V^{\pi}(s) = \sum_{s' \in S} P(s'|s, \pi(s))(R(s, \pi(s)) + \gamma V^{\pi}(s')), \tag{2.3}$$

which defines the value as the sum of the immediate reward and the discounted future reward collected in all successor states, weighted by their probability.

Solving an MDP corresponds to finding the optimal policy π^*, such that $V^{\pi^*}(s) \geq V^{\pi}(s)$ for each $s \in S$ and for all policies π. The value function of the optimal policy π^* satisfies the Bellman optimality equation:

$$V^{\pi^*}(s) = \max_{a \in A} \sum_{s' \in S} P(s'|s, a)\big(R(s, a) + \gamma V^{\pi^*}(s')\big). \tag{2.4}$$

The optimal policy π^* can be expressed as a function of this value function:

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} P(s'|s,a)\big(R(s,a) + \gamma V^{\pi^*}(s')\big), \tag{2.5}$$

which takes the action which provides the maximum expected reward.

Numerous algorithms exist to compute an optimal value function. The value iteration algorithm (Bellman, 1957) initializes an initial value function V_0 and iteratively generates new value functions V_{k+1} from value function V_k until convergence. It initializes $V_0(s)$ to zero for each state s, and subsequently it uses the following equation to generate a sequence of value functions:

$$V_{k+1}(s) = \max_{a \in A} \sum_{s' \in S} P(s'|s,a)(R(s,a) + \gamma V_k(s')). \tag{2.6}$$

This process is known to converge in the limit to the optimal value function V_{π^*} defined in Equation 2.4, which also defines the optimal policy π^*. It is common to terminate value iteration if the Bellman error magnitude $\max_{s \in S} |V_{k+1}(s) - V_k(s)|$ drops below a given tolerance ε.

An alternative to value iteration is the policy iteration algorithm (Howard, 1960), which operates in the space of policies rather than the space of value functions. Policy iteration repeatedly executes a policy evaluation step, followed by a policy improvement step. It initializes an arbitrary policy π, which it evaluates by computing its value function V^π as defined in Equation 2.3 (e.g., by solving a linear constraint system). Subsequently, it updates the current policy π based on V^π, using an update rule almost identical to Equation 2.5. This repeats until convergence.

A third methodology for solving MDPs is based on linear programming. Since linear programs are widely used throughout this dissertation, we provide a separate description in Section 2.1.4.

2.1.3. Finite-horizon problems

Finite-horizon MDPs are used for problems in which an agent collects reward in a finite number of steps. The solution concepts for finite-horizon MDPs are similar to the infinite-horizon case, and therefore we only present the most important equations. The optimality criterion for finite-horizon problems is be defined as:

$$E\left[\sum_{t=1}^{h} r_t\right], \tag{2.7}$$

where h is a parameter defining the planning horizon. Optimal policies should maximize the total expected reward received during execution, and hence a discount factor is not required. On purpose we start counting time steps at 1, because this means that finite-horizon formulations consider h steps in total.

In finite-horizon problems the current time step t has influence on the decisions made by the agent. For example, if there are 10 steps left until the end of the horizon then the action to be executed may be different compared to the action that would be chosen in case there are only 3 steps left. In the first situation it may choose a risky action because there is sufficient time to recover, whereas a more conservative action would be appropriate near the end of the horizon. More formally, policies for finite-horizon problems are called non-stationary and they depend on time.

The optimal value functions V^* for finite-horizon problems are almost identical to the infinite horizon case, and they can be defined as follows:

$$V^*(t,s) = \begin{cases} \max_{a \in A} \sum_{s' \in S} P(s'|s,a)(R(s,a) + V^*(t+1,s')) & t < h \\ \max_{a \in A} R(s,a) & t = h \end{cases}. \tag{2.8}$$

For each time step t the corresponding non-stationary policy π_t^* is defined similarly:

$$\pi^*(t,s) = \begin{cases} \arg\max_{a \in A} \sum_{s' \in S} P(s'|s,a)(R(s,a) + V^*(t+1,s')) & t < h \\ \arg\max_{a \in A} R(s,a) & t = h \end{cases}. \tag{2.9}$$

Note that optimal value functions V^* and policies π^* in the finite-horizon case can be computed using one single dynamic programming pass from the end of the horizon until the first step.

Besides the aforementioned infinite-horizon optimality criterion and the finite-horizon optimality criterion, there are other characterizations which define whether a policy is optimal or not. For example, the average reward criterion defines that the expected average reward collected during execution should be maximized, rather than the expected sum. This can be defined as:

$$\limsup_{h \to \infty} E\left[(1/h) \cdot \sum_{t=1}^{h} r_t\right], \tag{2.10}$$

in which the superior assures that the limit exists. Similar to the finite-horizon criterion, for a specific policy it cannot be distinguished whether an agent receives high reward during early stages of execution, because all rewards collected over time are treated similarly without having a notion of discounting. We do not further consider the average reward criterion in this dissertation.

2.1.4. Linear programming formulations

The notion of value functions and policies, as described in the previous sections, can be used to obtain MDP policies using value iteration or policy iteration. A third methodology for solving MDPs is based on linear programming, which captures the underlying ideas of value functions in one LP formulation that can be solved using standard optimization algorithms for LPs. An additional advantage of such formulations is that it allows for addition of constraints on the optimal policy, which becomes relevant in subsequent chapters. In the remainder of this section we provide a basic introduction to commonly used LP formulations.

The first formulation applies to infinite-horizon problems and treats the optimal values $V^{\pi^*}(s)$ as LP decision variables:

$$\begin{aligned} \min \; & \sum_{s \in S} V^{\pi^*}(s) \\ \text{s.t. } & V^{\pi^*}(s) \geq \sum_{s' \in S} P(s' | s, a) \big(R(s,a) + \gamma V^{\pi^*}(s') \big) \quad \forall s \in S, a \in A. \end{aligned} \tag{2.11}$$

Solving the LP gives the optimal value for each state, from which the optimal policy can be easily derived. Another interesting observation is that LPs can be solved in polynomial time (Khachiyan, 1980), and therefore we know that MDPs can be solved to optimality in polynomial time.

In some cases it is convenient to dualize the LP shown in (2.11), because the resulting (equivalent) formulation provides a different characterization of the optimal policy and allows for adding constraints. The formulation is as follows:

$$\begin{aligned} \max \; & \sum_{s \in S} \sum_{a \in A} R(s,a) \cdot x_{s,a} & & \\ \text{s.t. } & \sum_{a' \in A} x_{s',a'} - \gamma \sum_{s \in S} \sum_{a \in A} x_{s,a} \times P(s' | s, a) = P(s_1 = s') & & \forall s' \in S \\ & x_{s,a} \geq 0 & & \forall s \in S, a \in A \end{aligned} \tag{2.12}$$

where the decision variable $x_{s,a}$ denotes the (discounted) occupancy frequency for the state-action pair (s,a), and the term $P(s_1 = s)$ denotes the probability that the initial state is s. This term is equal to 1 if $s = s_1$, and equals 0 otherwise. The resulting optimal policy π^* is defined as:

$$\pi^*(s,a) = \frac{x_{s,a}}{\sum_{a' \in A} x_{s,a'}}, \tag{2.13}$$

which defines a probability distribution over actions for each state. Even though the formulation in (2.12) defines potentially stochastic policies, the optimal policy is fully deterministic. This can be explained by observing that the linear program has $|S|$ constraints, and hence at most $|S|$ variables $x_{s,a}$ will become positive in an optimal solution (Papadimitriou and Steiglitz, 1982).

The LP formulation for finite-horizon problems follows a similar line of reasoning, for which we only provide the dual formulation below.

$$\begin{aligned}
\max \; & \sum_{t=1}^{h} \sum_{s \in S} \sum_{a \in A} R(s,a) \cdot x_{t,s,a} & \\
\text{s.t.} \; & \sum_{a' \in A} x_{t+1,s',a'} = \sum_{s \in S} \sum_{a \in A} x_{t,s,a} \cdot P(s'|s,a) & \forall s' \in S, t \in \{1,\dots,h\} \\
& \sum_{a \in A} x_{0,s,a} = P(s_1 = s) & \forall s \in S \\
& 0 \leq x_{t,s,a} \leq 1 & \forall s \in S, a \in A, t \in \{1,\dots,h\}
\end{aligned} \tag{2.14}$$

In this formulation the variable $x_{t,s,a}$ denotes the probability that the agent encounters state s at time t and subsequently executes action a. The resulting optimal stochastic non-stationary policy π^* is defined as:

$$\pi^*(t,s,a) = \frac{x_{t,s,a}}{\sum_{a' \in A} x_{t,s,a'}}, \tag{2.15}$$

which defines the probability to execute a in state s at time t. Although not mentioned explicitly in this section, in finite-horizon formulations the rewards and transition probabilities can be made time-dependent as well.

2.1.5. Constrained problems

As discussed in the introductory chapter of this dissertation, in some application domains it is required that solutions to planning problems consider constraints. Constrained Markov Decision Processes (CMDPs) augment the standard MDP model with additional constraints (Altman, 1999). Besides the reward function it defines one or more cost functions $C : S \times A \rightarrow \mathbb{R}$ which specify the cost $C(s,a)$ for executing action a in state s. For example, the cost can be defined as the usage of a resource when executing an action in a certain state. A cost function can be used to create problems where an optimal policy should maximize the expected reward while the expected cost incurred during policy execution is upper-bounded by L.

For infinite-horizon problems the following constraint can be easily added to the dual LP formulation shown in (2.12):

$$\sum_{s \in S} \sum_{a \in A} C(s,a) \cdot x_{s,a} \leq L, \tag{2.16}$$

which states that the expected discounted cost should be upper-bounded by L. However, it should be noted that this is only useful in domains where bounding expected *discounted* cost is meaningful. Unfortunately, in several domains this is not the case, which we will illustrate with an example inspired by the domain sketched in the introduction chapter. If multiple agents are connected to a power grid then it can be useful to optimize for short-term reward by using discounting.

However, if the agents have to respect power grid constraints, then constraints on discounted power flows are not meaningful because the constraints at all time steps are equally important. In such domains it is more intuitive to bound the expected (undiscounted) cost, which can be integrated in the LP formulation shown in (2.14) for finite-horizon problems:

$$\sum_{t=1}^{h} \sum_{s \in S} \sum_{a \in A} C(s,a) \cdot x_{t,s,a} \leq L. \tag{2.17}$$

Constrained MDPs resemble problems with multiple objectives (Roijers et al., 2013), because the reward and cost function can be interpreted as multiple objectives which are taken into account when computing an optimal policy. However, it is important to note that in CMDPs it is only required to bound expected cost, rather than optimizing for this additional cost.

2.1.6. Planning for multiple agents

The standard MDP model can be used to model interactions between an agent and its environment. However, in many domains it is relevant to model n agents which sequentially execute actions to achieve an individual or collaborative goal. The Multi-agent MDP (MMDP) model (Boutilier, 1996) assumes that each agent i has an individual state space S_i and action space A_i, which are subsequently merged into a joint state space $S = S_1 \times \ldots \times S_n$ and joint action space $A = A_1 \times \ldots \times A_n$. The transition function $T : S \times A \times S \rightarrow [0,1]$ is defined over joint states and actions. The reward function $R : S \times A \rightarrow \mathbb{R}$ makes it possible to define rewards over the joint state and action space, and therefore this model is appropriate for settings where multiple agents collaborate to reach a common goal. Since the MMDP formulation reduces a multi-agent problem to a single-agent MDP, the standard algorithms still apply. However, the resulting state and action space grow exponentially in the number of agents, and therefore it is not tractable to use for larger problems.

Multi-agent problems involving global constraints on all policies can be defined by combining the concepts of MMDPs and CMDPs, but such an approach would still be affected by the aforementioned scalability problems. If there are no dependencies between the individual MDP models of the agents, and if the joint reward function is just the sum of individual reward functions $R_i : S_i \times A_i \rightarrow \mathbb{R}$ rather than a reward function defined over the joint state space S, then more efficient solutions approaches exist. In such situations there is no need to merge the individual models into a joint model, and hence policies can be optimized individually while accounting for the global constraints. Such models and algorithms will be further described in Chapters 5 and 6.

An additional formalism for multi-agent planning is the Decentralized POMDP model, in which multiple agents collaborate in the same environment while making decisions in a decentralized fashion without direct communiction (Oliehoek and Amato, 2016). The model is fundamentally different compared to the MMDP formalism, in which decisions are made centrally based on the joint state of the agents. Decentralized POMDPs are not considered in this dissertation. However, we want to emphasize that our algorithms for constrained multi-agent planning in Chapters 5 and 6 provide solutions that can be executed in a decentralized fashion without communication.

2.2. Partial observability

The Markov Decision Process framework described in the previous section can be used to model decision making problems which include uncertainty. It is assumed that the agent interacts with the environment by executing actions, and from the viewpoint of the agent the current environment state is always known. In several domains, however, it cannot be assumed that the state of the environment is known with full certainty. For example, a robot may have sensors which do not provide perfect information about the surrounding area in which the robot navigates. Another example occurs in maintenance problems where a decision maker needs to decide when to perform maintenance on paved roads. The decision maker knows that the condition of the roads deteriorates over time, but without performing inspections and initial maintenance there is no full certainty about the actual condition. In both examples the current state is *partially observable* and the agent needs to reason about the true state. In this section we introduce Partially Observable Markov Decision Processes (Kaelbling, Littman, and Cassandra, 1998; Spaan, 2012), which naturally model such sequential decision making problems.

2.2.1. Augmenting MDPs with observations

Partially Observable Markov Decision Processes (POMDPs) extend fully observable MDPs with an observation model describing a probabilistic relationship between the environment state and the observations made by the agent. Instead of observing the state directly, the agent perceives an observation according to this observation model. Based on observations made it can infer information about the actual state.

Formally, a POMDP is defined as a tuple $M = \langle S, A, O, T, \Omega, R, b_1 \rangle$, in which S, A and T are the set of states, the set of actions and the transition function, identical to the MDP definition. O represents a set containing a finite number of observations. The observation function $\Omega : A \times S \times O \rightarrow [0,1]$ defines the probability to make an observation, depending on the executed action and the environment state after executing that action. If the agent executes action $a \in A$ and the environment state transitions to $s' \in S$, then the agent observes $o \in O$ with probability $\Omega(a, s', o) = P(o|a, s')$. The probability to observe o is dependent on the successor state s', but it should be emphasized that the agent never receives explicit information about s'

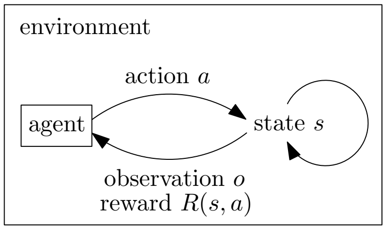


Figure 2.2: POMDP agent interacting with the environment

while interacting with the environment.

Similar to MDPs, we can visualize the interaction between the agent and its environment. Figure 2.2 shows an agent that executes action a, after which it receives observation o and it gets the reward $R(s,a)$. Here it is important to note that it does not receive information about state s itself. More details about specific horizons and discounting will be discussed in Section 2.2.3 and Section 2.2.4.

2.2.2. Belief states and belief updates

In fully observable MDPs the environment state provides a Markovian signal based on which the agent can make optimal decisions. However, in POMDPs the sequence of observations does not provide sufficient information to make optimal decisions. All executed actions and observations encountered in the past can affect the knowledge the agent has about the current state, and hence a notion of memory is necessary to define an optimal decision making policy.

For POMDPs a Markovian planning signal can be defined using belief states b rather than actual states s. A belief state b is a vector of length $|S|$ defining the probability $b(s)$ that the current environment state is s. In other words, the vector characterizes the current belief of the agent regarding the actual environment state. A belief state is a sufficient statistic for the full history of actions and observations, and therefore there are no other representations which provide the agent with more information about the history. In a POMDP it is assumed that the agent has an initial belief b_1. If the agent has no initial knowledge about the state, this belief would correspond to a uniform distribution over states.

While interacting with the environment the agent updates its belief b. After executing action a and receiving observation o, the resulting belief b_a^o is defined using Bayes' rule:

$$b_a^o(s') = \frac{P(o|a,s')}{P(o|b,a)} \sum_{s \in S} P(s'|s,a)b(s), \tag{2.18}$$

where $P(o|b,a)$ corresponds to the probability to observe o after executing action a

in belief b. This probability is calculated as follows:

$$P(o|b,a) = \sum_{s' \in S} P(o|a,s') \sum_{s \in S} P(s'|s,a)b(s), \tag{2.19}$$

and in the belief update equation this term serves as a normalizing constant.

2.2.3. Infinite-horizon problems

Similar to MDPs, we can distinguish POMDPs with an infinite horizon and a finite horizon. In this section we first describe infinite-horizon POMDPs based on a discount factor $0 \leq \gamma < 1$. Conceptually it is almost identical to the solution concepts for MDPs, and therefore we only provide a brief introduction.

The solution of an infinite-horizon POMDP is a policy $\pi : \Delta(S) \rightarrow A$ mapping beliefs to actions, in which $\Delta(S)$ denotes the continuous set of probability distributions over S. Similar to infinite-horizon MDPs, the aim is to maximize the expected sum of discounted rewards. For a given policy π the expected discounted reward $V^{\pi}(b)$ collected when executing π starting from b is defined as:

$$V^{\pi}(b) = E_{\pi}\left[\sum_{k=1}^{\infty} \gamma^{k-1} R(b_k, \pi(b_k)) \,\middle|\, b_1 = b\right], \tag{2.20}$$

where $R(b_t, \pi(b_t)) = \sum_{s \in S} R(s, \pi(b_t))b_t(s)$ denotes the expected reward when executing $\pi(b_t)$ in belief b_t.

For the optimal policy π^* it holds that $V^{\pi^*}(b) \geq V^{\pi}(b)$ for each $b \in \Delta(S)$ and for all policies π. Similar to MDPs it satisfies the Bellman optimality equation:

$$V^{\pi^*}(b) = \max_{a \in A}\left[\sum_{s \in S} R(s,a)b(s) + \gamma \sum_{o \in O} P(o|b,a)V^{\pi^*}(b_a^o)\right]. \tag{2.21}$$

The optimal policy π^* corresponding to this value function is defined as:

$$\pi^*(b) = \arg\max_{a \in A}\left[\sum_{s \in S} R(s,a)b(s) + \gamma \sum_{o \in O} P(o|b,a)V^{\pi^*}(b_a^o)\right]. \tag{2.22}$$

The value functions introduced in this section provide a conceptual characterization of an optimal value function and the corresponding optimal policy. In Section 2.2.5 we discuss a convenient technique to represent these value functions in memory. Moreover, we discuss techniques to compute such value functions in Section 2.2.6 and Section 2.2.8.

2.2.4. Finite-horizon problems

Finite-horizon POMDPs include a parameter h which represents the time horizon, such that the agent executes actions in time steps $1, \ldots, h$, and execution ends at time step $h+1$. The solution to a finite-horizon POMDP is a time-dependent policy $\pi : \{1,\dots,h\} \times \Delta(S) \rightarrow A$, which maps beliefs and time steps to actions, and it maximizes the expected sum of rewards received by the agent. A policy can be seen as a plan which enables the agent to perform its task in the best possible way, and its quality can be evaluated using a value function $V^{\pi} : \{1,\dots,h\} \times \Delta(S) \rightarrow \mathbb{R}$. The value $V^{\pi}(t,b)$ denotes the expected sum of rewards that the agent receives when following policy π starting from belief b at time t, and it is defined as:

$$V^{\pi}(t,b) = E_{\pi}\left[\sum_{t'=t}^{h} R(b_{t'},\pi(t',b_{t'})) \,\middle|\, b_t = b \right], \tag{2.23}$$

where $b_{t'}$ is the belief at time t' and $R(b_{t'},\pi(t',b_{t'})) = \sum_{s\in S} R(s,\pi(t',b_{t'}))b_{t'}(s)$. For an optimal policy π^* it holds that it always achieves the highest possible expected reward during execution. Formally, it holds that $V^{\pi^*}(1,b) \geq V^{\pi}(1,b)$ for each belief b and for each possible policy π. The optimal value function $V^{\pi^*}(t,b) = \max_{\pi} V^{\pi}(t,b)$ is defined by the following recurrence:

$$V^{\pi^*}(t,b) = \begin{cases} \max_{a\in A}\left[\sum_{s\in S} R(s,a)b(s) + \sum_{o\in O} P(o|b,a)V^{\pi^*}(t+1,b_a^o)\right] & t \leq h \\ 0 & \text{otherwise} \end{cases} \tag{2.24}$$

The optimal policy π^* corresponding to the optimal value function is defined as:

$$\pi^*(t,b) = \arg\max_{a\in A}\left[\sum_{s\in S} R(s,a)b(s) + \sum_{o\in O} P(o|b,a)V^{\pi^*}(t+1,b_a^o) \right], \tag{2.25}$$

for $1 \leq t \leq h$. It returns the value-maximizing action for a time step and belief.

Similar to MDPs, both finite-horizon and infinite-horizon POMDPs can be generalized to multiple agents, and the models can be used to solve planning problems which involve constraints. For the purpose of readability we discuss this separately in Chapter 5, in which we present algorithms for Constrained Multi-agent POMDPs.

2.2.5. Vector-based value functions and backups

The value functions in the previous sections have been defined over the continuous belief space. When computing value functions this can be inconvenient, because it requires function representations as well as function manipulations defined over a continuous space. Fortunately, it has been shown that POMDP value functions have a special shape which allows for more efficient representations.

It turns out that value functions for finite-horizon POMDPs are piecewise linear and convex (Sondik, 1971). This means that the value function can be represented using a finite set of $|S|$-dimensional vectors. This also applies to infinite-horizon problems with discounting, because the discount factor γ implicitly defines an upper bound on the number of time steps that is relevant to consider. A value function V can be represented as a set of vectors $\alpha \in V$, such that

$$V(b) = \max_{\alpha \in V} b \cdot \alpha, \tag{2.26}$$

where $\cdot$ denotes the inner product. In this representation V refers to a set of vectors, and $V(b)$ denotes the function value computed using b and the set of vectors.

Value iteration for POMDPs executes dynamic programming stages based on Equation 2.21, in which each stage accounts for one additional time step. If the agent executes only one action, then we can define the initial value function $V_0(b)$ as follows:

$$V_0(b) = \max_{a \in A} \left[\sum_s R(s,a) b(s) \right] = \max_{\{\alpha_0^a\}_{a \in A}} \alpha_0^a \cdot b, \tag{2.27}$$

where $\alpha_0^a(s) = R(s,a)$ denotes a vector containing the immediate rewards. Hence, we can define this value function in terms of vectors as $V_0 = \{\alpha_0^a \mid a \in A\}$.

Given a value function V_n, value iteration algorithms aim to compute the value function V_{n+1} using the Bellman equation. We can abbreviate this as $V_{n+1} = HV_n$, in which H denotes the Bellman backup operator. For convenience we let $\alpha_n^b = \arg\max_{\alpha \in V_n} b \cdot \alpha$ denote the value-maximizing vector from the set V_n in belief b. Computing all vectors belonging to V_{n+1} seems computationally difficult, but given V_n and a belief b we can easily compute the vector α_{n+1}^b such that $\alpha_{n+1}^b = \arg\max_{\alpha \in V_{n+1}} b \cdot \alpha$, where V_{n+1} is the unknown set of vectors representing HV_n. We refer to this operation as executing a backup on belief b:

$$\alpha_{n+1}^b = \texttt{backup}(b), \tag{2.28}$$

such that $V_{n+1}(b) = b \cdot \texttt{backup}(b)$. It is important to observe that this vector represents the gradient of the value function V_{n+1} in belief b.

We can derive the computation of $\texttt{backup}(b)$ directly from the Bellman optimality equation. For convenience we first define

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

as the backprojection of a vector $\alpha_n \in V_n$ based on action a and observation o. The full derivation for the infinite-horizon case now proceeds as follows:

$$V_{n+1}(b) = \max_{a \in A} \left[b \cdot \alpha_0^a + \gamma \sum_{o \in O} P(o|b,a) V_n(b_a^o) \right] \tag{2.30}$$

$$= \max_{a \in A} \left[b \cdot \alpha_0^a + \gamma \sum_{o \in O} P(o|b,a) \max_{\alpha_n \in V_n} \left(\sum_{s' \in S} b_a^o(s') \alpha_n(s') \right) \right] \tag{2.31}$$

$$= \max_{a\in A}\left[b\cdot\alpha_0^a + \gamma\sum_{o\in O}\max_{\alpha_n\in V_n}\sum_{s'\in S}P(o|a,s')\sum_{s\in S}P(s'|s,a)b(s)\alpha_n(s')\right] \quad (2.32)$$

$$= \max_{a\in A}\left[b\cdot\alpha_0^a + \gamma\sum_{o\in O}\max_{\alpha_n\in V_n}\sum_{s\in S}b(s)\sum_{s'\in S}P(o|a,s')P(s'|s,a)\alpha_n(s')\right] \quad (2.33)$$

$$= \max_{a\in A}\left[b\cdot\alpha_0^a + \gamma\sum_{o\in O}\max_{\{g_{ao}^{\alpha_n}\}_{\alpha_n\in V_n}} b\cdot g_{ao}^{\alpha_n}\right] \quad (2.34)$$

$$= \max_{a\in A}\left[b\cdot\alpha_0^a + \gamma\sum_{o\in O}b\cdot\operatorname*{argmax}_{\{g_{ao}^{\alpha_n}\}_{\alpha_n\in V_n}} b\cdot g_{ao}^{\alpha_n}\right] \quad (2.35)$$

$$= \max_{a\in A}\left[b\cdot\alpha_0^a + \gamma b\cdot\sum_{o\in O}\operatorname*{argmax}_{\{g_{ao}^{\alpha_n}\}_{\alpha_n\in V_n}} b\cdot g_{ao}^{\alpha_n}\right] \quad (2.36)$$

$$= \max_{a\in A}\left[b\cdot\left(\alpha_0^a + \gamma\sum_{o\in O}\operatorname*{argmax}_{\{g_{ao}^{\alpha_n}\}_{\alpha_n\in V_n}} b\cdot g_{ao}^{\alpha_n}\right)\right] \quad (2.37)$$

$$= \max_{a\in A}\left[b\cdot g_a^b\right] \quad (2.38)$$

$$= b\cdot\operatorname*{argmax}_{a\in A}\left[b\cdot g_a^b\right] \quad (2.39)$$

with

$$g_a^b = \alpha_0^a + \gamma\sum_{o\in O}\operatorname*{argmax}_{\{g_{ao}^{\alpha_n}\}_{\alpha_n\in V_n}} b\cdot g_{ao}^{\alpha_n}. \quad (2.40)$$

Note that we have applied the definition of the belief update, the identity $b\cdot x + b\cdot y = b\cdot(x+y)$ and the identity $\max_\alpha b\cdot\alpha = b\cdot\operatorname{argmax}_\alpha b\cdot\alpha$ in the derivation. Now we can define the backup operator as follows:

$$\texttt{backup}(b) = \operatorname*{argmax}_{a\in A}\left[b\cdot g_a^b\right]. \quad (2.41)$$

This operator is easy to implement and provides the value-maximizing vector $\alpha_{n+1}^b \in V_{n+1}$ in belief b based on the value function V_n and b itself. It is also common to associate the maximizing action a with a vector α, which is denoted by $a(\alpha)$. The derivation for finite-horizon POMDPs is almost identical, which we discuss in Chapter 4. Unless stated otherwise, we only treat the infinite-horizon case in the remaining parts of this chapter.

The backup operator can be used to compute a vector that is optimal in one specific belief. However, this is not sufficient to compute all vectors belonging to V_{n+1}. Next we explain how to find the set of vectors that is optimal for all beliefs.

2.2.6. Exact value iteration

Value iteration for POMDPs repeatedly computes a value function V_{n+1} using the vectors representing the value function V_n from the previous stage. Based

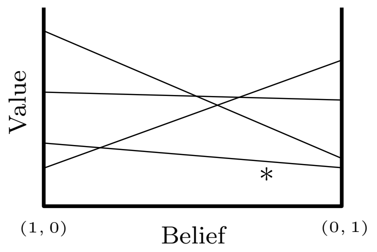


Figure 2.3: Value function with a dominated vector

on the backup operator we can define this computation as $\bigcup_b \texttt{backup}(b)$, but unfortunately this requires knowledge about the beliefs b which are required to compute all vectors belonging to this set. Since there is an infinite number of beliefs, enumeration of beliefs is clearly not possible.

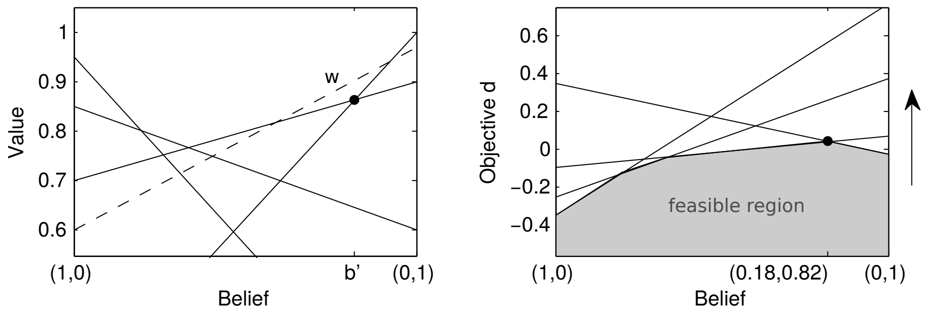
Two types of strategies for computing the vectors of V_{n+1} can be distinguished. The first type iteratively identifies belief points b for which the corresponding vector α_{n+1}^b is value-maximizing in a region of the belief space. Such a belief is typically referred to as witness point, which provides evidence that the corresponding vector is value-maximizing in a non-empty region of the belief space. The second type simply enumerates all possible vectors and runs a pruning procedure that removes vectors for which there is no belief in which it is a value-maximizing vector. In this dissertation we focus on enumeration algorithms because they generally provide better performance and they are more popular in academic literature.

The enumeration algorithm by Monahan (1982) simply enumerates all possible vectors that can be generated by the backup operator:

$$V_{n+1} = \bigcup_{a \in A} G_a \quad \text{with} \quad G_a = \bigoplus\nolimits_{o \in O} G_a^o \quad \text{and} \quad G_a^o = \left\{ \frac{1}{|O|} \alpha_0^a + \gamma g_{ao}^{\alpha_n} \;\middle|\; \alpha_n \in V_n \right\}, \tag{2.42}$$

which produces $|A||V_n|^{|O|}$ vectors in total. The operator $\bigoplus$ denotes the cross sum operator. For two sets P and Q the operator can be defined as $P \bigoplus Q = \{p + q \mid p \in P, q \in Q\}$. Many of the computed vectors do not contribute to the value function, which means that there are no belief points for which the vectors are value-maximizing. An example is shown in Figure 2.3 for a POMDP with two states. The vector labeled with the asterix is never a value-maximizing vector, and such vectors can be discarded without changing the value function and its corresponding policy. Moreover, discarding such vectors is important because they influence the computation time required in subsequent dynamic programming stages. Pruning of vectors can be done using a special procedure `prune`:

$$V_{n+1} = \texttt{prune}\left(\bigcup_{a \in A} G_a \right), \tag{2.43}$$



(a) Vectors in U and vector w (dashed)   (b) LP constraints of (U, w)

Figure 2.4: Value function U with vector w and the feasible region of the corresponding LP

which only keeps vectors if they are value-maximizing in a region of the belief space. More details about pruning of vectors will be provided in the next section.

Additional computation time can be saved by pruning vectors incrementally, rather than pruning vectors after enumeration. This insight led to the Incremental Pruning algorithm (Cassandra, Littman, and Zhang, 1997), which computes V_{n+1} as follows:

$$V_{n+1} = \texttt{prune}\left(\bigcup_{a\in A} G_a\right) \quad \text{with} \quad G_a = \texttt{prune}\Big(\texttt{prune}\left(\bar{G}_a^1 \bigoplus \bar{G}_a^2\right) \dots \bigoplus \bar{G}_a^{|O|}\Big), \tag{2.44}$$

in which $\bar{G}_a^o = \texttt{prune}(G_a^o)$. The latter ensures that the operands of the cross sum are pruned before computing the cross sum. Pruning is a crucial component of Incremental Pruning and our algorithms in Chapter 3.

2.2.7. Vector pruning

As discussed in the previous section, dominated vectors can be pruned using a subroutine `prune`. This subroutine takes a vector set as input, and it returns the vector set obtained after pruning dominated vectors from the input set. In the literature this procedure is also known as filtering, and the resulting vector set after pruning is also known as the parsimonious value function. We first explain how it can be checked whether a specific vector is dominated by a set of vectors, after which we describe how this check is integrated in a vector pruning algorithm.

Pruning algorithms repeatedly verify whether there is a region of the belief space in which a given vector $w \notin U$ dominates vectors in a set U. As an example we consider Figure 2.4a, which shows a value function U consisting of 4 vectors, each of which is visualized as a solid line. As can be seen, for each of these vectors there is a region of the belief space in which it is the value-maximizing vector, and the dashed vector w dominates the vectors in U in a small region.

The corner highlighted with the dot corresponds to the belief b' in which the vector w dominates the vectors in the set U the most. In other words, in the belief b' the value function induced by U improves the most when adding the vector w to the set U. Sometimes the belief b' is referred to as a witness point in the literature (Kaelbling, Littman, and Cassandra, 1998), because it can act as a witness which proves that w has the potential to improve the value function induced by U.

In the visual example it is easy to recognize the belief point b' in which w improves the vector set U the most, but in general it requires solving the following optimization problem:

$$\begin{aligned} b' &= \operatorname*{arg\,max}_{b\in\Delta(S)} \left[w\cdot b - \left(\max_{u\in U} u\cdot b \right) \right] && (2.45)\\ &= \operatorname*{arg\,max}_{b\in\Delta(S)} \left[\min_{u\in U} (w\cdot b - u\cdot b) \right] && (2.46)\\ &= \operatorname*{arg\,max}_{b\in\Delta(S)} \left[\min_{u\in U} ((w-u)\cdot b) \right]. && (2.47) \end{aligned}$$

The first term in the maximization problem in Equation 2.45 corresponds to the value defined by the vector w in belief b, and the second term corresponds to the value in belief b defined by the value function induced by the vectors in U. The value function induced by U improves when adding the vector $w \notin U$ if:

$$\max_{b\in\Delta(S)} \left[\min_{u\in U} ((w-u)\cdot b) \right] > 0, \qquad (2.48)$$

which states that the resulting value improvement should be strictly positive.

The optimization problem introduced above finds a belief in the continuous belief simplex $\Delta(S)$, which we can conveniently formulate as a linear program:

$$\begin{aligned} \max\ & d \\ \text{s.t. } & d \le (w-u)\cdot b \quad && \forall u \in U \\ & \sum_{s\in S} b_s = 1 && \\ & b_s \ge 0 && \forall s \in S \\ & d \in \mathbb{R}, && \end{aligned} \qquad (2.49)$$

in which b_s denotes the belief corresponding to state s. For each vector $u \in U$ it defines a constraint of the form $d \le (w-u)\cdot b$, which means that the objective d is upper-bounded by $(w-u)\cdot b$, identical to Equation 2.47. For the example problem the constraints are visualized in Figure 2.4b, where each line corresponds to a constraint $d \le (w-u)\cdot b$, and the gray area defines the feasible region of the linear program. The arrow indicates the direction for optimization, which is maximizing

Algorithm 1: `FindBelief` – computes the belief in which the vector w improves the value function induced by U the most

```
input  : vector set $U$, vector $w$
output : belief state $b$ or symbol $\phi$
1 if $U = \emptyset$ then
2     return arbitrary belief $b$
3 end
4 max  $d$
5 s.t.  $d \leq (w - u) \cdot b \quad \forall u \in U$
6       $\sum_{s \in S} b_s = 1$
7       $b_s \geq 0 \quad \forall s \in S$
8       $d \in \mathbb{R}$
9 return $b$ if $d > 0$ and $\Phi$ otherwise
```

in this case, and the dot shows where the optimal solution is located. In this case it can be seen that the optimal objective of the linear program is strictly positive, which means that there is a region of the belief space in which the vector w dominates the vectors in U. Moreover, in the example it can be seen that there is a direct correspondence between the dots highlighted in Figure 2.4a and Figure 2.4b.

The linear program can be integrated in a procedure `FindBelief` which finds the belief in which a vector $w \notin U$ improves the vectors in U the most, as shown in Algorithm 1. If adding w to U leads to an improvement then it returns the corresponding belief based on the solution of the linear program, and otherwise it returns the symbol Φ. If the set U is empty then the corresponding value function will always improve when adding w, and therefore the algorithm can return an arbitrary belief without solving the linear program.

The procedure `FindBelief` is a crucial component of many vector pruning algorithms. These algorithms incrementally build a parsimonious value function by iterating over vectors w, and for each of them it is checked whether it dominates previously added vectors in a region of the belief space. The vector is added to the parsimonious value function if it dominates, and the vector is discarded otherwise. This concept was first employed by Lark's vector pruning algorithm, which is described in a survey by White (1991) based on personal communication with Lark. Throughout this thesis we will refer to the algorithm as White & Lark.

The algorithmic description of the vector pruning algorithm by White & Lark is shown in Algorithm 2. As input it takes the vector set W, and it incrementally builds the parsimonious representation D. In each iteration of the algorithm a vector $w \in W$ is considered. On lines 4-5 the vector w is discarded if it is fully dominated by an existing vector in D, which means that there is a vector $u \in D$

Algorithm 2: Vector pruning algorithm (White & Lark)

```
input  : vector set $W$
output : pruned set $D$
1  $D \leftarrow \emptyset$
2  while $W \neq \emptyset$ do
3      $w \leftarrow$ arbitrary element in $W$
4      if $w(s) \leq u(s), \exists u \in D, \forall s \in S$ then
5          $W \leftarrow W \setminus \{w\}$
6      else
7          $b \leftarrow$FindBelief($D, w$)
8          if $b = \Phi$ then
9              $W \leftarrow W \setminus \{w\}$
10         else
11             $w \leftarrow$ BestVector($b, W$)
12             $D \leftarrow D \cup \{w\}$
13             $W \leftarrow W \setminus \{w\}$
14         end
15     end
16 end
17 return $D$
```

whose entries are all larger than the corresponding entries in w. On line 7 the algorithm uses the linear program to find a belief b in which the vector w improves the value function induced by D the most. If there is no belief for which the value function would improve, which is indicated by the symbol Φ, then the vector w is discarded. In all other cases the algorithm finds the value-maximizing vector in belief b from the set W on line 11 and adds this vector to D. The description of the corresponding `BestVector` procedure is shown in Algorithm 3, in which the operator $<_{\text{lex}}$ denotes the lexicographic ordering. If multiple vectors have an equal value in belief b, then this ordering turns out to be crucial to break ties when selecting the best vector (Littman, 1996).

The pruning algorithm by White & Lark is the most commonly-used vector pruning algorithm for POMDP value functions. However, additional methods have appeared in the literature. Most notably, a collection of region-based pruning methods has been proposed by Feng and Zilberstein (2004). These methods exploit the structure of the cross sum in incremental pruning to decide whether a vector should be part of the value function that is obtained when computing the cross sum of two value functions. This involves alternative LP formulations which typically contain fewer constraints than the LP found in the `FindBelief` procedure. A more

Algorithm 3: `BestVector` – returns the best vector from U in belief b

```
input  : vector set $U$, belief $b$
output : best vector from $U$ in belief $b$
1 $m \leftarrow -\infty$
2 for $u \in U$ do
3     if $(b \cdot u > m) \vee (b \cdot u = m \wedge u <_{lex} w)$ then
4         $w \leftarrow u$
5         $m \leftarrow b \cdot u$
6     end
7 end
8 return $w$
```

recent vector pruning algorithm is known as the Skyline algorithm (Raphael and Shani, 2012), which traverses the upper-surface of a value function and marks the vectors it visits. The algorithm only keeps the vectors that have been marked during execution. Furthermore, a more elaborate discussion on pruning is provided in the related work section at the end of Chapter 3.

2.2.8. Point-based value iteration

Exact value iteration becomes significantly faster due to pruning of dominated vectors, but unfortunately the scalability remains limited to relatively small problems. The reason is that the number of vectors can grow exponentially in the planning horizon, and if many of them dominate in a small region of the belief space then many of these vectors will be kept. In order to be able to compute policies for larger POMDPs, research has focused on the development of approximate algorithms. Point-based value iteration algorithms (Pineau, Gordon, and Thrun, 2003) form an important class of approximate algorithms which execute backups on a finite number of belief points $b \in B$, rather than optimizing over the entire belief simplex $\Delta(S)$. This can be advantageous, because within each dynamic programming stage the backup operator is only applied to beliefs $b \in B$, which means that the resulting number of vectors is bounded by the size of B.

Executing backups on a finite number of belief points only leads to improved tractability, but the question becomes how the set B should be constructed, which eventually influences the quality of the computed solution. Intuitively, the set B should provide coverage for the beliefs b that are reachable under the execution of an optimal policy. The vector α_{n+1}^{b} obtained by computing `backup`(b) defines the gradient of the value function in belief b, and therefore it can be expected that the value function also generalizes to belief points that are not part of B.

Various strategies have been proposed to initialize and update the set B. The original PBVI algorithm (Pineau, Gordon, and Thrun, 2003) executes an approximate backup operator $\tilde{H}$ as follows:

$$V_{n+1} = \tilde{H} V_n = \bigcup_{b \in B} \texttt{backup}(b). \tag{2.50}$$

The algorithm iteratively expands the set B by looking at the beliefs that are reachable from beliefs $b \in B$ in one step, and it adds the belief with the largest distance (e.g., based on the L_1 norm). This strategy aims to reduce the density of the belief points in B, which is defined as the maximum distance between two beliefs.

The Perseus algorithm (Spaan and Vlassis, 2005) is an improvement over PBVI. It starts with random sampling of belief trajectories by interacting with the POMDP model, based on which it constructs the belief set B. In each dynamic programming stage it executes backups on a randomly sampled belief subset $\tilde{B} \subseteq B$, rather than executing a backup for each $b \in B$. This is motivated by the observation that the vector α_{n+1}^{b} may also improve the expected value for beliefs other than b. The algorithm executes backups only on beliefs $b \in \tilde{B}$, but it ensures that $V_{n+1}(b) \geq V_n(b)$ holds for each $b \in B$. In contrast to PBVI, the set B remains fixed during the execution of the algorithm.

PBVI and Perseus only keep track of a lower bound on the optimal value function of the POMDP, represented by a set of vectors. More recent algorithms such as HSVI (Smith and Simmons, 2005), SARSOP (Kurniawati, Hsu, and Lee, 2008) and GapMin (Poupart, Kim, and Kim, 2011) are based on the same underlying idea as PBVI and Perseus, but they also keep track of an upper bound on the optimal value function. This is beneficial for two reasons. First, it allows for assessing whether a value function is far from optimal or not. Second, and most importantly, it allows the algorithms to iteratively expand the belief set B using a heuristic search procedure guided by the gap between the lower and upper bound. This search procedure ensures that the algorithms converge to an optimal value function in the limit, which means that they eventually deliver an optimal value function and hence an optimal policy. The main differences between HSVI, SARSOP and GapMin can be found in the heuristic search procedure, the computation of value upper bounds and the propagation of changes in these bounds.

2.2.9. Representing policies as policy graphs

Executing a POMDP policy induced by a set of vectors V_n is conceptually simple, but it requires additional computation time. First, it is required to keep track of a belief state, which is updated after executing an action and getting an observation. Second, when making a decision it is required to compute the dot product of the current belief and each vector in V_n. If an agent has limited time or resources to perform these calculations, then policy execution may become difficult.

Some POMDP policies have a special property which makes it possible to transform the vector representation in a graph representation, which does not require belief tracking during execution. This property is called finite transience (Sondik, 1971), and it can be explained as follows. Each vector $\alpha_j \in V_n$ has a belief region B_j in which it is the value-maximizing vector. If there is another vector $\alpha_k \in V_n$ with region B_k such that for each $b \in B_j$ it holds that $b_a^o \in B_k$ for each action a and observation o, then the policy induced by V_n is finite transient. In words this means that each belief in the region B_j is transformed into a belief in exactly the same region B_k when executing an action and making an observation.

A finite transcient policy can be converted into an equivalent policy graph, which can be interpreted as a finite-state controller used during execution. For each $\alpha_n^b \in V_n$ there is a node q_b with an associated action q_b^a, which is the action to be executed if q_b is the current node. For each observation o there is a node transition, formalized by q_b^o, which points to the node corresponding to the value-maximizing vector in belief b_a^o. Policy execution starts in the node q_0 corresponding to the value-maximizing vector of the initial belief b_1.

There is an elegant correspondence between the policy induced by the vectors in V_n and the nodes of the policy graph. We can compute the expected discounted reward of the policy induced by a policy graph by solving the following linear constraint system:

$$\mathcal{V}(q_b, s) = R(s, q_b^a) + \gamma \sum_{o \in O} \sum_{s' \in S} P(s' | s, q_b^a) P(o | q_b^a, s') \mathcal{V}(q_b^o, s') \quad \forall q_b,\ s \in S, \tag{2.51}$$

in which $\mathcal{V}(q,s)$ denotes the expected discounted reward obtained when starting policy execution in node q while the current state is s. The expected discounted reward of the policy graph is computed as:

$$\sum_{s \in S} \mathcal{V}(q_0, s) \cdot b_1(s). \tag{2.52}$$

It is interesting to observe that the expectations $\mathcal{V}(q_b, s)$ together can be interpreted as a vector $\mathcal{V}(q_b, \cdot)$ with an entry for each state. This means that for a policy graph derived from a vector set, we can again obtain $|S|$-dimensional vectors $\mathcal{V}(q, \cdot)$ corresponding to nodes q in the policy graph.

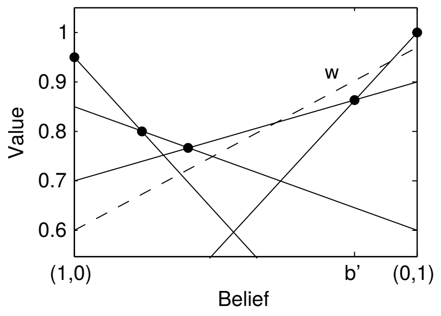
For finite-horizon problems policy graphs can be formulated and evaluated identically, except that there is a separate collection of nodes for each time step. This means that nodes at time t only transition to nodes of the next time step $t+1$. A full description of policy graphs and policy graph construction is provided in Chapter 5, in which we present an algorithm that generates a sequence of policy graphs during execution.

3 Accelerating optimal planning for POMDPs

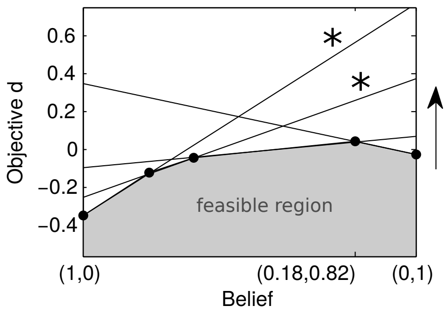
In the previous chapter we introduced exact value iteration algorithms for POMDPs which require a subroutine for pruning of vectors. This pruning operation is crucial as it reduces the number of vectors that is considered in subsequent iterations, but it turns out that the pruning operation itself is computationally expensive to execute. In this chapter we specifically focus on the vector pruning subroutine, and in particular the role of solving linear programs, which represents a significant fraction of the total computation time. The main contribution of this chapter is a POMDP vector pruning algorithm which uses a constraint generation procedure for accelerating the underlying linear programs. In addition, we also present an algorithm which improves the constraint generation procedure by bootstrapping from linear programs solved previously. An empirical evaluation shows that the resulting algorithm is the fastest pruning-based value iteration algorithm for solving POMDPs optimally.

3.1. Constraint generation for vector pruning

The vector pruning algorithm by White & Lark, as described in Algorithm 2 in the previous chapter, is a key element of the Incremental Pruning algorithm for solving POMDPs (Cassandra, Littman, and Zhang, 1997). It is used to incrementally prune the vectors produced by the exact Bellman backup, as shown in Equation 2.44. Within each call to `prune`, it potentially invokes the method `FindBelief` for each vector in the original set (on line 7 in Algorithm 2). This requires a linear program, and it turns out that solving these linear programs represents a major part of the



(a) Vectors in U and vector w (dashed)



(b) LP constraints of (U,w)

Figure 3.1: Value function U with vector w and the feasible region of the corresponding LP

total running time of incremental pruning (Cassandra, Littman, and Zhang, 1997).

In this section we present a constraint generation procedure to accelerate solving of linear programs within the vector pruning algorithm[1]. In Section 3.1.1 we analyze the structure of the original linear program, and we explain why this structure allows for deriving more efficient algorithms. In Section 3.1.2 and Section 3.1.3 we use the Benders decomposition method (Benders, 1962) for linear programs as a principled framework to derive a constraint generation procedure. Finally, in Section 3.1.4 we analyze the theoretical properties of the constraint generation procedure, which explains us why the resulting method is potentially faster than solving the original linear program.

3.1.1. Analysis of the LP formulation

In Section 2.2.7 we described the linear program that is required to check whether a vector w is dominated by a set of vectors U. For the specific example instance in Figure 2.4a we have illustrated how the constraints of the linear program in Equation 2.49 define a feasible region, as well as an optimal solution. As a first step in understanding the structure of the linear program, this section provides an initial analysis of the correspondence between the value function induced by U, the vector w and the resulting linear program. This analysis forms an important motivation for the methods introduced in the next section, because it intuitively explains why it is not always required to consider all constraints of the linear program explicitly.

Before analyzing the structure of the linear program, we first mathematically formalize a few key concepts, which makes further analysis and descriptions significantly easier. The linear program shown in Equation 2.49 is defined by the vector set U and the vector w. Therefore, we refer to this linear program as the

[1]The section is an extended version of a paper presented at AAAI-17 (Walraven and Spaan, 2017).

standard linear program, parameterized by the tuple (U,w). It maximizes the value improvement d, and returns the belief point b' where the vector w improves the value function induced by U the most. In Figure 3.1 we visualize the same example as before, augmented with additional symbols which we use throughout this section.

A *corner belief* is defined as a belief point in which the value function changes slope. In Figure 3.1a the corner beliefs are indicated by the dots shown on the upper surface of the value function. Each extremum of the belief simplex is also defined as an *extreme corner belief*. In the example in the figure we can see 5 corner beliefs, out of which 2 corner beliefs are also an extreme corner belief. Cheng (1988) has shown that the belief point where the vector w improves U the most is also one of the corner beliefs. We will refer to this belief as the *witness corner belief*. In Figure 3.1a and 3.1b this belief is represented by the belief b' at $(0.18, 0.82)$.

As discussed before, each line in Figure 3.1b corresponds to a constraint of the form $d \leq (w-u)\cdot b$, with $u \in U$. The objective of the linear program is represented by the vertical axis, and the set of feasible solutions of the linear program is represented by the shaded area under the concave surface. Based on the definitions of corners we are able to establish a relationship between the corner beliefs and the corners of the feasible region of the linear program, as shown below.

Lemma 1. *Each corner of the feasible region of the LP (U,w) corresponds to a corner belief of value function U.*

Proof. Based on Equation 2.45 we can formalize the concave surface defining the linear program as $w \cdot b - \max_{u\in U} u \cdot b$, where b is a belief. The expression $\max_{u\in U} u \cdot b$ corresponds to the convex surface of value function U. By definition, the slope of the value function U changes at the corner beliefs. Since w is a single vector that acts as a constant, it follows that the concave surface $w\cdot b - \max_{u\in U} u\cdot b$ also changes slope at the corner beliefs. □

As can be seen in Figure 3.1b, only constraints that are intersecting at the witness corner belief are necessary to define the optimal solution of the linear program. Other constraints, such as the constraints labeled with the asterix, can be removed without changing this optimal solution. If there are multiple witness corner beliefs with the same optimal objective value, then it is important to keep the intersecting constraints for at least one witness corner belief. In the theorem below we formalize our intuition that only a few constraints are necessary to define the optimal solution.

Theorem 1. *Constraints that do not intersect at the witness corner belief are irrelevant and can be removed without affecting the optimal objective value d.*

Proof. We assume that the value function U has m corner beliefs $b_1,\dots,b_m$ and without loss of generality we assume that b_m is the witness corner belief. We define $d(b)$ as the optimal objective value in belief b. From Lemma 1 we know that each corner belief b_l corresponds to a corner of the feasible region with objective value $d(b_l)$. It holds that $d(b_l) \leq d(b_m)$ for $l = 1, \dots, m-1$ because b_m is the witness corner belief and the objective is maximized. The LP returns the value $\max(d(b_1), \dots, d(b_m)) = d(b_m)$. Only the constraints intersecting at witness corner belief b_m are required to impose constraints on this value. □

The lemma and theorem tell us that it is possible to find the optimal solution of the linear program without adding all constraints, because it is only required to add the constraints intersecting at the witness corner belief. This is an important observation, because if the vector set U is large then there can be many constraints while only a few of them intersect at the witness corner belief. However, deciding which constraints are necessary to add is difficult as it requires knowledge about the (initially unknown) optimal linear program solution. This means that we are unable to decide which constraints are necessary without actually solving the linear program to optimality. Fortunately, we are able to use a constraint generation procedure which we can terminate in case we find the optimal solution before all constraints have been added. Details about this procedure are provided in the next two sections.

3.1.2. Benders decomposition method for linear programs

In order to derive a constraint generation scheme we use the Benders decomposition technique for decomposing linear programs (Benders, 1962). This decomposition technique can be used to decompose linear programs that are intractably large, and it provides a principled way to split a linear program into a so-called master problem and a slave problem. The linear program in the pruning algorithm is not necessarily large, but we can still use the decomposition method to create a constraint generation scheme while providing guarantees on convergence and optimality.

The Benders decomposition technique can be applied to linear programs having the following form:

$$\begin{aligned} \max\ & px + hy \\ \text{s.t. } & Cx + My \leq q \\ & x, y \in \mathbb{R}, \end{aligned} \tag{3.1}$$

where p and h are row vectors containing coefficients and the column vectors x and y represent decision variables of the linear program. The matrices C and M and the column vector q define the constraints, and they contain only constants. If the vector x is replaced by a fixed vector $\bar{x}$ containing only constants, then the linear program reduces to:

$$\begin{aligned} \phi(\bar{x}) = \max\ & hy \\ \text{s.t. } & My \leq q - C\bar{x} \\ & y \in \mathbb{R}, \end{aligned} \tag{3.2}$$

in which we move the constant $C\bar{x}$ to the right-hand side of the constraint, and we use $\phi(\bar{x})$ to denote the the optimal objective value for a fixed $\bar{x}$. Now we can state the optimization problem in (3.1) as follows:

$$\max_{x}\ px + \phi(x), \tag{3.3}$$

in which the max-operator should maximize over x in such a way that $\phi(x)$ defined in (3.2) has a feasible solution[2]. The dual of (3.2) can be written as:

$$\begin{aligned} \phi(\bar{x}) = \min\ & (q - C\bar{x})^{\top} z \\ \text{s.t. } & M^{\top} z = h^{\top} \\ & z \geq 0, \end{aligned} \tag{3.4}$$

where z is a column vector containing the dual decision variables and the symbol $\top$ denotes the transpose operator. Any vector z satisfying the dual constraints remains feasible if $\bar{x}$ in the objective function is replaced by another vector because the constraints of the dual problem do not depend on $\bar{x}$. When solving the dual in (3.4) for a given $\bar{x}$ to obtain the dual solution $\bar{z}$, then it holds that:

$$\phi(x) \leq (q - Cx)^{\top} \bar{z} \tag{3.5}$$

for all possible vectors x. This holds because $\bar{z}$ is dual feasible and for each x it represents a solution with objective value $(q - Cx)^{\top}\bar{z}$. The optimal objective value $\phi(x)$ cannot be larger than this quantity because (3.4) is a minimization problem.

A Benders decomposition algorithm initializes the following master problem:

$$\max\ px + \phi, \tag{3.6}$$

in which ϕ is a real-valued variable. First it solves the master problem to obtain $\bar{x}$. Then it solves the dual (3.4) to obtain the optimal dual solution $\bar{z}$. Based on this solution the constraint $\phi \leq (q - Cx)^{\top}\bar{z}$ is added to the master problem, which is called a Benders cut. This process is repeated until convergence, and it is important to note that this process is guaranteed to converge to an optimal solution to the original problem defined in (3.1). In the next section we use the same line of reasoning to derive a constraint generation scheme for solving the linear program used by the pruning algorithm.

[2]In this section we can always ensure that we maximize over x such that $\phi(x)$ is feasible. For Benders decompositions in general it may occur that the choice of x leads to an infeasible subproblem. In that case additional cuts can be added which enforce feasibility, but they are not described in this thesis.

3.1.3. Derivation of a constraint generation procedure

In this section we derive a constraint generation procedure for solving the linear program used for vector pruning:

$$\begin{aligned} &\max\ d \\ &\text{s.t. } d \leq (w-u)\cdot b \quad \forall u \in U \\ &\quad \sum_{s\in S} b_s = 1 \\ &\quad b_s \geq 0 \qquad\qquad \forall s \in S \\ &\quad d \in \mathbb{R}, \end{aligned} \tag{3.7}$$

which is the same linear program as the LP introduced earlier in Equation 2.49. The main idea of the derivation is that we can apply the steps outlined in (3.1)-(3.6) to the linear program in (3.7). For convenience we formalize this using matrix and vector notation. In particular, we define vector w and the vectors in $U = \{u_1,\dots,u_k\}$ as row vectors and $b = [b_1,\dots,b_{|S|}]^\top$ is a column vector. Based on these notational conventions we can rewrite the linear program as follows:

$$\begin{aligned} d^* = &\max\ [1][d] \\ &\text{s.t. } -\begin{bmatrix} w-u_1 \\ \vdots \\ w-u_k \end{bmatrix}\begin{bmatrix} b_1 \\ \vdots \\ b_{|S|} \end{bmatrix} + \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}[d] \leq \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}. \\ &\quad [1\dots1]\left[b_1\dots b_{|S|}\right]^\top = 1, \\ &\quad b_i \geq 0 \quad \forall i \in \{1,\dots,|S|\}, \\ &\quad d \in \mathbb{R}, \end{aligned} \tag{3.8}$$

Note that there is a correspondence with the linear program shown in (3.1). For example, b corresponds to x and d corresponds to y. If the vector b is replaced by a fixed belief $\bar{b}$ for which $\sum_{i=1,\dots,|S|} \bar{b}_i = 1$ and $\bar{b}_i \geq 0$ ($i = 1,\dots,|S|$), then (3.8) reduces to the LP below.

$$\begin{aligned} d^*(\bar{b}) = &\max\ [1][d] \\ &\text{s.t. } \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}[d] \leq C\bar{b} \quad \text{with } C = \begin{bmatrix} w-u_1 \\ \vdots \\ w-u_k \end{bmatrix} \\ &\quad d \in \mathbb{R} \end{aligned} \tag{3.9}$$

The dual of (3.9) can be written as:

$$\begin{aligned} d^*(\bar{b}) = &\min\ (C\bar{b})^\top z \\ &\text{s.t. } [1\dots1]z = 1, \\ &\quad z_j \geq 0 \quad \forall j \in \{1,\dots,k\}, \end{aligned} \tag{3.10}$$

where z is a column vector representing the dual solution. As before, any vector z satisfying the dual constraints remains feasible if the belief $\bar{b}$ is replaced by another belief, because the dual constraints are not dependent on $\bar{b}$. After solving (3.10) for a fixed belief $\bar{b}$ to obtain dual solution $\bar{z}$, we can obtain the following upper bound on the optimal LP solution d^*:

$$d^* \leq (Cb)^\top \bar{z}, \tag{3.11}$$

which is a valid upper bound for each belief b. The Benders decomposition algorithm initializes the following LP:

$$\begin{aligned} \max\ & d^* \\ \text{s.t. } & [1 \dots 1]b = 1 \\ & b_i \geq 0 \quad \forall i \in \{1, \dots, |S|\} \\ & d^* \in \mathbb{R} \end{aligned} \tag{3.12}$$

and solves this master LP to obtain $\bar{b}$. Then it solves (3.10) using $\bar{b}$ to obtain $\bar{z}$, after which constraint $d^* \leq (Cb)^\top \bar{z}$ is added to the master LP. This process repeats until the solution of the master LP converges to a fixed point, which is guaranteed to happen due to the construction based on the Benders decomposition principle.

We observe that the solution of (3.10) for a given $\bar{b}$ can be obtained without solving a linear program. It holds that $\bar{z} = (\bar{z}_1, \dots, \bar{z}_k)^\top$, with

$$\bar{z}_j = \begin{cases} 1 & j = \arg\min_{r \in \{1,\dots,k\}} \{(w - u_r)\bar{b}\} \\ 0 & \text{otherwise} \end{cases} . \tag{3.13}$$

Since $\bar{z}$ contains only one entry $\bar{z}_j$ that equals 1, the constraint in (3.11) can be written as $d^* \leq (w - u_j)b$ using row j of matrix C, where $j = \arg\min_{r=1,\dots,k} \{(w - u_r)\bar{b}\}$. The insight that the dual corresponds to finding the the vector $u \in U$ which minimizes $(w - u)b$ is convenient, because it prevents us from solving a linear program for the subproblem.

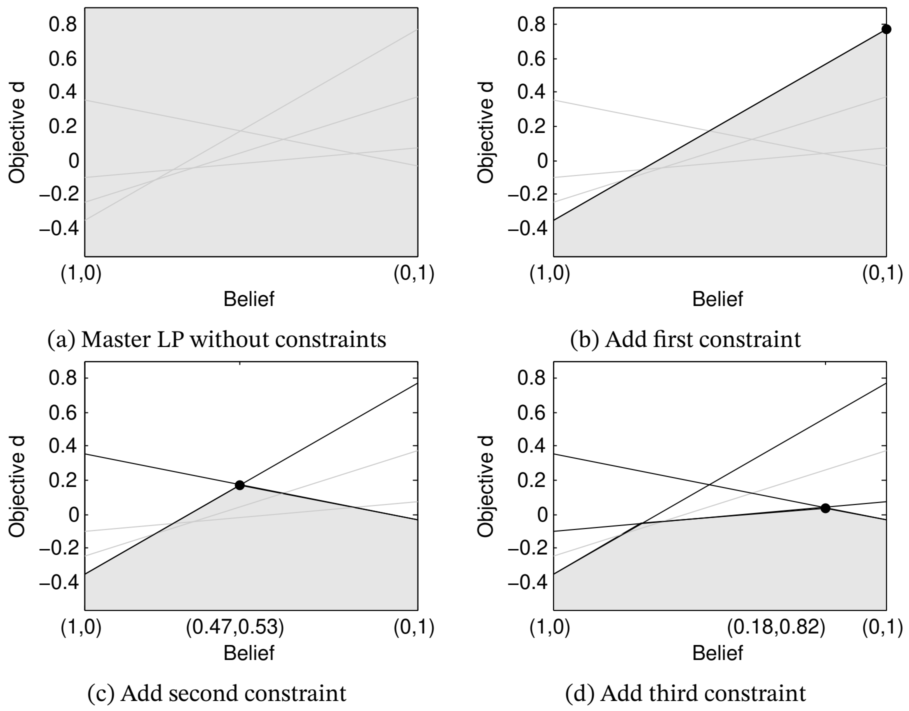
Based on the Benders decomposition method we have identified a master problem and a subproblem, which we can use to create a constraint generation procedure. We present an algorithm which replaces the procedure `FindBelief` shown in Algorithm 1. Our new algorithm is based on the principle of Incremental Constraint Generation (ICG), and therefore we call it `FindBeliefICG`, as shown in Algorithm 4. The algorithm is identical to Algorithm 1, except that it generates the constraints of the linear program incrementally. It first initializes a master LP which initially only imposes constraints on the belief b. This ensures that solving the master problem always gives a valid belief, which can be used to solve the subproblem. On lines 11-17 the algorithm iteratively selects vectors $\hat{u}$ and adds the corresponding constraints $d^* \leq (w - \hat{u})b$ to the master LP. In each iteration

Algorithm 4: `FindBeliefICG` – computes the belief in which w improves U the most, based on Incremental Constraint Generation (ICG)

```
input  : vector set $U$, vector $w$
output : belief $b$ or symbol $\phi$
1  if $U = \emptyset$ then
2      return arbitrary belief $b$
3  end
4  define the following master LP:
5  max $d^*$
6  s.t. $\sum_{s \in S} b_s = 1$
7       $b_s \geq 0 \quad \forall s \in S$
8       $d^* \in \mathbb{R}$
9  choose an arbitrary belief $b'$
10 $U' \leftarrow \emptyset$
11 do
12     $\bar{b} \leftarrow b'$
13     $\hat{u} \leftarrow \arg\min_{u \in U} \{(w - u) \cdot \bar{b}\}$
14     add $d^* \leq (w - \hat{u}) \cdot b$ to master LP
15     $U' \leftarrow U' \cup \{\hat{u}\}$
16     solve master LP to obtain belief $b'$
17 while $b' \neq \bar{b}$;
18 $\bar{d} \leftarrow$ last objective $d^*$ found
19 return $\bar{b}$ if $\bar{d} > 0$ and $\Phi$ otherwise
```

the master LP is solved to optimality, and it finds a new constraint which reduces the objective value for belief point b the most. If the belief points found in two successive iterations are identical, then the objective cannot be further reduced and the algorithm terminates. Since the algorithm only returns a belief on line 19 in case the optimal objective value is strictly greater than zero, constraint generation can be terminated if the optimal objective of the master LP drops below zero. This prevents running unnecessary iterations in which additional constraints are added without affecting the solution returned by the algorithm. The algorithmic description in this section does not include this minor detail because in our theoretical analysis it is important that the algorithm always finds an optimal solution. Additional implementation details are discussed separately in the shaded box at the end of the section.

From a general point of view, our constraint generation technique in Algorithm 4 can be interpreted as a cutting-plane method which iteratively refines the set of feasible solutions until an optimal solution is found. We can also interpret our



(a) Master LP without constraints
(b) Add first constraint
(c) Add second constraint
(d) Add third constraint

Figure 3.2: Example illustrating execution of `FindBeliefICG` (Algorithm 4)

method as a decomposition of the original LP shown in Equation 3.7, because we have divided the original problem into a smaller master problem and a subproblem.

The execution of Algorithm 4 can be illustrated using an example, as shown in Figure 3.2. Execution starts with a master linear program in which the constraints corresponding to the vectors have not been added yet. In Figure 3.2a the shaded area represents the (unbounded) feasible region, and the four gray constraints represent constraints that have not been added. The algorithm starts in belief point $(1, 0)$ and selects the constraint which reduces the objective value the most. This constraint is added in Figure 3.2b, in which the new optimal solution is indicated by the dot. Now the algorithm selects the constraint which reduces the objective value the most for belief point $(0, 1)$, which is added in Figure 3.2c. In the final step the algorithm selects the constraint which reduces the objective the most in belief point $(0.47, 0.53)$, and this constraint is added in Figure 3.2d. At this point the optimal solution of the linear program corresponds to the belief point $(0.18, 0.82)$. For this point no constraint can be found which further reduces the objective value, and hence the algorithm terminates. In this example there is one constraint, corresponding to the gray line in Figure 3.2d, which was not added. This constraint was considered while finding constraints which reduce the objective value, but it was never considered while solving the master linear program. This illustrates how `FindBeliefICG` is able to find an optimal solution without adding all possible constraints.

As a final remark, it should be noted that `FindBeliefICG` can be easily integrated in incremental pruning for POMDPs, because it only replaces the `FindBelief` procedure in the vector pruning subroutine. This also means that the optimal POMDP solution computed by incremental pruning remains unaffected.

Implementation detail: early termination of constraint generation

The constraint generation procedure outlined in Algorithm 4 only returns the belief $\bar{b}$ in case $\bar{d} > 0$. In all other cases it returns the symbol Φ, which indicates that there is no belief in which the vector w improves the value function U. Based on this insight we can conclude that it is possible to terminate constraint generation before reaching the optimal solution to the original LP in Equation 2.49. To be more specific, constraint generation can be terminated if the optimal solution to the master LP becomes strictly negative, because in the remaining iterations the optimal objective value never becomes positive and Algorithm 4 will always return Φ. Unless stated otherwise our implementation always terminates constraint generation if the optimal master LP objective drops below zero. In a few experiments we disable this feature because the experimental setup requires us to do so, which will be mentioned explicitly in the corresponding sections.

3.1.4. Theoretical analysis

Our constraint generation procedure is easy to integrate in the incremental pruning algorithm for POMDPs, but it is not immediately clear why this procedure would accelerate solving. In this section we provide a theoretical analysis which aims to provide insight into the behavior of Algorithm 4. Moreover, the analysis makes clear why the algorithm does not always add irrelevant constraints, and why this can eventually lead to a reduction of the time required for solving linear programs.

First we characterize the optimality of the constraint generation procedure. The theorem below indicates that Algorithm 4 returns a belief point for which the corresponding optimal objective value is identical to the optimal objective value found by Algorithm 1. The belief point returned may be different if there are multiple belief points with the same optimal objective value. Since both belief points correspond to the same value improvement, they both represent a witness corner belief.

Theorem 2. *Algorithm 4 computes a solution with the same optimal objective value d as the linear program in Algorithm 1, and constraint generation terminates after a finite number of iterations.*

Proof. The constraint generation procedure has been derived using a Benders decomposition (Benders, 1962). Hence, the optimal objective value of the solution returned is identical and it is found in a finite number of iterations. □

Before we proceed with our analysis we introduce the notation corresponding to important concepts which we use throughout the analysis. Algorithm 4 incrementally adds constraints, and each constraint corresponds to a vector $u \in U$. At any point in time during the execution of the Algorithm 4, the constraints added to the master LP are defined using a set $U' \subseteq U$. This set is also defined on line 10 of Algorithm 4. For each $u \in U'$ there is a constraint $d \leq (w-u)\cdot b$. The constraints in $U' \subseteq U$ define an optimal solution b' and the corresponding objective value d'. If the algorithm adds a constraint u on line 14 for a given $\bar{b}$, then we say that the algorithm *uses* belief $\bar{b}$ to add u. The region Z_u in which u restricts the LP solution space is:

$$Z_u = \{b \mid (w-u)\cdot b \leq (w-u')\cdot b \quad \forall u' \in U'\}. \tag{3.14}$$

The belief b' has neighbors $b_1,\dots,b_l$ which are also corners of the feasible region, with corresponding objective values $d(b_1),\dots,d(b_l)$. In Figure 3.1b each corner of the feasible region has two neighbors, except the corners at the extrema of belief space. For state spaces with more than two states corner beliefs may have more than two neighbors. We define the neighbors of b' using a set $NB(b')$:

$$\begin{aligned} NB(b') = \{b \mid\ & b \text{ is corner belief and } \exists c \in U' \\ & \text{such that } b' \in Z_c \text{ and } b \in Z_c, b \neq b'\}. \end{aligned} \tag{3.15}$$

This set contains the corners b of the feasible region that can be reached from b' in one step, because there is at least one constraint $c \in U'$ such that $b' \in Z_c$ and $b \in Z_c$. The lowest objective value of the neighbors is $d_{\min}(b') = \min_{b \in NB(b')} d(b)$. Due to the convexity of feasible regions of LPs[3], it holds that $d_{\min}(b') \leq d'$. The region $Z(b')$ in which the objective value is at least $d_{\min}(b')$ is defined as:

$$Z(b') = \{b \mid \min_{u \in U'}\{(w-u)\cdot b\} \geq d_{\min}(b')\}. \tag{3.16}$$

In the example in Figure 3.3 the lines (except the bold vertical line) correspond to constraints in U. The black constraints have been added so far and belong to the set $U' \subseteq U$. The belief b' is the current optimal solution of the master LP, and its two neighbors are represented by dots. In the example it holds that

[3]In the visualization in Figure 3.1b the feasible region of the maximization problem is concave, but in general linear programming solvers use the equivalent convex minimization problem.

$d_{\min}(b')$ equals 0.05, and therefore the region $Z(b')$ contains the beliefs in which the objective is at least 0.05, which is represented by the horizontal double arrow.

We first show that the current optimal solution b' of the master LP is always a belief point in $Z(b')$ while executing iterations in which constraints are added. The original linear program in Algorithm 1, as shown in Equation 3.7, will be referred to as the standard LP. The theorem below shows that the optimal objective value of the standard LP, which corresponds to the objective value $\bar{d}$ on line 18 of Algorithm 4, is at least $d_{\min}(b')$ throughout the execution of Algorithm 4. This immediately implies that the final solution $\bar{b}$ returned by Algorithm 4 is a belief point in $Z(b')$.

Theorem 3. *Given the current optimal solution b' and the corresponding objective value d' of the master LP, it holds that $d^* \geq d_{\min}(b')$, where d^* is the optimal objective value of the original LP.*

Proof. By contradiction. We assume that $d^* < d_{\min}(b')$. For each $b \in Z(b')$ there must be a constraint $u \notin U'$ such that $(w-u)\cdot b \leq d^* < d_{\min}(b')$. We consider an arbitrary neighbor $b_l \in NB(b')$ of b' and a constraint $c \in U'$ such that $b' \in Z_c$ and $b_l \in Z_c$. All corner beliefs $b \in Z_c$ except b' are also neighbor of b' according to definition of NB, which implies that $d(b) \geq d_{\min}(b')$ for each $b \in Z_c$. Now we can conclude that $Z_c \subseteq Z(b')$. Consider the belief b^c that was used to add c. We know that $b^c \in Z_c$ because b^c is a belief in which c restricts the current LP solution space. It is impossible that $b^c \notin Z_c$ because outside the region Z_c there is already another constraint which is more restrictive than c in point b^c, which would have been selected in point b^c instead of c. It holds that $d(b^c) \geq d_{\min}(b')$ because $b^c \in Z_c \subseteq Z(b')$. For b^c there must be a constraint $u \notin U'$ for which $(w-u)\cdot b \leq d^* < d_{\min}(b')$. Constraint u must have been added before c on line 11, which leads to a contradiction. □

Corollary 1. *Given the current optimal solution b' and the corresponding objective value d' of the master LP, it holds that $\bar{b} \in Z(b')$, which means that the optimal solution $\bar{b}$ of the standard LP is a belief point in $Z(b')$.*

Proof. Follows immediately from Theorem 3 the definition of $Z(b')$. □

Corollary 2. *Consider the current optimal solution b'. Algorithm 4 only finds belief points $b \in Z(b')$ during subsequent iterations.*

Proof. By contradiction. Suppose that a belief $b \notin Z(b')$ is found during a subsequent iteration, then it holds that $d(b) < d_{\min}(b')$. For each $b \in Z(b')$ there must be a constraint $u \notin U'$ such that $(w-u)\cdot b < d_{\min}(b')$. There exists a constraint $c \in U'$ for which $Z_c \subseteq Z(b')$, and we consider the belief b^c that was used to add c. It holds that $b^c \in Z_c$ because b^c is a belief point in which c restricts the current LP solution space. Moreover, it holds that $d(b^c) \geq d_{\min}(b')$ because $b^c \in Z_c \subseteq Z(b')$. In

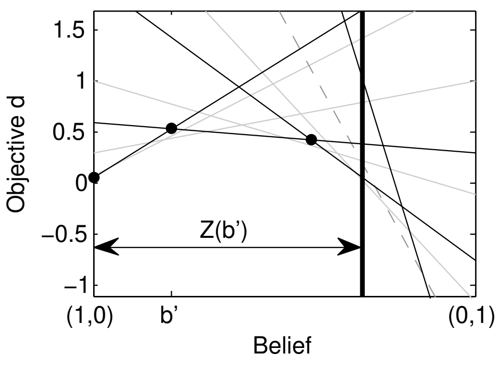


Figure 3.3: Region $Z(b')$ example

belief b^c there must be a constraint $u \notin U'$ for which $(w-u)\cdot b < d_{\min}(b')$. Hence, constraint u must have been added before c, which leads to a contradiction. □

We can use the theorem and corollaries to define when a constraint $u \notin U'$ is never added during subsequent iterations of Algorithm 4. This is relevant, because it shows us why the algorithm does not always add a constraint for each $u \in U$.

Theorem 4. *Consider the current optimal solution b' and a constraint $c \notin U'$. If $Z_c \cap Z(b') = \emptyset$, then constraint c will never be added to the master LP in subsequent iterations.*

Proof. For each $b \in Z_c$ it holds that $(w-c)\cdot b < d_{\min}(b')$ because $Z_c \cap Z(b') = \emptyset$. During subsequent iterations Algorithm 4 will never find a belief b in which $d(b) < d_{\min}(b')$, because it terminates after finding the optimal solution, which is at least $d_{\min}(b')$ according to Theorem 3 and Corollary 1. This implies that Algorithm 4 never finds a belief $b \in Z_c$ during subsequent iterations. Hence, constraint c is never added to the master LP during subsequent iterations. □

Figure 3.3 visualizes the ideas behind Theorems 3 and 4. The optimal solution of the original LP belongs to the region $Z(b')$ and is at least $d_{\min}(b')$. The dashed constraint restricts the solution space in a region that is not part of $Z(b)$, and therefore it is never added in remaining iterations. The condition stated in Theorem 4 is a sufficient condition and tells us whether it is guaranteed that a constraint is not going to be added in remaining iterations. If the size of $Z(b')$ shrinks quickly during the execution of Algorithm 4, then this can reduce the number of added constraints significantly. If there is a large number of constraints restricting the solution space in regions outside $Z(b')$, then these constraints will never be considered explicitly by the linear programming solver, which potentially leads to a running time reduction.

3.2. Bootstrapping for constraint generation

In the previous section our attention has focused on constraint generation for individual linear programs in vector pruning algorithms for POMDPs. These linear programs are used to verify whether there exists a region of the belief space in which a given vector dominates a value function. A key property of exact value iteration for POMDPs is that the Bellman backup operator is applied until the computed value function has converged to a fixed point (Bellman, 1957). During the execution of value iteration the value functions become increasingly similar, as well as the linear programs that are executed while pruning value functions. However, so far this property remains unexploited as the linear programs within an iteration of value iteration are solved without looking at the linear programs that have been solved in previous iterations of value iteration. In this section we present a bootstrapping procedure which uses linear programs from the previous iteration to initialize a subset of constraints for a new linear program before constraint generation actually starts[4]. If the constraints added based on bootstrapping would have been added by the original constraint generation procedure as well, then this reduces the number of constraint generation iterations and also running time.

3.2.1. Convergence of value iteration

Exact value iteration repeatedly executes Bellman backups based on the Bellman optimality equation shown in Equation 2.22. As discussed before, it first initializes a value function V_0 and then it generates a sequence of new value functions V_{n+1} based on the previous value function V_n. For infinite-horizon POMDPs with discounting this is known to converge to a fixed-point solution (Bellman, 1957), such that:

$$V_{n+1} = HV_n = V_n, \tag{3.17}$$

where H denotes the Bellman backup operator. This occurs when the Bellman error magnitude has become zero, which is defined as:

$$\max_{b \in \Delta(S)} |V_{n+1}(b) - V_n(b)|. \tag{3.18}$$

In other words, while executing exact value iteration the vectors constituting the computed value functions V_n get increasingly similar and eventually they remain constant. In that case it is not required to execute additional iterations of value iteration, because this produces identical value functions.

3.2.2. Bootstrapping of linear program constraints

In this section we consider two successive value functions V_{n-1} and V_n. As explained in the previous section, we expect the set of vectors V_n in incremental

[4]The section is based on a paper presented at ICAPS-18 (Roijers, Walraven, and Spaan, 2018).

pruning to contain increasingly similar vectors to V_{n-1}. To identify which vectors need to be retained in V_n, the vector pruning algorithm of White & Lark (Algorithm 2) incrementally builds up the parsimonious vector set. Starting from an empty set, this algorithm identifies a belief b for which a candidate vector w is possibly optimal. The best vector for that b is added to V_n. To identify b, Algorithm 4 solves a series of LPs with an increasing number of constraints based on our constraint generation procedure. Based on the characteristics of Algorithm 4 and the convergence characteristics of value iteration we can make two observations, as described next.

First, we observe that the beliefs identified by Algorithm 4 become increasingly similar during the execution of value iteration. To illustrate this, we consider a call to `prune` in Equation 2.44 in iteration $n-1$ and the corresponding call to `prune` in the iteration n. These calls have the form

$$\texttt{prune}\left(\ldots \bigoplus \bar{G}_a^o\right) \tag{3.19}$$

in Equation 2.44, parameterized by action a and o. Two calls in successive iterations correspond to each other if a and o are identical. Without loss of generality we assume that the pruning algorithm always considers the vectors in a specific (e.g., lexicographic) order[5]. For both method calls we consider a vector w_{n-1} and a vector w_n, respectively. These vectors are used as input to the ICG linear program (Algorithm 4) as the candidate vector and if the vectors are similar then the identified point returned by Algorithm 4 will be similar too[6]. In other words, similar vectors in two successive iterations will be optimal for similar beliefs. This is because value iteration converges to a fixed point of the value function, which is a convex set of vectors. As an example we can consider the linear programs shown in Figure 3.4. In Figure 3.4b each line corresponds to a constraint and, as can be seen, the optimal solution is similar to the optimal solution of the linear program in Figure 3.4a.

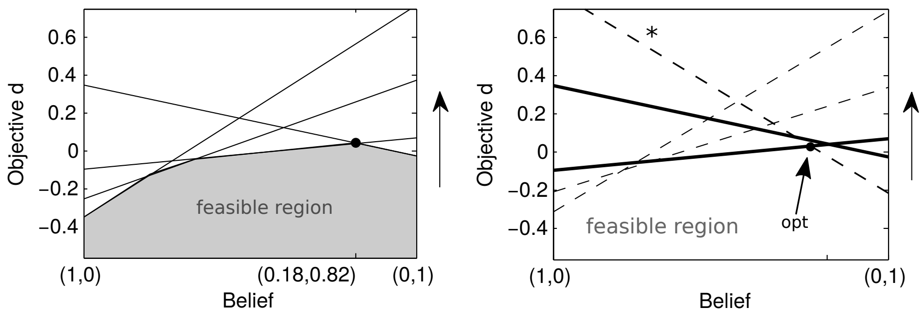
Second, we observe that there are only a handful of constraints, $\mathcal{C}_{prev}$, that ultimately constitute the solution of the linear program. In Figure 3.4a these constraints are the constraints intersecting in the dot corresponding to the optimal solution. These constraints can be identified easily (following the notation of Algorithm 4) as:

$$\mathcal{C}_{prev} = \underset{u \in U}{\arg\min}(w-u)\cdot \bar{b}, \tag{3.20}$$

where $\bar{b}$ is the ultimately returned point. Note that $\mathcal{C}_{prev}$ is a set containing vectors corresponding to the constraints intersecting in $\bar{b}$. For the example this would be

[5]More details about this assumption will be provided in Section 3.2.3.

[6]The vector pruning algorithm by White & Lark may visit one vector multiple times during pruning, which means that it identifies multiple points for one given vector. In that case the identified points will also get increasingly similar. We discuss this detail in the shaded box at the end of the section.



(a) LP in iteration $n-1$ (b) LP in iteration n

Figure 3.4: Bootstrapping example with LPs in iteration $n-1$ and iteration n

the vectors intersecting in the dot in Figure 3.4a. Since there is a direct correspondence between vectors and constraints, we will use both terms interchangeably if the meaning is clear from context.

By combining the two observations we hypothesize that we can reuse the constraint set induced by $\mathcal{C}_{prev}$ for other vectors similar to vector w_{n-1}. We consider iteration $n-1$, in which `FindPoint(ICG)` was called with a vector set U_{n-1} and a vector w_{n-1}. If the same function is called in iteration n with vector set U_n and a similar vector w_n, then we select the closest vectors from the new set U_n (according to Euclidean distance) to initialize the linear program in iteration n:

$$\mathcal{C}_{init} = \bigcup_{u_{n-1} \in \mathcal{C}_{prev}} \arg\min_{u_n \in U_n} |u_n - u_{n-1}|. \tag{3.21}$$

This set contains vectors from U_n similar to the vectors from U_{n-1} which correspond to the constraints defining the optimal solution for w_{n-1}.

As an example we consider the linear program shown in Figure 3.4a. Suppose that ICG solves this linear program and finds the solution indicated by the dot. Now suppose that we encounter a similar linear program in a subsequent iteration of incremental pruning, as shown in Figure 3.4b. In this case we would like to initialize constraints which are likely to be intersecting in the optimal solution. Therefore our bootstrapping technique initializes the linear program with the constraints shown as a bold solid line, as these constraints are similar to the constraints intersecting in the optimal solution in the previous iteration (see Figure 3.1b). In the new linear program the optimal solution (indicated by opt) is slightly different compared to the previous linear program, and constraint generation needs to add only one more constraint (labeled $*$). This is beneficial, as the original ICG would start from an empty linear program, and iterates multiple times before reaching the same solution.

It is important to note that the initialization of constraints based on $\mathcal{C}_{init}$ never shrinks the feasible region of the linear program too much. The reason is that the constraints defined by $\mathcal{C}_{init}$ correspond to vectors from the current vector set U_n, rather than vectors from a linear program solved previously. In other words: the algorithm always initializes constraints that are valid constraints in the linear program that is currently being solved.

3.2.3. Integrating bootstrapping in vector pruning

In order to use the concept of bootstrapping constraints we need to modify the vector pruning algorithm, as well as the algorithm for finding beliefs. In this section we discuss the modifications in both algorithms.

To perform bootstrapping, we need to store and retrieve the sets $\mathcal{C}_{prev}$. As discussed in the previous section, within a call to `prune` we want to retrieve such sets based on the corresponding call to `prune` from the previous iteration. In other words, the constraints and solutions that can be reused are context-dependent, where the context refers to the location in Equation 2.44 where `prune` is invoked. To integrate context-dependent bootstrapping we make the following two changes to the original pruning algorithm.

First, we ensure that the vectors that need to be pruned are lexicographically ordered. That is, each time Algorithm 2 is called on a set of vectors U, we sort the set in this order. Consistency in this order is crucial for our first observation in the previous section, as the order in which Algorithm 2 considers the vectors influences the sequence of arguments `FindBeliefICG` (U and w) is called with, as well as the belief points returned by `FindBeliefICG`.

Second, the `prune` subroutine is implemented as an adapted version of Algorithm 2, with the following modifications: it is now parameterized by iteration n, and context element θ, which represents the action-observation pair (a, o). On the top level (i.e., the `prune` call after the union over all sets in Equation 2.44) we use `null` as the context element. The iteration number and context are used to identify the LPs based on which bootstrapping can be performed. Hence, the pruning subroutine passes these arguments to a new subroutine to identify beliefs while reusing LP information, `FindBeliefBLP`, as implemented in Algorithm 5. This subroutine replaces `FindBelief` and `FindBeliefICG`.

The algorithm `FindBeliefBLP` retrieves the relevant constraints from the previous iteration $n-1$. It does this by matching the closest vector w' from a cache with the same context from the previous iteration on line 4. Aside from w', the constraints $\mathcal{C}_{n-1}$ in the form of vectors from the previous iteration and the belief that was optimal for the corresponding LP are also retrieved. If this is the first iteration, i.e., the cache is empty, we use a vector of zeroes, an empty set of constraints and an arbitrary belief as default. Because $\mathcal{C}_{n-1}$ is in the form of vectors we can match the closest vectors from U in the current iteration, n, on line 5. These vectors are stored in a set $\mathcal{C}_{init}$, and it is used to the initialize the constraints of the LP on

Algorithm 5: `FindBeliefBLP` – computes the belief in which w improves U the most

```
input  : vector set $U$, vector $w$, and a context iteration number $n$, and
         context element $\theta$
output : belief $b$ or symbol $\phi$
1  if $U = \emptyset$ then
2      return arbitrary belief $b$
3  end
4  $w_{n-1}, \mathcal{C}_{n-1}, b' \leftarrow \arg\min_{(w',\mathcal{C},b') \in \texttt{cache}(n-1,\theta)} |w - w'|$
5  $\mathcal{C}_{init} \leftarrow \bigcup_{c \in \mathcal{C}_{n-1}} \arg\min_{u \in U} |c - u|$
6  define the following LP:
7  max  $d^*$
8  s.t. $\sum_{s \in S} b_s = 1$
9       $b_s \geq 0$                        $\forall s \in S$
10      $d^* \leq (w - \hat{u}) \cdot b$    $\forall \hat{u} \in \mathcal{C}_{init}$
11      $d^* \in \mathbb{R}$
12 do
13     $\bar{b} \leftarrow b'$
14     $\hat{u} \leftarrow \arg\min_{u \in U} \{(w - u) \cdot \bar{b}\}$
15     add $d^* \leq (w - \hat{u}) \cdot b$ to the LP
16     solve the LP to obtain belief $b'$
17 while $b' \neq \bar{b}$;
18 $\mathcal{C}_{prev} \leftarrow \arg\min_{u \in U} (w - u) \cdot \bar{b}$
19 add $(w, \mathcal{C}_{prev}, \bar{b})$ to the cache$(n, \theta)$
20 $\bar{d} \leftarrow$ last objective $d^*$ found
21 return $\bar{b}$ if $\bar{d} > 0$ and $\Phi$ otherwise
```

line 10. The initial constraints are of the form: $d^* \leq (w - \hat{u}) \cdot b$, where w is the input vector, and $\hat{u}$ is a vector in $\mathcal{C}_{init}$. After constructing the initial LP, BLP generates the constraints incrementally until an optimal solution has been found (similar to `FindPointICG` in Algorithm 4) on lines 12–17. This leads to the final solution $\bar{b}$ of the LP. Given the LP solution $\bar{b}$, the method `FindBeliefBLP` retrieves and stores constraints and $\bar{b}$ itself for reuse in subsequent iterations. The constraints $\mathcal{C}_{prev}$ are those $u \in U$ that are optimal for $\bar{b}$ (line 18). This is stored in the cache (line 19) before returning $\bar{b}$ if there is a belief for which w is an improvement over U, or Φ if there is not.

Compared to the original FindBeliefICG subroutine, the bootstrapping procedure introduces additional overhead in the form of bookkeeping which is necessary to match the contexts and the similar constraints. Furthermore, sorting induces extra work, which is $O(|S||U|\log|U|)$ for sorting vectors in U lexicographically. Finally, it should be noted that FindBeliefBLP does not change the solutions, because the calls to the belief finding subroutine and the corresponding output remain identical. Moreover, constraints added due to bootstrapping are always valid constraints in the linear program that is currently being solved.

Implementation detail: integrating bootstrapping in vector pruning

The vector pruning algorithm shown in Algorithm 2 may visit one vector w multiple times. This occurs if the vector identified by BestVector is not identical to w. As a result, the algorithm calls FindBeliefBLP multiple times for the same vector w, but the returned belief is not the same because the input vector set U is different. This leads to multiple entries in the cache for the same vector w. It is important to note that these entries also get increasingly similar during execution of incremental pruning. If there are multiple cache entries for the same vector w, then our implementation constructs the set $\mathcal{C}_{init}$ based on multiple cache entries, rather than just one. This ensures that the cache entry corresponding to the current w is always considered. Another implementation detail arises when considering the *generalized* incremental pruning algorithm for POMDPs. In this version the pruning algorithm invokes FindBelief for a subset of D in Algorithm 2, depending on the size of D. If the subset choice in iteration $n-1$ is not the same as in iteration n, then the identified belief may become different too, which means that information from the previous iteration in the cache is not always useful. It should be noted, however, that the choice of the subset also converges when the value function converges. In experiments which test specific bootstrapping characteristics we only consider the standard incremental pruning algorithm, such that the uncontrollable subset choice does not introduce noise.

3.3. Experiments

In this section we present the results of our experimental evaluation. We start with the experiments that test the constraint generation procedure. After that, we present the results of experiments which evaluate our bootstrapping procedure combined with constraint generation.

3.3.1. Constraint generation experiments

In this section we provide an experimental evaluation of the constraint generation procedure. We perform our evaluation at three different levels. First, we study

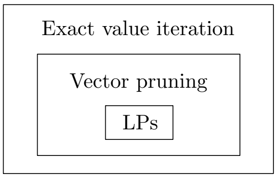


Figure 3.5: Three-level hierarchy of the experiments

individual LPs which check whether there is a belief point in which a vector dominates a set of vectors. Second, we study the performance of the vector pruning algorithm by White & Lark, which solves its LPs using constraint generation. Third, we study whether the vector pruning algorithm combined with constraint generation accelerates exact value iteration. The three abstraction levels are shown in Figure 3.5, which illustrates that exact value iteration relies on pruning. Pruning requires solving LPs for which constraint generation can be used.

We execute our experiments on all POMDP domains used by Cassandra, Littman, and Zhang (1997), Feng and Zilberstein (2004) and Raphael and Shani (2012). These papers represent the previous state of the art in optimal POMDP solving using pruning-based algorithms. More information about the size of the POMDP domains considered is provided in Table 3.1. Note that the domain 4x3CO is not included in the experiments because it represents a POMDP with full observability.

Performance of constraint generation

In the first experiment we compare `FindBelief` and `FindBeliefICG`. The main objective of this experiment is investigating whether solving LPs with constraint generation requires less time than solving the LPs immediately based on all constraints. For each domain we consider the first 30000 LPs that are solved during the execution of incremental pruning. Some domains could be solved with fewer than 30000 LPs, and for RockSmp4x4 we only consider the first 1000 LPs due to memory limits. The actual number of LPs considered is also listed in Table 3.1. For each LP we execute `FindBelief` and `FindBeliefICG`, during which we measure the running times and the number of constraints added by `FindBeliefICG`.

The results of our runtime evaluation are shown in Table 3.2, which shows the total runtime of the LPs in `FindBelief` (column Std) and the total runtime required to solve the same LPs in `FindBeliefICG` (column Constr Gen). For each domain the column Speedup shows the corresponding speedup. From the results we conclude that constraint generation reduces the running time that is required to obtain LP solutions. In order to explain why constraint generation leads to a speedup, we also measure the number of constraints that was actually added during the execution of `FindBeliefICG`, as shown in the column Constr. As can be seen,

Name	$\|S\|$	$\|A\|$	$\|O\|$
1D	4	2	2
4x3	11	4	6
4x4	16	4	2
4x5x2	39	4	4
AircraftID	12	6	5
Cheese	11	4	7
Hallway	60	5	21
Hallway2	92	5	17
Network	7	4	2
Partpaint	4	4	2
RockS4x4	257	9	2
Shuttle	8	3	5
Tiger-grid	36	5	17

Table 3.1: Parameters of the POMDP domains

in many cases the constraint generation procedure only adds a small fraction of the constraints. For example, in the Hallway2 domain it adds 17.87 percent of the constraints on average. The relatively large standard deviation can be explained by observing that in small LPs a relatively large fraction of the constraints is needed, which subsequently affects the standard deviation. We further analyze this in Figure 3.6, which visualizes percentage of constraints used as a function of the total number of constraints in the LP. As expected, in LPs with a few constraints the algorithm adds a relatively large number of constraints, and in LPs with many constraints it adds only a small fraction of the constraints. The domain 1D is not included because there are a few LPs with only a few constraints, which does not allow us to study the relationship between the total number of constraints and the number of constraints added.

The relationship between the total number of constraints and number of constraints used, as shown in Figure 3.6, also enables us to explain when constraint generation potentially introduces overhead. If the number of constraints is small, then a relatively large number of constraints is added iteratively. Under these circumstances it may be better to initialize the constraints all at once, rather than applying our iterative constraint generation procedure. We further discuss this implementation detail in the shaded box at the end of this section.

We conclude that constraint generation is able to reduce the time required to solve the LPs. Moreover, the experiment confirms our initial motivation from Section 3.1.4, which hypothesizes that constraint generation may lead to a speedup due to the fact that only a few constraints eventually define an optimal LP solution.

Domain	#LPs	Std (s)	Constr Gen (s)	Speedup	Constr (%)
4x5x2	30000	67.01	23.70	2.83	11.75 ± 15.45
AircraftID	30000	45.09	16.19	2.79	8.79 ± 14.69
Hallway2	30000	64.89	27.05	2.40	17.87 ± 27.07
Tiger-grid	30000	58.67	26.63	2.20	12.64 ± 18.48
RockS4x4	1000	0.84	0.47	1.80	37.80 ± 23.51
4x3	30000	35.40	21.97	1.61	17.72 ± 19.39
Shuttle	30000	29.04	21.14	1.37	18.67 ± 20.16
Hallway	30000	23.16	19.74	1.17	26.80 ± 27.73
1D	50	0.01	0.01	1.00	84.0 ± 21.85
Cheese	1514	0.19	0.20	0.99	87.78 ± 22.18
Network	30000	10.96	13.41	0.82	26.32 ± 19.62
4x4	2836	0.43	0.56	0.78	75.98 ± 22.52
Partpaint	7571	1.67	2.27	0.74	38.66 ± 31.81

Table 3.2: Comparison of LPs solved in `FindBelief` and `FindBeliefICG`

Performance of vector pruning

Now we study the performance of the vector pruning algorithm by White & Lark, which invokes `FindBeliefICG` rather than `FindBelief`. This enables us to study whether constraint generation for LPs improves the performance of the vector pruning procedure. In the previous experiment we found that constraint generation leads to a running time reduction at the level of individual LPs. The vector pruning algorithm solves many LPs, and therefore we expect that it also reduces the total running time required for pruning.

Implementation detail: no constraint generation for small LPs

The results in Figure 3.6 show that the constraint generation procedure iteratively adds almost all constraints if the size of U is small. This suggests that constraint generation potentially introduces overhead if the LP is small, and in such cases it may be better to initialize all constraints immediately rather than iterating multiple times. Our implementation invokes the regular `FindBelief` procedure rather than `FindBeliefICG` or `FindBeliefBLP` in case U contains fewer than 50 vectors. This parameter was selected because we found empirically that it provides good performance, and it aligns with the results observed in Figure 3.6 because typically the percentage decays quickly when the number of constraints exceeds 50. We only apply the threshold 50 in the experiments in which we study the performance of incremental pruning and bootstrapping. It is not applied in the vector pruning experiment because there we also discuss potential overhead.

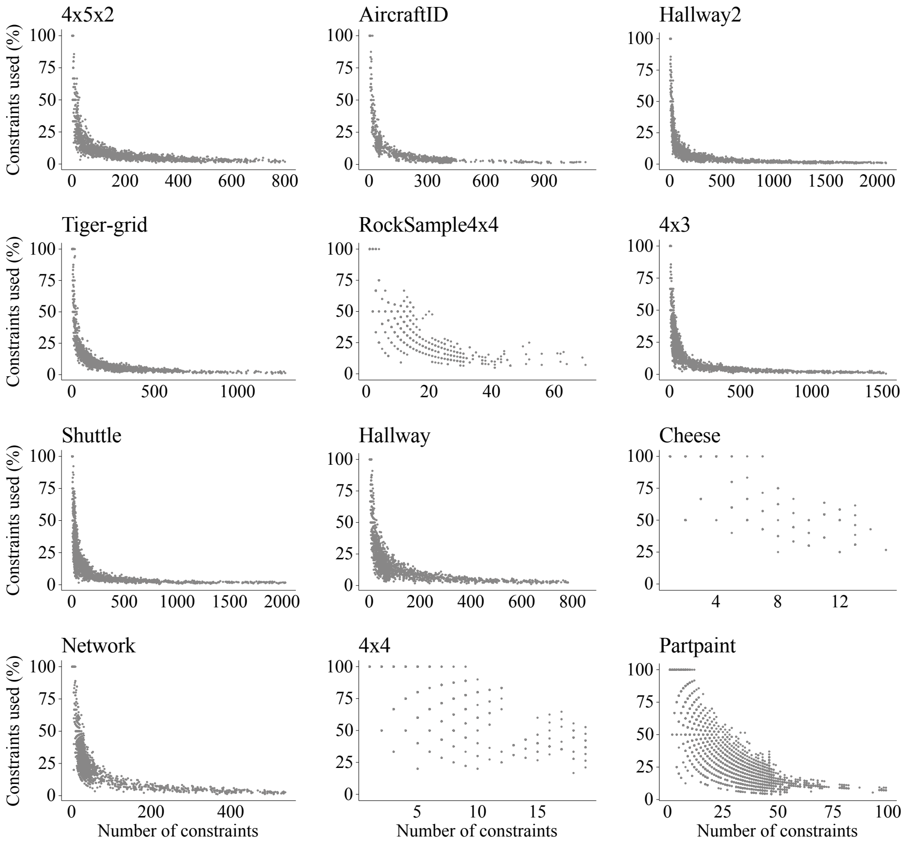


Figure 3.6: Constraints used, as a function of the total number of constraints

For each domain we take the largest value function V encountered during the execution of value iteration. Since we are not able to solve all domains to optimality, we take the largest value function encountered during the first 60 minutes of execution. We use this parsimonious value function V to create value functions $V_1, V_2, \ldots, V_\theta$ for which $|V_q| = q$ and $\texttt{prune}(V_q) = V_q$. Each value function V_q contains the first q vectors from V. Since $\texttt{prune}(V_q) = V_q$ holds for each V_q, we know that V_q does not contain dominated vectors. This property is important because it means that a pruning algorithm needs to solve q LPs to prune V_q.

In order to study the potential speedup we compare the standard vector pruning algorithm of White & Lark (Algorithm 2) and the vector pruning algorithm of White & Lark combined with constraint generation (Algorithm 2+4). For each V_q we measure the speedup that is obtained when using constraint generation for the LPs solved during vector pruning. The results are shown in Figure 3.7, which visualizes the speedup as a function of the size of the value function. On small

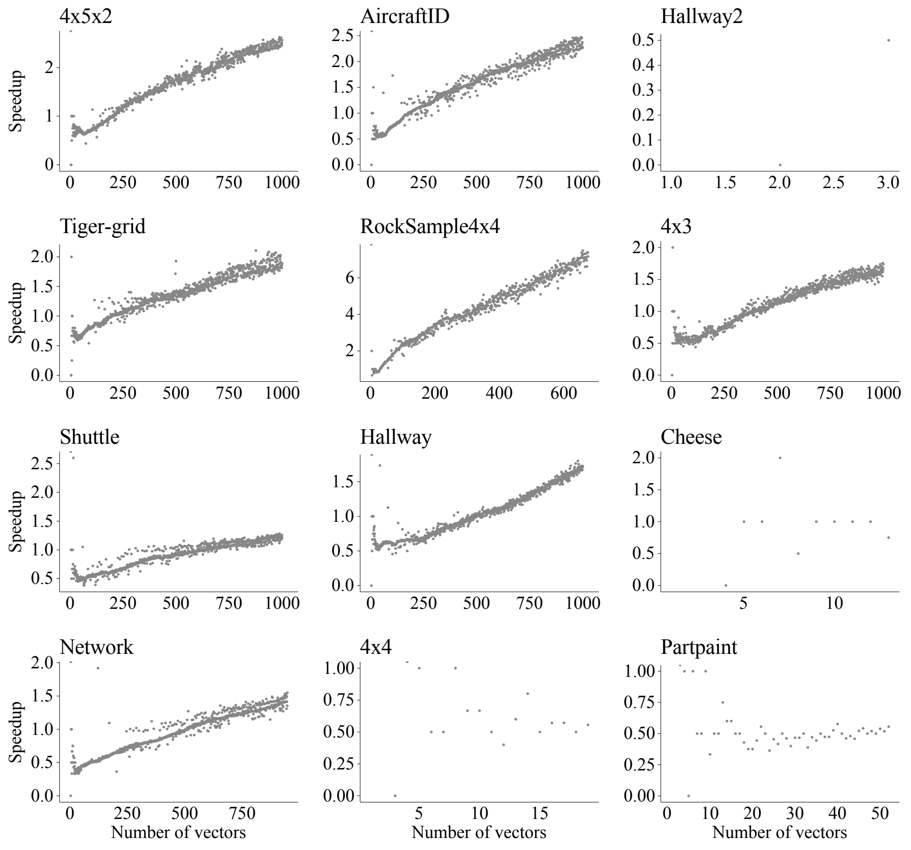


Figure 3.7: Vector pruning speedup as a function of value function size

value functions constraint generation makes vector pruning slightly slower. This supports our explanation from the previous section, in which we hypothesized that constraint generation introduces overhead if there are only a few constraints in the LPs. On larger instances the speedup is always positive, which confirms our expectation that constraint generation improves the performance of the vector pruning algorithm by White & Lark. Figure 3.8 shows the actual running times for the same experiment. Consistent with the results in Figure 3.7 we can see that incremental constraint generation improves the performance of White & Lark's vector pruning algorithm.

There are a few other alternative algorithms which we can involve in our comparison: Cheng's pruning algorithm (Cheng, 1988) and Skyline (Raphael and Shani, 2012). We do not consider Cheng's pruning algorithm because it enumerates corners of the belief space, which scales exponentially in the number of states. Skyline makes transitions in the so-called Skyline graph using an algorithm inspired by simplex for LPs. For Skyline we compare with the iterative variant, which is the fastest variant available, and the implementation is based on the source code provided by the original authors. The running times of this algorithm have been added to Figure 3.8.

We found that Skyline is outperformed by White & Lark's pruning algorithm and the variant based on constraint generation when pruning large value functions. In the original paper Skyline runs faster than White & Lark's algorithm for some instances. This result can be explained as follows. The original authors compared Skyline with a version of White & Lark's algorithm which is based on a manual implementation of simplex, rather than a state-of-the-art LP solver. This means that the version of White & Lark's algorithm used for their experiments is significantly slower than regular implementations that are used in practice. In our experiments we use a regular implementation of White & Lark's algorithm based on an external LP solver, because we think this gives the best comparison from a practical viewpoint.

To conclude, we found that incremental constraint generation improves the performance of the vector pruning algorithm by White & Lark. Moreover, we conclude that the resulting algorithm is currently the best-performing vector pruning algorithm for POMDPs.

Performance of incremental pruning

In our final experiment we show that integrating constraint generation in incremental pruning creates the fastest variant of incremental pruning for POMDPs. We do not consider other value iteration algorithms, because incremental pruning delivers superior performance compared to other exact POMDP algorithms (Cassandra, Littman, and Zhang, 1997). We implemented incremental pruning (IP) and its generalized variant (GIP), which we enhanced with incremental constraint generation (IP+ICG and GIP+ICG, respectively). We also compare with region-based incremental pruning algorithms (Feng and Zilberstein, 2004), abbreviated IBIP and RBIP, which exploit information about regions of the belief space when pruning vectors after computing the cross sum.

Solving the domains to optimality is typically not tractable due to the required running time and the required system memory. Therefore, we run the algorithms for one hour, after which their execution is terminated. We also terminate the algorithm if it reaches a solution with Bellman error 0.01 or lower. For each algorithm we measure the number of stages of value iteration, the Bellman residual error and the running time. In order to ensure a fair comparison, we compare the algorithms based on the stage they could all complete within the time limit. In Table 3.3 we report the stage number and the corresponding Bellman residual error. Furthermore, for each domain we provide the running time in seconds, which is the running time until the completion of the aforementioned stage. In

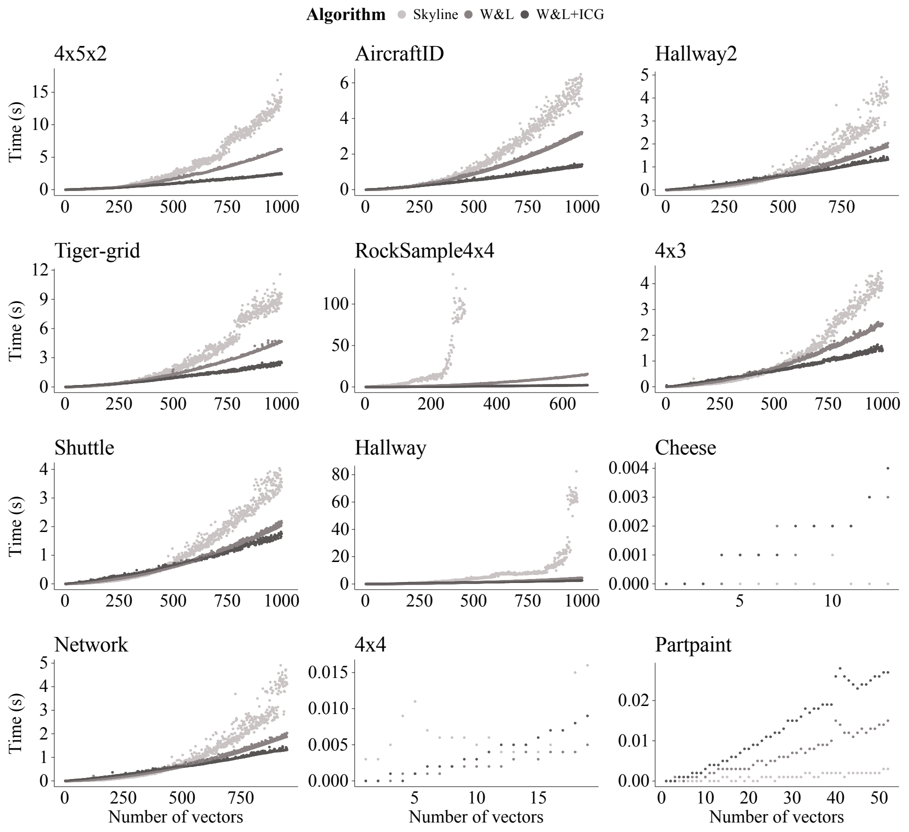


Figure 3.8: Runtime comparison for multiple pruning methods

some cases the reported running time is higher than 3600 seconds. This occurs if the time limit is reached while calculating the Bellman residual error. During this calculation we do not interrupt the algorithm. The domains Tiger-grid and Hallway2 are not included because the algorithms ran out of memory. The results computed before reaching the memory limit are not suitable for a fair comparison. IBIP encountered numerical instability when computing the first three stages of Hallway, but it ran longer than RBIP did, which means that the algorithm is slower in this particular case. For 4x3 we encountered minor numerical instabilities, and therefore we let FindBelief return the belief $\bar{b}$ if $\bar{d} > 0.0001$ in this domain.

Based on the results we conclude that vector pruning enhanced with incremental constraint generation significantly improves the performance of both standard incremental pruning, as well as the generalized incremental pruning algorithm. Incremental pruning is currently the standard pruning-based algorithm to solve POMDPs to optimality, and our results show that combining this algorithm with

	Stages	Error	IP	IP+ICG	GIP	GIP+ICG	IBIP	RBIP
4x5x2	16	0.13	3335.85	1925.10	3237.75	**1898.75**	3204.81	3597.36
AircraftID	6	77.38	22.52	10.62	15.84	**9.70**	37.33	16.09
RockS4x4	4	8.57	172.06	24.29	73.78	**22.92**	139.48	90.17
4x3	49	0.01	864.54	663.23	610.61	**503.10**	629.14	613.87
Shuttle	103	0.01	474.80	414.80	332.34	**314.90**	382.32	430.69
Hallway	3	0.38	180.54	104.57	75.14	**54.51**	>283.39	283.39
Cheese	58	0.01	0.83	0.82	0.83	**0.80**	0.87	1.48
Network	143	0.01	936.31	707.45	637.68	**551.50**	651.82	702.28
4x4	59	0.01	2.04	1.86	1.67	**1.66**	1.69	2.16
Partpaint	57	0.01	3.60	**3.40**	3.54	3.45	7.14	11.91

Table 3.3: Incremental pruning performance in seconds for a fixed number of stages

incremental constraint generation creates the fastest variant that is now available.

3.3.2. Bootstrapping experiments

In this section we study how bootstrapping of constraints influences the behavior and performance of the incremental pruning algorithm. Bootstrapping uses information from the previous iteration of value iteration, which means that we only execute experiments based on incremental pruning. In this section we do not consider experiments at the level of constraint generation and vector pruning. The structure of this section is as follows. First we present a collection of metrics which we can use to study the bootstrapping performance. After that, we describe multiple experiments in which we evaluate the bootstrapping technique based on these metrics.

Metrics for evaluation of bootstrapping

We evaluate our bootstrapping technique based on four metrics, which we informally introduce and motivate below. In the sections describing our experiments we provide further details and, if applicable, a formal definition of the metric.

1. **NumIter** – The bootstrapping procedure has been designed based on the insight that LP solutions get increasingly similar during the execution of value iteration, and a subset of LP constraints can be initialized based on the constraints that defined an optimal LP solution in a previous iteration. As a result, this would potentially decrease the number of additional iterations in which constraints are added before reaching an optimal LP solution again. Moreover, this would imply that the LP solver needs to (re)solve fewer master LPs. In general we hypothesize that bootstrapping reduces the number of

iterations in which a new constraint is added to the master LP. We measure this by counting the total number of iterations in which constraints are added, and we compare this with the version of the algorithm without bootstrapping.

2. **LPSuccess** – Bootstrapping of constraints can be considered successful if the `FindBeliefBLP` procedure runs fewer iterations than `FindBeliefICG` on the same LP. Ideally this is the case for all LPs solved during the execution of incremental pruning. In order to assess this, we measure the number of LPs in which `FindBeliefBLP` makes fewer iterations, which we express as a percentage of the total number of LPs solved. This also allows us to study how this effect changes over time as the solution converges towards optimality and LPs become increasingly similar.

3. **ConstraintsBinding**(β) – Ideally the bootstrapping procedure only initializes LP constraints which are actually binding constraints in the optimal solution of the LP. We can count the number of binding constraints, which we express as a percentage of the total number of constraints initialized based on bootstrapping. Due to minor differences in the LPs solved in successive iterations of value iteration we use a threshold β to decide if a constraint counts as a binding constraint, and this becomes a parameter of our metric.

4. **Overhead** – Bootstrapping introduces overhead due to the bookkeeping it performs during the execution of value iteration. Ideally, this overhead is only a small fraction of the total running time of incremental pruning. We can measure this by keeping track of the running time of the bootstrapping operations, which we express as a percentage of the total running time of the dynamic programming stages performed during the execution of incremental pruning.

In the next sections we proceed with more detailed descriptions of our experiments, in which we evaluate the bootstrapping performance based on the metrics 1–4. In all the experiments we execute incremental pruning with incremental constraint generation and bootstrapping for at most 60 minutes. In these experiments we did not include Hallway, Hallway2 and Tiger-grid because we could only execute a few iterations due to memory and storage limitations, and these iterations do not allow us to analyze the behavior of the bootstrapping procedure.

The experiments in which we evaluate metrics are based on the standard variant of incremental pruning, rather than the generalized variant. The reason is that the generalized variant invokes `FindBelief` based on a subset of U, and this subset choice is dependent on properties we cannot control, which would introduce noise in our experiments. The subset choice also converges during the execution of incremental pruning, and therefore it can be expected that bootstrapping also becomes effective in the generalized incremental pruning variant. This is confirmed by the experiments at the end of this section, where we evaluate both the standard and generalized variant of incremental pruning.

Constraint generation iterations

In our first experiment we study the bootstrapping performance by measuring the number of iterations of the constraint generation procedure. In order to estimate the potential reduction of the number of iterations, we run both `FindBeliefICG` and `FindBeliefBLP` and compare the number of iterations they execute. We let #iterations_ICG denote the number of iterations performed by the incremental constraint generation algorithm, and #iterations_BLP denotes the number of iterations performed by incremental constraint generation after constraints have been initialized by the bootstrapping procedure. The reduction of the number of iterations is calculated as follows:

$$-100\times\left(\frac{\#\text{iterations\_BLP}-\#\text{iterations\_ICG}}{\#\text{iterations\_ICG}}\right). \qquad (3.22)$$

With this metric a reduction of 20 percent means that the number of iterations decreased by 20 percent due to bootstrapping of constraints. We use the metric to measure the reduction for each iteration of value iteration, which allows us to visualize the reduction as a function of the Bellman error magnitude during algorithm execution.

The results are shown in Figure 3.9. As we can see, bootstrapping consistently leads to a reduction of the number of iterations performed by the incremental constraint generation procedure. Furthermore, in several domains the reduction increases when the Bellman error approaches zero, which is natural since solutions and hence LPs get increasingly similar during the execution of value iteration. For several domains (e.g., AircraftID and RockSample4x4) it was not possible to approach a low Bellman error within an hour, and value iteration only executed a few iterations, but we can still see that there is a reduction of the number of iterations in which constraints are added. Note that a reduction of 100 percent is not possible, because when solving a linear program it is always required to perform at least one iteration of `FindBeliefBLP` in which the LP is solved. We conclude that, as we expected, bootstrapping reduces the number of iterations in which the incremental constraint generation procedure adds constraints.

LPs with fewer iterations

In our second experiment we count the number of LPs for which the `FindBeliefBLP` procedure runs fewer iterations than `FindBeliefICG`. We let #linear_program denote the number LPs solved, which corresponds to the number of calls to `FindBelief` (either ICG or BLP). We let #linear_program_reduced denote the number of LPs in which the version with bootstrapping runs fewer iterations,

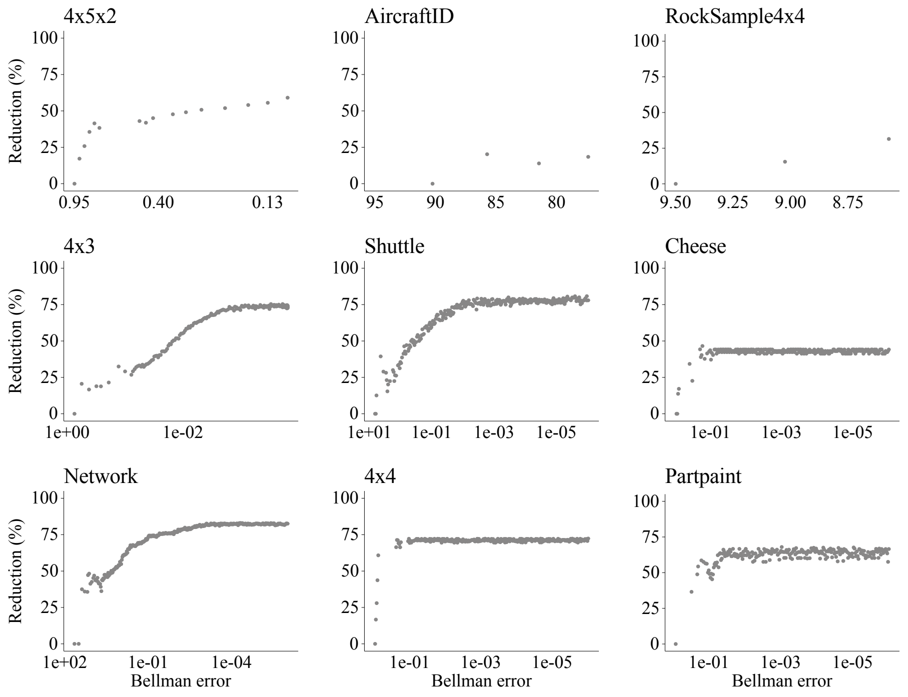


Figure 3.9: Reduction of the number of iterations as a function of the Bellman error

which can be computed by running both versions of `FindBelief`. The fraction of LPs with fewer iterations of constraint generation is expressed as follows:

$$100 \times \left(\frac{\#\text{linear\_program\_reduced}}{\#\text{linear\_program}} \right) \tag{3.23}$$

Similar to the previous experiment, we can compute the fraction for each iteration of value iteration individually. If the fraction is equal to 80 percent, for example, then it means that bootstrapping leads to a reduction of the number of iterations in 80 percent of the calls to `FindBelief`.

The results are shown in Figure 3.10, which visualizes the percentage LPs with fewer iterations as a function of the Bellman error. The experimental results confirm our initial expectation that bootstrapping of constraints leads to a reduction of the number of additional iterations performed by `FindBeliefBLP`. As can be seen in several domains (e.g., Network, Shuttle and 4x5x2), the number of LPs with fewer iterations increases when the Bellman error goes down. This is what we initially expected, because if the Bellman error approaches zero, then LPs become increasingly similar and our algorithm identifies such similar constraints from a previously-solved LP. This reduces the number of additional iterations in which constraints are added. During early stages of value iterations the LPs

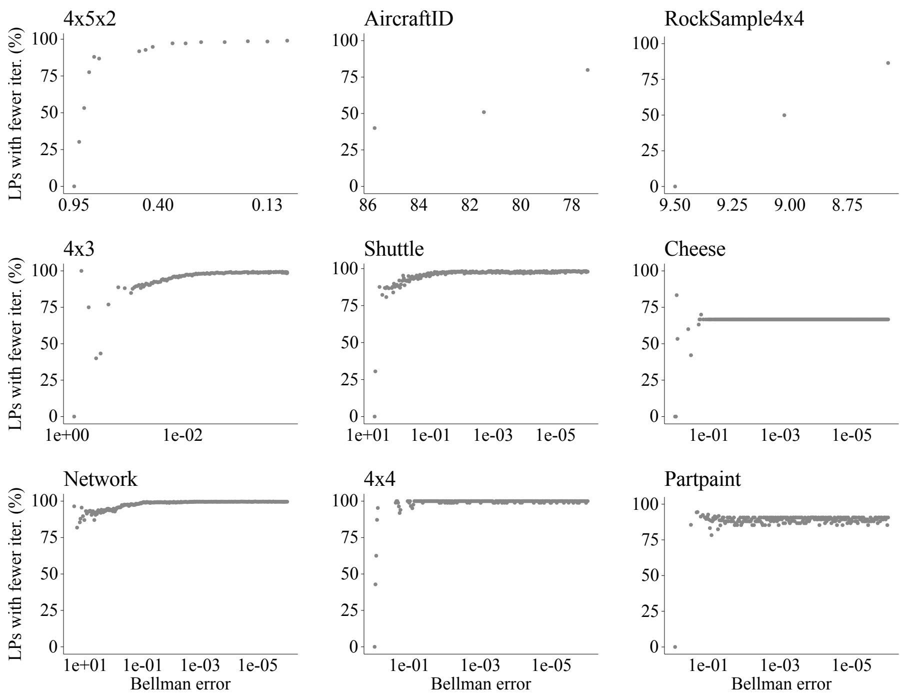


Figure 3.10: Number of LPs in which bootstrapping leads to fewer iterations, as a function of the Bellman error

are not necessarily similar, which means that `FindBeliefBLP` may execute more iterations than `FindBeliefICG`. This can also be recognized in the graphs in the figure, which show that the percentage is typically lower during early stages of value iteration. This result also aligns with our initial expectations regarding the performance of the bootstrapping procedure.

Binding constraints

In the third experiment we investigate whether the bootstrapping technique initializes constraints that are actually binding constraints for the optimal LP solution. Recall that the method `FindBeliefBLP` in Algorithm 5 initializes a constraint $d^* \leq (w - \hat{u}) \cdot b$ for each $\hat{u} \in \mathcal{C}_{init}$. The optimal solution is denoted by the belief $\bar{b}$ together with the optimal objective value $\bar{d}$. The constraint $d^* \leq (w - \hat{u}) \cdot b$ is called binding if it holds that

$$\left|(w - \hat{u}) \cdot \bar{b} - \bar{d}\right| \leq \beta, \tag{3.24}$$

in which the parameter β serves as a small threshold. The latter is necessary because minor changes in value functions make it highly unlikely that the difference between $(w - \hat{u}) \cdot \bar{b}$ and $\bar{d}$ becomes exactly zero. We let #constraints_total denote

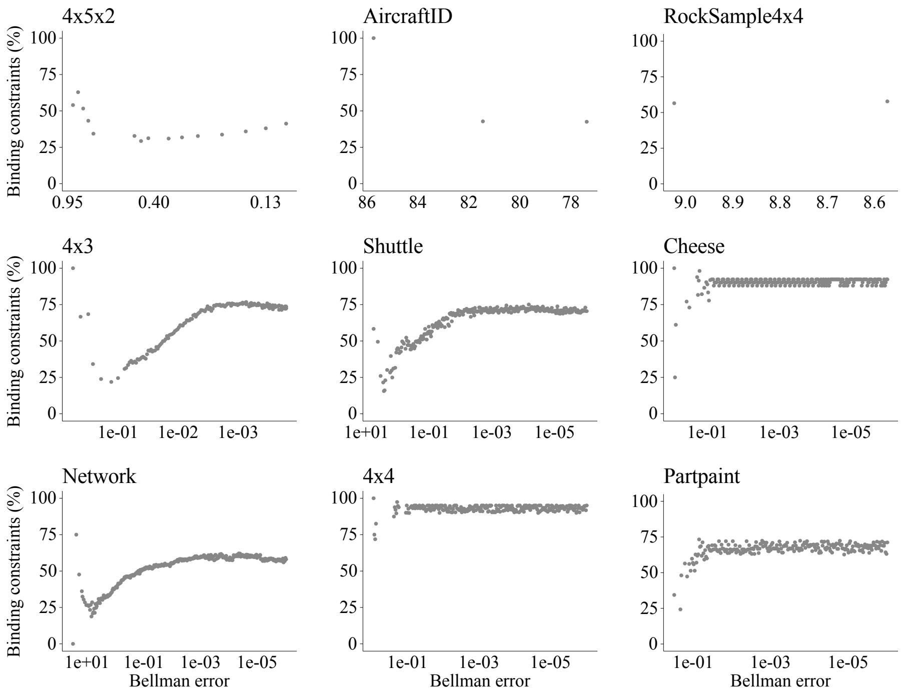


Figure 3.11: Number of constraints initialized based on bootstrapping which are also binding constraints for the optimal LP solution, as a function of the Bellman error

the total number of constraints initialized during bootstrapping (i.e., the total number of vectors in the $\mathcal{C}_{init}$ sets) and #constraints_binding denotes the number of constraints that is actually binding, based on the aforementioned criterion. The percentage of binding constraints now corresponds to:

$$100 \times \left(\frac{\#\text{constraints\_binding}}{\#\text{constraints\_total}} \right). \tag{3.25}$$

As an example, we consider an LP for which bootstrapping initializes 10 constraints (i.e., $|\mathcal{C}_{init}| = 10$), then 60 percent means that 6 out of 10 constraints are binding constraints in the optimal LP solution. Besides individual LPs we can also apply this metric to iterations of value iteration, in which we count across multiple LPs.

The results of our experiment are shown in Figure 3.11 for $\beta = 0.001$, which visualizes the fraction of binding constraints for each iteration of value iteration. As can be seen, the results confirm that the bootstrapping procedure initializes constraints which turn out to be binding constraints in the optimal solution. This is an important result, because it indicates that our bootstrapping procedure correctly identifies binding constraints from the previous iteration of value iteration. In the graphs it can be seen that the percentage does not approach 100 percent in several domains, which can be explained as follows. If there are multiple cache entries for the same vector w, then we initialize all the corresponding constraints (see implementation detail Section 1.3). This means that it initializes binding constraints from the previous iteration for the same vector, but also a few more constraints, which are not necessarily binding. Finally, the experiment also confirms that the fraction of binding constraints increases when the Bellman error approaches zero, similar to previous experiments. This indicates that our bootstrapping procedure starts to perform better when the Bellman error starts to decrease and value functions become increasingly similar.

Overhead due to bookkeeping

The bootstrapping procedure executes additional operations to store, identify and retrieve vector sets, which requires additional running time. In our final experiment we study the additional running time required for bookkeeping. We can measure this overhead as a fraction of the total running time of exact value iteration. We let DP_time denote the total elapsed time of exact value iteration, and bootstrap_time denotes the total running time of the bootstrapping operations executed so far. The bookkeeping time can now be expressed as a fraction of the total running time:

$$100 \times \left(\frac{\text{bootstrap\_time}}{\text{DP\_time}} \right), \tag{3.26}$$

such that 5 percent means that 5 percent of the total running time is spent on additional operations required for bootstrapping. In contrast to the previous experiments, we do not compute the overhead for individual iterations. Instead, we report the total overhead incurred until the current iteration of value iteration. This is necessary because measurements based on individual iterations of value iteration have a runtime close to 0.

Our results are shown in Figure 3.12, in which each dot indicates the total bookkeeping from the start of the algorithm execution. As can be seen, the total overhead introduced due to bookkeeping is negligible compared to the total running time of value iteration. This result is important, because it means that our bootstrapping techniques are relatively cheap to execute. Furthermore, if the runtime gains are larger than the additional bookkeeping time, then it means that our bootstrapping techniques potentially accelerate value iteration. We study this in the experiment in the next section, in which we compare multiple variants of incremental pruning.

Performance of incremental pruning

In the final experiment we test whether bootstrapping improves the performance of incremental pruning for POMDPs. The setup of the experiment is identical to the incremental pruning experiment that we executed for constraint generation.

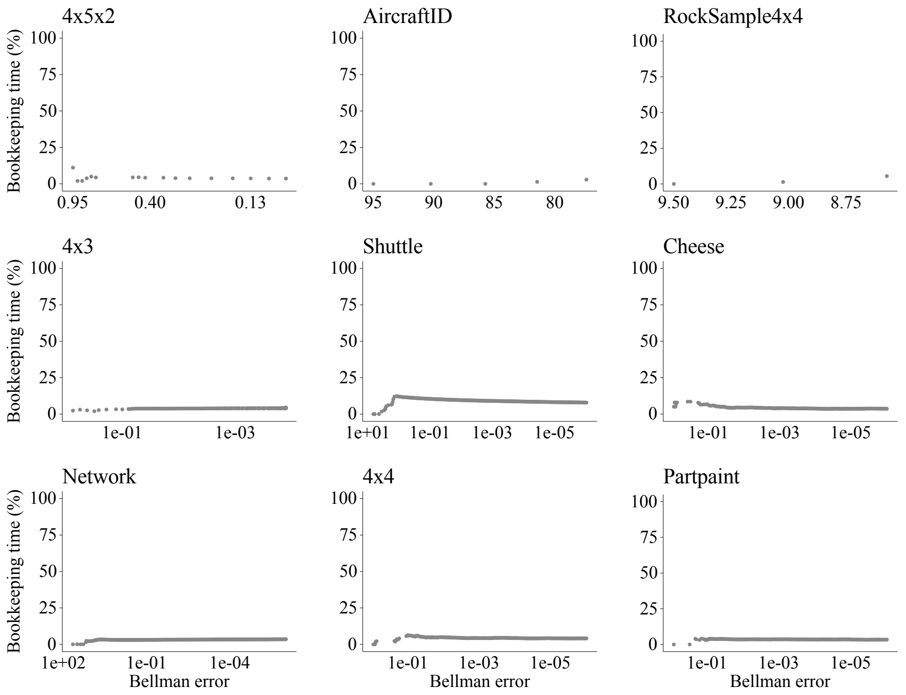


Figure 3.12: Total bootstrapping time as a percentage of the total elapsed time during execution of value iteration

Intuitively, we expect that bootstrapping improves the performance of incremental pruning and incremental constraint generation if incremental pruning runs many iterations in which bootstrapping can be formed, because small gains in individual iterations accumulate into a larger overall gain. Additionally, we also expect that it performs well in case the algorithm approaches a low Bellman error, because that means that LP solutions become increasingly similar and in that case bootstrapping becomes more effective.

The results of our experiment are shown in Table 3.4. The table shows the runtimes of incremental pruning combined with incremental constraint generation for solving LPs, which we copied from Table 3.3. In addition, the columns IP+BLP and GIP+BLP show the runtimes of the algorithms which include our bootstrapping procedure. We conclude that bootstrapping improves the performance of the standard incremental pruning algorithm (IP) and the generalized variant (GIP) for almost all the domains. In domains with a significant number of iterations and a low Bellman error we can see that bootstrapping effectively reduces the running time. If there are many iterations, then small gains in single iterations accumulate into a larger gain, which is the case in, e.g., the domains Network and Shuttle.

	Stages	Error	IP+ICG	IP+BLP	GIP+ICG	GIP+BLP	Speedup
4x5x2	16	0.13	1925.10	1861.61	1898.75	**1828.72**	1.75
AircraftID	6	77.38	10.62	10.16	9.70	**9.48**	1.67
RockS4x4	4	8.57	24.29	24.82	**22.92**	24.72	3.22
4x3	49	0.01	663.23	547.33	503.10	**438.23**	1.40
Shuttle	103	0.01	414.80	377.53	314.90	**288.75**	1.15
Hallway	3	0.38	104.57	107.85	**54.51**	54.96	1.38
Cheese	58	0.01	0.82	0.82	**0.80**	0.84	1.04
Network	143	0.01	707.45	514.51	551.50	**479.03**	1.15
4x4	59	0.01	1.86	1.78	1.66	**1.62**	1.03
Partpaint	57	0.01	3.40	3.46	3.45	**3.31**	1.07

Table 3.4: Comparison incremental pruning with bootstrapping and without bootstrapping, with runtimes in seconds, and total speedup compared to former state of the art

In a few domains the bootstrapping techniques did not improve the performance. For example, in the domains RockS4x4 and Hallway there is no improvement, which can be explained by observing that the algorithm did not converge to a solution with a low Bellman error, and under these circumstances bootstrapping is not very effective yet. In the Cheese domain GIP+BLP is slightly slower than GIP+ICG, but it should be noted that the value functions in this domain have at most 14 vectors, and when pruning small value functions it is not beneficial to use incremental constraint generation because this creates additional overhead, as discussed in the constraint generation experiments.

Finally, combined with the results from Section 3.3 we can conclude that incremental constraint generation combined with bootstrapping creates the fastest variant of incremental pruning for POMDPs. The column Speedup in the table indicates the speedup obtained by our methods compared to the existing state of the art (either GIP, IBIP or RBIP). We calculated this speedup by comparing the fastest existing algorithm and the fastest variant which includes our techniques. As can be seen, the speedup is consistently higher than 1, which means that our techniques consistently outperform the current state of the art.

3.4. Related work

Region-based pruning methods have been developed (Feng and Zilberstein, 2004), which exploit the structure of the cross sum when checking whether a vector should be part of the parsimonious representation of the cross sum. Rather than enumerating all vectors and pruning afterwards, as in our work, region-based pruning methods use several LPs to detect which vectors of the cross sum should be computed. Such an approach does not necessarily reduce the total number of

LPs solved, but the LPs it solves typically have fewer constraints. From a more general point of view, region-based pruning methods aim to improve the performance of incremental pruning by introducing alternative LP formulations which check vector dominance, while our line of work aims to improve the algorithm by introducing techniques to solve the existing LPs more efficiently. A key advantage of our techniques is that they improve vector pruning in general, while the techniques by Feng and Zilberstein (2004) assume that vector pruning is executed on a cross sum. Another recent pruning technique is Skyline (Raphael and Shani, 2012), which traces the upper-surface of the value function. Compared to our work and compared to the work by Feng and Zilberstein (2004) the algorithm is entirely different, because it does not build upon the initial concepts introduced by White & Lark. Similar to our techniques, Skyline does not assume that the value function considered is the result of the cross sum operation. Our experimental evaluation has shown that vector pruning based on incremental constraint generation and bootstrapping outperforms both region-based pruning methods and Skyline.

The approximate POMDP algorithm α-min (Dujardin, Dietterich, and Chadès, 2015) also uses a notion of incremental construction of constraint sets. The algorithm uses a mixed-integer problem in which so-called facets are generated incrementally, which resembles constraint generation. Our incremental constraint generation and bootstrapping procedure selects constraints from a known set of constraints, whereas the facets in α-min are used to approximate a set of constraints that is initially unknown. The latter is computationally more difficult, and both ICG and BLP do not need to rely on such a method since the constraint set is finite and already known.

Our work is related to decomposition approaches for linear programs, such as row and column generation (Benders, 1962; Gilmore and Gomory, 1961). Rather than solving an LP directly, such approaches decompose an LP into smaller parts to improve tractability of solving. Algorithm 4 has been derived using such a decomposition technique. Row and column generation also found applications in Factored MDPs (Guestrin and Gordon, 2002) and security games (Jain et al., 2010), as well as heuristic search for stochastic shortest path problems (Trevizan et al., 2016). The latter uses heuristics to guide how one individual LP should be expanded with new variables and new constraints. An important difference in our work is that we bootstrap from a previous LP when initializing constraints in a new LP, rather than just studying one individual LP. Detecting dominated vectors also resembles detecting irrelevant constraints in linear programs (Mattheiss, 1973). It is important to note that such constraints never exist in our LPs, because they only contain constraints corresponding to vectors which dominate in a region of the belief space. This means that there is always a region of the belief space in which a constraints is binding.

3.5. Conclusions

In this chapter we studied the linear programming formulation that plays a crucial role in exact POMDP value iteration algorithms which require pruning of value functions, such as the incremental pruning algorithm (Cassandra, Littman, and Zhang, 1997). In particular, this linear program is used to check whether a vector improves a given value function, and typically it is solved many times during the execution of exact value iteration. It turns out that solving the linear programs represents a significant part of the running time of exact value iteration, which means that a running time reduction for an individual linear program potentially leads to a major running time reduction for exact value iteration in general. Based on this intuition we designed and analyzed two techniques which can be used to solve the linear programs more efficiently, without affecting the solution computed by exact value iteration.

Our first contribution is a constraint generation procedure which generates constraints incrementally, rather than solving the linear program immediately with all constraints included. This is beneficial because typically there are only a few constraints which are actually binding constraints in the optimal solution, and our constraint generation algorithm only uses a small fraction of the constraints from the original linear program, which leads to a significant speedup.

Our second contribution is a bootstrapping procedure which further improves the constraint generation procedure. We observed that the solutions to linear programs become increasingly similar during the execution of value iteration due to the convergence of the value functions towards a fixed point. Our bootstrapping procedure exploits this property and initializes a subset of constraints based on similar linear programs solved in the previous iteration of exact value iteration. If this subset is close or identical to the set of constraints which binds the optimal solution of the new LP, then it leads to a reduction of the number of iterations performed by the constraint generation procedure.

A series of experiments has shown that constraint generation effectively reduces the running time required for solving linear programs. Moreover, our experiments confirm that this also leads to a reduction of the running time required for vector pruning, which in turn leads to a reduction of the running time of the incremental pruning algorithm. Our bootstrapping procedure reduces the running time of incremental pruning even more, and the resulting algorithm is the fastest pruning-based value iteration algorithm to solve POMDPs optimally.

Future applications and research directions

There are several research directions that can be pursued in the future, which we describe below in more detail. We also describe other applications in which the improved pruning approach can be applied.

Several other decision making algorithms rely on the same linear program as the one we studied extensively in this chapter, which means that our line of work can be used to accelerate other algorithms in the future. For example, Convex Hull Value Iteration for Multi-Objective MDPs also prunes vector sets after computing a cross sum (Barrett and Narayanan, 2008), and similar to POMDPs the linear programs that are solved become increasingly similar. In our paper describing the bootstrapping method we already show that incremental constraint generation and bootstrapping also improves the performance of this value iteration algorithm (Roijers, Walraven, and Spaan, 2018). This means that our work can also be used to obtain faster algorithms for multi-objective decision making. Moreover, we expect that the bootstrapping ideas from our work can also be used in the context of reinforcement learning for Multi-Objective MDPs (Hiraoka, Yoshida, and Mishima, 2009; Wiering, Withagen, and Drugan, 2014), which also relies on policy computation. Our techniques can also be applied in the context of exact dynamic programming for Decentralized POMDPs (Oliehoek and Amato, 2016), which relies on the same LP when pruning dominated sub-tree policies. However, due to the computational complexity of this problem we expect that solving Decentralized POMDPs to optimality remains prohibitive in practice.

Reinforcement learning algorithms for zero-sum Markov games iteratively update a value function and policy based on interaction with the environment (Littman, 1994). During each iteration of the learning algorithm the value function and the associated policy are updated, which requires linear programming. This linear program is solved many times and it also becomes increasingly similar during execution. This raises the question whether the solutions to previously-solved linear programs can be used to accelerate learning. Our incremental constraint generation procedure does not immediately apply since the linear program is not the same, but it may be possible to derive a similar technique to accelerate solving linear programs.

Our incremental constraint generation procedure and bootstrapping techniques exploit the property that solutions become increasingly similar. This is a general property of the POMDP formalism and the solution algorithm, but we do not exploit any additional information about the actual problem that is solved. As a long-term research direction we envision exploiting characteristics of the POMDP domain, in such a way that we can reason about areas of the belief space in which vectors are potentially dominating. Based on such an approach it may be possible to reduce the number of terms in the LP constraints, because intuitively we expect that several parts of the belief space do not need to be considered. Additionally, within this research line it may be possible to exploit the factored structure of the POMDP model to create smaller LPs.

This chapter entirely focuses on exact solution methods, which are computationally demanding to execute. It should be noted that the speedup achieved by incremental constraint generation and bootstrapping may not be sufficient to make exact policy computation tractable in large domains. However, the techniques from this chapter can also be used to improve existing approximate POMDP algorithms. For example, the approximate algorithm EVA computes solutions within a predefined error bound and operates based on a subset of vectors (Varakantham et al., 2007). EVA relies on almost the same linear program as exact value iteration for POMDPs, which means that our techniques are potentially useful in the approximate setting as well.

4 Approximate planning for finite-horizon POMDPs

The POMDP model and the corresponding solution algorithms have been studied extensively in academic literature. Most existing work focuses on the infinite-horizon variant of the model which includes discounting of reward. However, there are multiple domains in which a finite time horizon without discounting is required, and this raises the question how such finite-horizon problems should be treated. In this chapter we start with a discussion of finite-horizon problems and infinite-horizon problems. After that, we provide an overview of several approaches which may be used to solve finite-horizon problems, and we explain why it is not desirable to use them. In the second part of the chapter we describe FiVI[1], an anytime approximate algorithm for finite-horizon POMDPs which unifies ideas from state-of-the-art algorithms developed for infinite-horizon POMDPs.

4.1. Planning horizons and discounting

Finite-horizon POMDPs are used in domains where a policy is executed during a finite number of time steps. As an example we consider an EV charging provider which optimizes day-to-day operations based on finite-horizon forecasts of, e.g., electricity price and charging demand. In such domains it can be the objective to charge a fleet of EVs as cheap as possible while accounting for the uncertainty in arrival time and demand. It is natural to compute a policy which maximizes the

[1]This chapter is based on an article that appeared in the Journal of Artificial Intelligence Research (Walraven and Spaan, 2019).

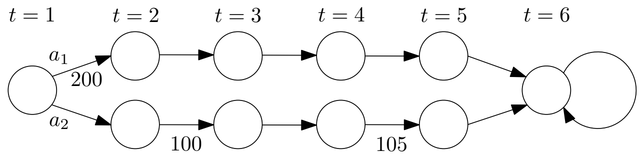


Figure 4.1: POMDP in which discounting causes suboptimality

expected sum of reward:

$$E\left[\sum_{t=1}^{h} r_t\right], \tag{4.1}$$

in which we intentionally count from $t = 1$ rather than $t = 0$, such that there are h steps in total. The term r_t denotes the reward collected at time t.

Infinite-horizon POMDPs, on the other hand, are used in problem domains where control policies have to be executed infinitely long. As an example, we consider an elevator control problem (Crites, 1996), in which the control policy needs to ensure that people get moved to the right floor in a short amount of time. From a decision making point of view this problem is partially observable, because pressing the button to request an elevator does not provide information about the destination floor. Short-term performance is important because passengers do not want to wait too long, and the notion of a finite horizon is not suitable because new passengers can arrive at any point in time in the future. For infinite-horizon problems with discounting the following optimality criterion is used:

$$E\left[\sum_{t=1}^{\infty} \gamma^{t-1}\, r_t\right], \tag{4.2}$$

in which the discount factor γ is used as a weight for the collected reward. The discount factor ensures that short-term reward is considered more important than reward received much later in time.

Although discounting can be justified from an application point of view in several domains, in many cases it is only used for mathematical convenience (Hansen, 2007). Discounting can be convenient because it ensures that the sum of an infinite number of rewards, as shown in Equation 4.2, becomes equivalent to the sum of a finite number of rewards. This means that the expectation becomes a well-defined and finite sum, and it means that it becomes possible to solve infinite-horizon problems by considering only a finite number of time steps in the future.

4.2. Strategies for solving finite-horizon problems

We start with an overview of approaches for solving finite-horizon POMDPs. For each approach we explain how existing techniques for MDPs and infinite-horizon

POMDPs can be potentially applied, and we argue why these techniques have several limitations when solving finite-horizon problems.

The first approach we discuss treats the POMDP as a fully-observable MDP. Since the number of reachable beliefs in a finite-horizon POMDP is finite, it is possible to enumerate these beliefs prior to planning. Recall from Chapter 2 that POMDP belief states provide a Markovian signal for a POMDP planning task. Therefore, after belief enumeration it is possible to solve a regular MDP defined in terms of belief states rather than actual states. This approach is well-defined and it provides an optimal finite-horizon POMDP policy, but unfortunately it is often intractable due to the large number of beliefs that needs to be enumerated, which is at most $(|A||O|)^h$.

A second approach based on infinite-horizon algorithms would simply compute an infinite-horizon policy to take decisions in a finite-horizon problem. This is straightforward, because one can assume a discount factor, after which an infinite-horizon algorithm is invoked to obtain a policy. There are two disadvantages associated with this approach. First, invoking an infinite-horizon algorithm leads to undesirable effects if the algorithm thinks that reward can be collected late in time, whereas execution ends early due to the finite time horizon. For example, if a policy has been optimized under the assumption that high reward can be collected after 20 steps, then the policy is unlikely to be optimal if execution ends after 5 steps. The second disadvantage of the approach are the undesirable effects due to the discount factor that is assumed, which we illustrate using an example. We consider a POMDP with fully-observable states and deterministic state transitions, as shown in Figure 4.1. In the initial state, the agent chooses either action a_1 or a_2, leading to either the top or bottom trajectory. The numbers below the transitions correspond to reward, and transitions without a number have zero reward. When casting the problem to an infinite-horizon problem with $\gamma = 0.95$, then a_1 is optimal since it gives expected reward $0.95^0 \cdot 200 = 200$, while a_2 gives expected reward $0.95 \cdot (100 + 0.95^2 \cdot 105) = 185.02$. However, a_2 is optimal for a finite-horizon problem where all rewards are equally important, because the bottom trajectory gives reward 205 while the top trajectory gives 200. This shows that casting a finite-horizon problem to an infinite-horizon problem with discounting can lead to suboptimal policies.

A third strategy augments the POMDP with a time state variable as part of the state description. This means that states become time-indexed, and a trap state is entered at the end of the horizon, resulting in a model with $|S| \times h + 1$ states. More efficient encodings are possible if not all states are reachable during all steps, but in general we can conclude that this strategy does not scale well if a large number of time steps needs to be considered. Although the increase of the model size is linear in the number of time steps, the augmented POMDP model and solution representations (e.g., alpha vectors) quickly become too large, which significantly increases the running time and memory requirements of POMDP algorithms. Augmenting states with a time state variable is not sufficient to obtain a finite-horizon policy. In addition, it is required to assume $\gamma = 1$, but this assumption leads to implementation issues and undesirable effects in several state-of-the-art algorithms. This is further discussed in the next section.

A fourth approach would interpret the aforementioned augmented POMDP with a trap state as a stochastic shortest path problem for Goal POMDPs. The Goal POMDP formulation assumes that the POMDP has a fully-observable goal state that cannot be left, which is the case when defining the POMDP with a trap state at the end of the time horizon. Real-Time Dynamic Programming (RTDP) can be used to find solutions to such problems and it has been generalized to POMDPs as well (Bonet and Geffner, 2009). The resulting RTDP-Bel algorithm does not include discounting and it can potentially be adapted to support time-dependent value functions. However, due to discretization of belief states it does not provide performance guarantees and it does not keep track of an upper bound on the optimal value function. Existing RTDP extensions for MDPs do account for upper bounds (Smith and Simmons, 2006), but to the best of our knowledge these upper bounds have not been applied in RTDP for problems with partial observability.

A fifth approach for solving finite-horizon problems consists of an adaptation of the algorithm α-min (Dujardin, Dietterich, and Chadès, 2015). This algorithm keeps track of separate value functions for each time step, and it imposes the additional restriction that there should be a maximum of N vectors for each time step. The algorithm may be applied without this restriction and with a low gap tolerance, but in that case it starts to invoke a large number of mixed-integer linear programs in order to expand the belief sets, which are expensive to solve and this leads to scalability problems. Other adaptations of α-min provide more scalability but they do not provide any performance guarantees (Dujardin, Dietterich, and Chadès, 2017).

Based on our discussion we can conclude that there are several straightforward approaches for solving finite-horizon POMDPs without discounting, but all these approaches are affected by either scalability problems or undesirable effects. In the next section we describe why state-of-the-art POMDP algorithms cannot be used for finite-horizon models with a discount factor that is equal to 1.

4.3. Discarding the discount factor

As noted in the previous section, the application of infinite-horizon algorithms to finite-horizon formulations with time-indexed states requires a discount factor γ that is equal to 1. Unfortunately, many state-of-the-art algorithms for infinite-horizon problems do not support such a discount factor, and they cannot be modified easily without changing the characteristics. Next, we discuss for each algorithm why it cannot be used for finite-horizon planning with the discount factor $\gamma = 1$. We also discuss whether the algorithms converge to optimality, and whether they compute an upper bound on an optimal solution.

GapMin (Poupart, Kim, and Kim, 2011) is a point-based value iteration algorithm which computes both lower bounds and upper bounds on the optimal value function, and it converges in the limit to an optimal POMDP solution. The algorithm contains several subroutines which require $\gamma < 1$, and the algorithm is not well-defined in case we set $\gamma = 1$. Assuming a discount factor $\gamma < 1$ that is arbitrarily close to 1 leads to a situation in which many subroutines have slow convergence, which is undesirable. Without significant adaptations GapMin cannot be used with $\gamma = 1$.

The point-based value iteration algorithms SARSOP (Kurniawati, Hsu, and Lee, 2008) and HSVI (Smith and Simmons, 2005) follow a similar approach as GapMin, in the sense that they also incrementally expand a set of belief points based on heuristic search. They also produce an upper bound on the optimal value function, and the algorithms converge to optimality in the limit. The backups and upper bound updates performed by the algorithms are well-defined for $\gamma = 1$. However, the initialization of lower bounds and upper bounds require $\gamma < 1$ and therefore it is necessary to initialize them differently. Similar to GapMin, without adaptations both algorithms cannot be used with $\gamma = 1$.

Perseus (Spaan and Vlassis, 2005) is a randomized point-based value iteration algorithm which iteratively performs backups on a set of randomly-sampled belief points. The initialization of the lower bound requires $\gamma < 1$, and therefore this also requires modification. The algorithm does not keep track of an upper bound on the optimal value function, and it provides no guarantees on performance, which means that it is not guaranteed to converge to optimality.

The original PBVI algorithm (Pineau, Gordon, and Thrun, 2003) executes backups on a belief set that is expanded incrementally. The algorithm can be interpreted as an anytime algorithm, and for reaching an optimal solution this boils down to full enumeration of the reachable belief space. The bounds on the worst-case error assume a discount factor $\gamma < 1$, but the algorithm itself can be used without discounting. In general this is still not desirable because the number of belief points is potentially large, and it has been shown empirically that the algorithms GapMin, SARSOP, HSVI and Perseus typically outperform the original PBVI algorithm.

Exact value iteration supports the discount factor $\gamma = 1$ and it always computes an optimal policy by definition. However, due to its limited scalability it is not desirable to use the algorithm for problems with large state spaces, which would be the case if we use a formulation with time-indexed states and a trap state.

An overview of the algorithm characteristics is presented in Table 4.1, which compares the algorithms in term of their ability to compute an upper bound, conver-

Algorithm	Upper bound	Convergence to optimality	Supports $\gamma = 1$
GapMin	✓	✓	
SARSOP	✓	✓	
HSVI	✓	✓	
Perseus			
PBVI		✓	✓
Exact VI		✓	✓
RTDP-Bel			✓
FiVI	✓	✓	✓

Table 4.1: Comparison of infinite-horizon algorithms and FiVI

gence to optimality, and immediate support for discount factors $\gamma = 1$. RTDP-Bel has also been included in the table, which we briefly discussed in Section 4.2. As can be seen, there is no existing algorithm which has all three properties simultaneously. In contrast, the algorithm FiVI presented in the next section does have all these properties, as shown in the table. The algorithm unifies the desirable characteristics of GapMin, SARSOP and HSVI in such a way that we obtain a finite-horizon point-based value iteration algorithm which converges to optimality and it also computes both lower bounds and upper bounds.

4.4. Finite-horizon point-based value iteration

In this section we describe FiVI, a point-based value iteration algorithm for solving finite-horizon POMDPs. The algorithm unifies techniques and concepts from existing state-of-the-art point-based value iteration algorithms and it provides attractive convergence characteristics and optimality guarantees. This section describes the solution representations used by FiVI, the actual algorithm, its theoretical properties and relations to existing algorithms.

We start with an overview of the high-level structure of the solution computed by the FiVI algorithm in Section 4.4.1, based on time-dependent value functions and time-dependent backups. In Section 4.4.2 we explain how time-dependent value upper bounds can be obtained in a finite-horizon setting using the sawtooth approximation. A full description of the FiVI algorithm is provided in Section 4.4.3, which includes the aforementioned value functions and upper bounds. The convergence and optimality characteristics of the FiVI algorithm depend on the belief points used for computing the value functions and upper bounds. This is the topic of the final section, in which we provide an heuristic search procedure for finding beliefs, as well as a theoretical motivation which explains that FiVI converges to an optimal solution.

4.4.1. Time-dependent value functions and backups

Point-based value iteration algorithms compute value functions represented by a finite set of vectors, as introduced in Chapter 2. For infinite-horizon problems it suffices to keep track of one individual value function V, which represents the stationary policy that can be used to choose actions. In the finite-horizon case the policy is non-stationary, and in general it is no longer possible to encode the policy using just one value function. In our FiVI algorithm we use time-dependent value functions $\mathcal{V}_t$, in which t refers to a time step ranging from 1 to h. Note that we use $\mathcal{V}_t$ rather than V_t to avoid notation conflicts with infinite-horizon value iteration. The value function $\mathcal{V}_t$ is represented by a finite set of vectors Γ_t and it can be defined as follows:

$$\mathcal{V}_t(b) = \max_{\alpha \in \Gamma_t} b \cdot \alpha, \tag{4.3}$$

such that $\mathcal{V}_t(b)$ corresponds to the expected reward collected when executing the policy induced by the value functions $\mathcal{V}_t, \ldots, \mathcal{V}_h$ starting from belief b.

The vectors that constitute a value function can be computed using a point-based backup operator, as defined by Equation 2.41. In the infinite-horizon case the backups are executed on beliefs in a set B, as shown in Equation 2.50. Similar to the value functions, the belief sets can be made time dependent for the finite-horizon case, such that Γ_t is computed using the beliefs in the set B_t. In our algorithm we need to keep track of upper bounds $\bar{v}$ associated with beliefs b, and therefore the elements of the set B_t consist of pairs $(b, \bar{v}) \in B_t$. The role of the upper bounds will be described in the next section. The vector set Γ_t can be obtained as follows:

$$\Gamma_t = \bigcup_{(b,\bar{v}) \in B_t} \texttt{backup}(b, t), \tag{4.4}$$

where $\texttt{backup}(b,t)$ denotes a time-dependent backup operator that uses the vectors in Γ_{t+1} to compute a vector belonging to Γ_t. The time-dependent backup operator corresponds to the original backup operator for infinite-horizon POMDPs, but it has been formulated based on multiple time-dependent vector sets rather than one individual vector set. The time-dependent backup operator $\texttt{backup}(b,t)$ is defined as follows:

$$\texttt{backup}(b,t) = \operatorname*{arg\,max}_{\{z_{b,a,t}\}_{a \in A}} b \cdot z_{b,a,t}, \tag{4.5}$$

where

$$z_{b,a,t} = \begin{cases} r_a + \sum_{o \in O} \operatorname{arg\,max}_{\{z_{a,o}^{k,t+1}\}_k} b \cdot z_{a,o}^{k,t+1} & t < h \\ r_a & t = h \end{cases}, \tag{4.6}$$

and $z_{a,o}^{k,t}$ denotes the backprojection of vector $\alpha^{k,t} \in \Gamma_t$:

$$z_{a,o}^{k,t}(s) = \sum_{s' \in S} P(o|a,s')P(s'|s,a)\alpha^{k,t}(s') \quad \forall s. \tag{4.7}$$

The vector r_a contains the immediate reward for action a. In the remainder of the chapter we assume that the backup operator has access to all vector sets and the reward vectors r_a, such that additional arguments can be discarded from the equations and pseudocode.

Our finite-horizon point-based value iteration algorithm FiVI computes multiple time-dependent value functions $\mathcal{V}_t$ represented by vector sets Γ_t using the time-dependent backup operator that we introduced. The actual integration in the algorithm will be explained in Section 4.4.3, which discusses the algorithm in more detail.

4.4.2. Time-dependent value upper bounds and bound updates

Point-based value iteration algorithms typically keep track of upper bounds on the optimal expected value, which enables assessment of the quality of the computed solution. Our FiVI algorithm also includes such computations of upper bounds, for which we provide the required notation and algorithms in this section. The algorithms closely follow the upper bound computations for infinite-horizon POMDPs, but in order to improve understandability we provide a full description in this section.

We consider a time step t and the corresponding belief set B_t. Recall from the previous section that the pairs $(b,\bar{v}) \in B_t$ also contain a value upper bounds $\bar{v}$ corresponding to belief b. These upper bounds $\bar{v}$ can be used to obtain an upper bound for another belief b' that is not represented in the set B_t, based on an upper bound interpolation using the existing beliefs in B_t (Hauskrecht, 2000). The interpolation can be obtained using the following linear program:

$$\begin{aligned} \min \quad & \sum_{(b,\bar{v})\in B_t} c_b \cdot \bar{v} & & \\ \text{s.t.} \quad & \sum_{(b,\bar{v})\in B_t} c_b \cdot b(s) = b'(s) & \forall s & \\ & c_b \geq 0 & \forall (b,\bar{v}) \in B_t. & \end{aligned} \tag{4.8}$$

which assigns weights to the pairs in B_t and returns a linear combination of the upper bounds $\bar{v}$ represented by B_t.

Solving a linear program for every upper bound interpolation can be computationally expensive, and therefore it is more common to use a so-called sawtooth approximation (Hauskrecht, 2000). This approximation is based on the idea that the optimization problem can be simplified by imposing the constraint that weights c_b are assigned to corners of the belief simplex, and at most one belief that is not a corner of the belief simplex. A corner of the belief simplex is a belief in which the belief associated with one state equals 1, and we also refer to such a belief as a corner belief. Under the additional assumptions that we made the upper bound interpolation can be computed using a simple procedure that we call `UB`,

Algorithm 6: Sawtooth approximation (UB)

```
input  : belief $b'$, set $B$ containing belief-bound pairs
output : upper bound corresponding to belief $b'$
1 for $(b, \bar{v}) \in B \setminus \{(e_s, \cdot) \mid s \in S\}$ do
2     $f(b) \leftarrow \bar{v} - \sum_{s \in S} b(s)B(e_s)$
3     $c(b) \leftarrow \min_{s \in S} b'(s) \,/\, b(s)$
4 end
5 $b^* \leftarrow \arg\min_{\{b \mid (b,\bar{v}) \in B \setminus \{e_s \mid s \in S\}\}} c(b) f(b)$
6 return $c(b^*) f(b^*) + \sum_{s \in S} b'(s)B(e_s)$
```

as shown in Algorithm 6, rather than solving a linear program. The algorithm takes an arbitrary belief set B and a belief b' as input, and it returns an upper bound interpolation for b' based on the belief-bound pairs in B. In the algorithm e_s denotes the corner belief corresponding to state s, and the for loop iterates over all pairs $(b, \bar{v}) \in B$ for which b is not a corner of the belief simplex. Furthermore, $B(e_s)$ denotes the upper bound that is currently associated with e_s in the set B. Our notation closely follows the notation used by Poupart, Kim, and Kim (2011), and a justification of the procedure has been provided by Smith (2007). An additional description of upper bound computations has been provided by Shani, Pineau, and Kaplow (2013).

In the finite-horizon setting the upper bounds associated with beliefs can be updated in a point-based fashion, similar to executing regular backups on beliefs. We consider a time step $t < h$ and a belief b that belongs to B_t. The upper bound $\bar{v}$ in $(b, \bar{v}) \in B_t$ can be updated as follows:

$$\max_{a \in A} \sum_{s \in S} R(s,a)b(s) + \sum_{o \in O} P(o|b,a) \cdot \texttt{UB}(b_a^o, B_{t+1}), \tag{4.9}$$

in which the upper bound interpolation is based on the set B_{t+1} corresponding to the next time step. For the final time step $t = h$ it suffices to consider the immediate rewards, and the upper bound is defined by $\max_{a \in A} r_a \cdot b$. In the next section we combine the upper bound update scheme and the time-dependent value functions to create our FiVI algorithm.

4.4.3. Algorithm description of FiVI

The FiVI algorithm takes a POMDP model as input and computes a solution by executing a series of iterations. Within an iteration three phases can be distinguished. First the algorithm executes a procedure to find new belief points. After that, the algorithm computes a new vector set Γ_t for each time step t. Finally, the algorithm updates the upper bounds represented by the belief sets B_t. The full description of the algorithm is provided in Algorithm 7, which we discuss below in more detail.

On lines 1-4 the algorithm starts with initializing vector sets Γ_t, belief sets B_t and the immediate reward vectors r_a. Furthermore, the auxiliary variable τ' is used to keep track of the elapsed time, and δ represents an iteration counter. The latter is used in one of our heuristics in the next chapter.

An iteration of FiVI starts with a call to a procedure `expand`, which is used to find additional beliefs on line 7. The quality of the solution returned by FiVI and the convergence of the algorithm completely depends on these beliefs, because these beliefs are used for computing the value functions. A more detailed description of the procedure is deferred to the next section, which provides a detailed motivation and algorithmic description.

On lines 8-28 the algorithm computes alpha vectors and value upper bounds by iterating backwards over all time steps. The algorithm starts at the end of the horizon h, and it proceeds with the time steps $h-1, h-2, \ldots$ until the initial step is reached. On lines 10-13 the algorithm computes a new vector set Γ_t by executing backups based on a belief b and based on the value function Γ_{t+1} computed in the previous iteration. In this part of the algorithm we use the value functions and backup operator that we have introduced in Section 4.4.1. After computing the new vectors for Γ_t, the algorithm updates all upper bounds defined by B_t on lines 14-27. For this purpose it uses Equation 4.9 and the sawtooth approximation from Section 4.4.2.

An iteration of FiVI ends with the computation of the current value lower bound v_l and upper bound v_u for the initial belief b_1. The difference between these two bounds defines the current gap. Value iteration stops in case a time limit τ has been exceeded, or in case the gap is at most one unit at the ρ-th significant digit. The latter can be checked by computing the maximum allowed gap g_a under this criterion, as shown on line 31, and the algorithm terminates if the current gap is smaller than g_a. As an example we consider upper bound $v_u = 1400$ and lower bound $v_l = 1350$ with $\rho = 2$. At the second significant digit the difference is at most one since $14 - 13 = 1$. It can be verified that the maximum allowed gap g_a equals 100, and since the current gap $v_u - v_l = 1400 - 1350 = 50$ is smaller than 100 the algorithm will terminate. This termination condition is also used by the algorithm GapMin, and it is more generic than imposing an absolute threshold on the gap.

The solution returned by FiVI consists of alpha vectors in sets $\Gamma_1, \ldots, \Gamma_h$, representing the lower bound. In addition, the algorithm returns the upper bound v_u that corresponds to the initial belief b_1. The gap defined by the lower bound and upper bound implicitly defines a guarantee on the quality of the computed solution.

Algorithm 7: Finite-horizon point-based Value Iteration (FiVI)

```
input  : POMDP $M$, precision $\rho$, time limit $\tau$
output : sets $\Gamma_t$ for each time step $t$, upper bound $v_u$
$\Gamma_t \leftarrow \emptyset \;\; \forall t, B_t \leftarrow \emptyset \;\; \forall t$
$r_a \leftarrow (R(s_1,a),R(s_2,a),\ldots,R(s_{|S|},a)) \;\; \forall a$
add corner beliefs to $B_t$ with upper bound $\infty$ $(\forall t)$
$\tau' \leftarrow 0, \;\; \delta \leftarrow 0$
do
    $\delta \leftarrow \delta + 1$
    expand($M,\{\Gamma_1,\ldots,\Gamma_h\},\{B_1,\ldots,B_h\},r$)
    for $t = h, h-1, \ldots, 1$ do
        $\Gamma_t \leftarrow \emptyset$
        for $(b,\bar{v}) \in B_t$ do
            $\alpha \leftarrow$ backup($b,t$)
            $\Gamma_t \leftarrow \Gamma_t \cup \{\alpha\}$
        end
        for $(b,\bar{v}) \in B_t$ do
            $\bar{v} \leftarrow -\infty$
            for $a \in A$ do
                $v \leftarrow r_a \cdot b$
                if $t < h$ then
                    for $o \in O$ do
                        if $P(o|b,a) > 0$ then
                            $v \leftarrow v + P(o|b,a) \cdot \mathtt{UB}(b_a^o, B_{t+1})$
                        end
                    end
                end
                $\bar{v} \leftarrow \max(\bar{v}, v)$
            end
        end
    end
    $v_l \leftarrow \max_{\alpha \in \Gamma_1} \alpha \cdot b_1$
    $v_u \leftarrow$ upper bound $\bar{v}$ associated with $(b_1,\bar{v}) \in B_1$
    $g_a \leftarrow 10^{\lceil \log_{10}(\max(|v_l|,|v_u|)) \rceil - \rho}$
    $\tau' \leftarrow$ elapsed time after the start of the algorithm
while $\tau' < \tau \wedge v_u - v_l > g_a$;
return $(\{\Gamma_1,\ldots,\Gamma_h\}, v_u)$
```

4.4.4. Belief points and convergence of the algorithm

The computation of vectors and upper bounds assumes that we have a set of beliefs B_t for each time step. However, the performance of the algorithm and the quality of computed solution are highly dependent on the actual belief points for which backups are executed. Computing high-quality policies requires coverage of the region of the belief space that is reachable under the execution of an optimal policy. Unfortunately, the optimal policy and the corresponding reachable belief region are initially unknown, which means that these reachable belief points need to be found while computing a policy.

The algorithm FiVI incrementally expands the belief sets using heuristic search, which is guided by the current gap between the value lower bound and upper bound. Our heuristic search procedure is similar to the procedures found in HSVI, SARSOP and GapMin. Below we describe why the action and observation selection strategies in our belief search steer the algorithm towards an optimal solution.

The gap between the lower and upper bound of belief b_1 at time 1 implicitly defines the amount of uncertainty regarding the optimality of the solution. It is important to note that regret of the returned solution is bounded by the gap of the initial belief b_1. This means that the heuristic search procedure should choose actions and observations in such a way that backups and upper bound updates effectively reduce the overall gap.

In order to decide which action needs to be chosen, we first look at the effect of backups and upper bound updates on the gap associated with a belief. For the lower bound the backup is defined by Equation 2.24, and it maximizes over actions a. We define $V(t,b,a)$ as the new expected value when choosing action a in belief b at time t:

$$V(t,b,a) = \sum_{s\in S} R(s,a)b(s) + \sum_{o\in O} P(o|b,a)V(t+1,b_a^o). \tag{4.10}$$

In a similar way we can define the potential upper bound $U(t,b,a)$ that is considered for action a and belief b at time t in the update defined in Equation 4.9:

$$U(t,b,a) = \sum_{s\in S} R(s,a)b(s) + \sum_{o\in O} P(o|b,a)\cdot \mathtt{UB}(b_a^o,B_{t+1}), \tag{4.11}$$

Since both Equation 2.24 and Equation 4.9 maximize over actions, the new gap associated with belief b at time t is defined by:

$$\max_{a\in A} U(t,b,a) - \max_{a\in A} V(t,b,a). \tag{4.12}$$

It can be seen that the new gap is determined by the actions a that maximize $U(t,b,a)$ and $V(t,b,a)$. This suggests that the heuristic search procedure should choose one of these two maximizing actions in order to affect the gap associated with b. It is required to choose the action a that maximizes $U(t,b,a)$, because if a is suboptimal then its upper bound will eventually be lower than the upper bound associated with another action, which will change the action choice later. This behavior cannot be achieved using the action a that maximizes $V(t,b,a)$ because the lower bound can only increase and therefore it is not possible to detect the potential suboptimality of this action choice. The action selection strategy that we use is also known as the IE-MAX heuristic (Kaelbling, 1993) and it ensures the convergence of the algorithm. A theoretical analysis of the action selection rule has been provided by Ross, Pineau, and Chaib-Draa (2008) for general online heuristic search algorithms for POMDPs. It has been shown that the action selection rule ensures that the computed policy defines an ε-optimal action within finite time, which implies that the algorithm converges to optimality in the limit[2]. The action selection strategy that we use is identical to the strategy used in HSVI, SARSOP and GapMin for infinite-horizon problems.

After selecting an action it is required to choose an observation. It is important to note that the lower bounds and upper bounds associated with all reachable beliefs in time steps $t > 1$ contribute to the gap associated with the initial belief b_1. The reason is that both the lower bound computation and the upper bound update follow the structure of the Bellman equation, as shown in Equations 2.24 and 4.9. If one of the bounds associated with a reachable belief is not tight, it means that it also contributes to the gap associated with the initial belief b_1, and therefore it is important to execute backups and updates on such reachable beliefs. Our algorithm chooses an observation leading to a belief with maximum gap in the next time step $t+1$:

$$\operatorname*{arg\,max}_{\{o\in O \mid P(o|b,a)>0\}} \; \{ \, \mathtt{UB}(b_a^o, B_{t+1}) - \max_{\alpha\in\Gamma_{t+1}} \alpha \cdot b_a^o \, \}. \tag{4.13}$$

This criterion is also used by infinite-horizon algorithms, but they typically weight the gap using the discount factor, which is not required in the finite-horizon case.

The full description of the search procedure `expand` is shown in Algorithm 8. The algorithm performs a forward search starting from the initial belief, based on the action and observation selection rules that we described in this section. The belief points that are found during the search are added to the belief sets used by FiVI. The belief-bound pairs are added to the set B_{t+1} rather than B_t because the beliefs always correspond to the next time step. This also means that it is not required to consider the final time step $t = h$ in the for loop. The search procedure is invoked in each iteration of FiVI in order to add new beliefs, which ensures that the FiVI algorithm iteratively reduces the gap of the solution.

[2]From a theoretical perspective it is required to use the exact upper bound computation in order to ensure that convergence results are unaffected. If the sawtooth approximation is used, then it is important that exact bounds are computed periodically, rather than using the approximation in every iteration of the algorithm.

Algorithm 8: Belief expansion (`expand`)

```
input :$M, \{\Gamma_1, \dots, \Gamma_h\}, \{B_1, \dots, B_h\}, r$
1 $b \leftarrow b_1$
2 for $t = 1, \dots, h-1$ do
3     $a \leftarrow \arg\max_{a \in A}\{\ r_a \cdot b + \sum_{\{o \in O \mid P(o|b,a)>0\}} P(o|b,a) \cdot \texttt{UB}(b_a^o, B_{t+1})\ \}$
4     $o \leftarrow \arg\max_{\{o \in O \mid P(o|b,a)>0\}}\{\ \texttt{UB}(b_a^o, B_{t+1}) - \max_{\alpha \in \Gamma_{t+1}} \alpha \cdot b_a^o\ \}$
5     $B_{t+1} \leftarrow B_{t+1} \cup \{(b_a^o, \infty)\}$
6     $b \leftarrow b_a^o$
7 end
```

Based on the construction of the procedure `expand`, we can analyze the number of iterations performed by FiVI. The expand procedure finds at most $(|A||O|)^h$ new beliefs, and there are no iterations in which it does not find a new belief before convergence. This means that the total number of iterations of FiVI is $O((|A||O|)^h)$. The same bound applies to the space requirements of the algorithm, because the algorithm stores the beliefs and the corresponding vectors in memory. In practice it can be expected that the number of iterations is much lower than this worst case bound, since the `expand` procedure steers the search in the direction of beliefs reachable under the execution of an optimal policy. However, without making assumptions about the planning domain the bound cannot be tightened.

4.5. Backup and update heuristics

The point-based algorithm FiVI executes backups to compute new vector sets Γ_t in each iteration. This approach is clean and simple, but it can be relatively inefficient. For example, the algorithm constructs the new value functions from scratch by executing backups on all beliefs. Furthermore, the size of the B_t sets grows during the execution of the algorithm, and therefore an increasing amount of time is required to execute all backups.

The upper bound updates executed by the algorithm (lines 14-27) can also be considered inefficient, because in each iteration the algorithm computes a new upper bound for each belief. The upper bound interpolation function `UB` computes the upper bounds based on all beliefs, while only a few of these beliefs eventually affect the returned upper bound.

In the remainder of this section we address the aforementioned issues by discussing a strategy to enhance the efficiency of backups, and we identify a dependency structure which allows for more efficient upper bound updates.

Algorithm 9: Perseus Belief Selection (PBS)

```
input  : vector set $\Gamma_t$, belief set $B_t$
output : new vector set $\Gamma_t$ after executing backups
$\Gamma \leftarrow \Gamma_t,\ \Gamma_t \leftarrow \emptyset,\ B \leftarrow B_t$
while $B \neq \emptyset$ do
    $(b, \bar{v}) \leftarrow$ randomly selected pair from $B$
    $\alpha \leftarrow \texttt{Backup}(b, t)$
    $\alpha' \leftarrow \arg\max_{\alpha' \in \Gamma} \alpha' \cdot b$
    if $\alpha \cdot b \geq \alpha' \cdot b$ then
        $\Gamma_t \leftarrow \Gamma_t \cup \{\alpha\}$
    else
        $\Gamma_t \leftarrow \Gamma_t \cup \{\alpha'\}$
    end
    $B \leftarrow \{b \in B \mid \max_{\alpha \in \Gamma_t} \alpha \cdot b < \max_{\alpha \in \Gamma} \alpha \cdot b\}$
end
return $\Gamma_t$
```

4.5.1. Perseus Belief Selection (PBS)

In this section we present a strategy to improve the efficiency of backups, which employs a randomized belief selection method similar to the randomized backup stage found in Perseus (Spaan and Vlassis, 2005). The improve-only principle of this backup stage allows us to perform backups on randomly-selected points only, while ensuring that the newly computed Γ_t set is at least as good as in the previous iteration. A description is shown in Algorithm 9, which replaces lines 9-13 of our algorithm. The algorithm keeps track of a set B containing non-improved beliefs. The key improvement follows from the fact that one backup may improve the value for multiple beliefs. As a result, the set B shrinks quickly and it may not be required to execute backups for all beliefs.

4.5.2. Dependency-Based Bound Updates (DBBU)

In this section we improve the efficiency of upper bound updates performed during the execution of FiVI. Our preliminary observation is that we do not need to compute new upper bounds for beliefs with zero gap, because for such beliefs the upper bound is already the tightest possible bound. This means that we can mark beliefs with zero gap, and such beliefs will not be taken into account in remaining iterations of FiVI while computing new upper bounds. Unless stated otherwise, the implementations of our algorithms will always ignore beliefs with zero gap when computing new upper bounds.

Our main observation is that Algorithm 7 executes many upper bound inter-

$$\begin{tikzcd}
 & \mathtt{UB}(b^{o_1}_{a_1}, B_{t+1}) \arrow[r] & (b^*, \bar{v}) \in B_{t+1} \\
(b, \bar{v}) \in B_t \arrow[ur, "a_1, o_1"] \arrow[r, "a_1, o_2"] \arrow[ddr, "a_l, o_k"] & \mathtt{UB}(b^{o_2}_{a_1}, B_{t+1}) \arrow[r] & (b^*, \bar{v}) \in B_{t+1} \\
 & \vdots & \\
 & \mathtt{UB}(b^{o_k}_{a_l}, B_{t+1}) \arrow[r] & (b^*, \bar{v}) \in B_{t+1}
\end{tikzcd}$$

Figure 4.2: Dependencies between a belief $(b, \bar{v}) \in B_t$ and beliefs $(b^*, \bar{v}) \in B_{t+1}$ used for upper bound interpolation

polations when updating the bounds on lines 14-27. For several beliefs b_a^o there is a call to the function UB, which computes an upper bound interpolation based on the corner beliefs and just one additional belief $(b^*, \bar{v}) \in B_{t+1}$ (see line 5 of Algorithm 6). This structure is visually depicted in Figure 4.2. It shows a belief pair $(b, \bar{v}) \in B_t$ for which an upper bound is updated in the loop starting on line 14 of Algorithm 7. In order to compute the new upper bound, the algorithm needs several upper bound interpolations for different successor beliefs b_a^o. For clarity we denote these beliefs by $b_{a_1}^{o_1}, b_{a_1}^{o_2}, \dots, b_{a_l}^{o_k}$ in the figure, indicating that the successor beliefs used for interpolation are different.

Each interpolation computed by the function UB in Algorithm 6 is based on one belief $(b^*, \bar{v}) \in B_{t+1}$. The arrows induce a dependency graph between the beliefs of subsequent steps, and this graph implicity indicates how upper bounds have been propagated from $t = h$ back to $t = 1$. It turns out that the dependency graph remains relatively constant during the execution[3]. In other words: when computing a new upper bound for a belief $(b, \bar{v}) \in B_t$, it is often selecting the same beliefs $(b^*, \bar{v}) \in B_{t+1}$ in the calls to the function UB. We propose to exploit this dependency structure to reduce the number of beliefs considered by UB.

An overview of our Dependency-Based Bound Update (DBBU) method is shown in Algorithm 10. Once in θ iterations of point-based value iteration we keep track of the dependencies between beliefs, as visualized in Figure 4.2. When updating the upper bound $\bar{v}$ for a pair $(b, \bar{v}) \in B_t$, these dependencies can be determined by looking at the beliefs b^* used in the calls to UB on line 21 of Algorithm 7. We store all beliefs b^* that were used in the set B_b. We periodically determine the dependencies, because in general we cannot assume that the dependency graph remains constant. The reason is that the algorithm iteratively adds new beliefs, and such beliefs may be used for interpolation as well.

In all other iterations we still compute a new bound for each belief pair $(b, \bar{v}) \in B_t$, but we replace the calls to UB(b_a^o, B_{t+1}). Rather than computing the interpola-

[3]A similar observation has been made for the calls to the exact upper bound interpolation in GapMin (Poupart, Kim, and Kim, 2011), in which the convex combination remains fairly constant. The presented alternative uses a so-called augmented POMDP, but it is important to note that it still requires $|S||A||O| + 1$ calls to UB, and it requires the fast-informed bound, which does not directly apply to finite-horizon settings. GapMin does not focus on the dependency structure of the upper bound update.

Algorithm 10: Dependency-Based Bound Updates (DBBU)

```
input  : belief sets $B_t$ and $B_{t+1}$, iteration $\delta$, interval $\theta$
1  if $\delta$ mod $\theta = 0$ then
2      for $(b, \bar{v}) \in B_t$ do
3          Lines 15-26 from Algorithm 7
4          $B_b \leftarrow$ set containing beliefs $b^*$ used in UB calls
5      end
6      $B'_{t+1} \leftarrow B_{t+1}$
7  else
8      for $(b, \bar{v}) \in B_t$ do
9          $B^*_{t+1} \leftarrow \{(b, \bar{v}) \in B_{t+1} \mid b \in B_b \vee ((b, \bar{v}) \in B_{t+1} \wedge (b, \bar{v}) \notin B'_{t+1})\}$
10         Lines 15-26 from Algorithm 7, where UB uses $B^*_{t+1}$ rather than $B_{t+1}$
11     end
12 end
```

tion based on all beliefs in B_{t+1}, we use a subset $B^*_{t+1} \subseteq B_{t+1}$, where B^*_{t+1} contains all beliefs b^* defined by the dependency graph (e.g., when updating for $(b, \bar{v}) \in B_t$ this would be the set B_b). Typically this set is much smaller than B_{t+1}, and if the dependency graph is constant then it also contains the beliefs that would be used by an interpolation based on B_{t+1}. As a result, the function UB iterates over much fewer beliefs.

Beliefs that were found after the last construction of the dependency graph are always included in B^*_{t+1}. For this purpose the algorithm defines the auxiliary variable B'_{t+1} when constructing the graph, which contains all beliefs that were part of B_{t+1} when constructing the graph. This variable is used in the definition of B^*_{t+1} on line 9, such that it includes new beliefs that are part of B_{t+1} which were not part of B'_{t+1} yet.

It is important to note that DBBU can be easily combined with PBS, because DBBU affects the upper bound updates of FiVI, while PBS only changes the procedure to execute backups on beliefs. The influence of the interval parameter θ on the performance of DBBU will be studied in the next section.

4.6. Experiments

In this section we present our experimental evaluation. We start with a comparison of multiple variants of FiVI, in which we test the influence of our strategies PBS and DBBU on runtime, convergence and solution quality. After that, we provide a more in-depth study of the behavior of PBS and DBBU, and we provide a comparison with 3 alternative approaches which may be used for finite-horizon problems.

	4x5x2	AircraftID	Hallway	Network
$\lvert S \rvert$	39	12	60	7
$\lvert A \rvert$	4	6	5	4
$\lvert O \rvert$	4	5	21	2

Table 4.2: Properties of the domains involved in the experiments

4.6.1. Performance of FiVI with PBS and DBBU

In the first set of experiments we compare standard FiVI, FiVI augmented with PBS, and FiVI augmented with PBS and DBBU. For these variants of the algorithm we use the names VI, PBS and DBBU, respectively. We let the algorithms run for at most 15 minutes, after which execution is terminated. Furthermore, we stop algorithm execution if the gap between the lower bound and upper bound drops below 0.01. Since FiVI is an anytime algorithm, we can assess which variant of the algorithm provides the best solution given the fixed amount of computation time that is available.

We test our algorithms with multiple planning horizons h, which means that we discard the default discount factors defined by the domains. We use multiple domains from `pomdp.org`, which we solve with horizons $h = 5, 10, 15, 20$. The domains have been chosen such that the algorithm is able to reduce the gap to a value close to 0 within the time limit of 900 seconds. This is important for testing whether the dependency graph during algorithm execution becomes constant, and it enables us to test the effects of our heuristics until convergence of the algorithm. An overview of the domain properties is provided in Table 4.2. For DBBU we consider the parameters $\theta = 10, 20, 30, 40$, which we append to the names of the algorithms. We compare the algorithms by measuring the total runtime, the lower bound on the expected reward of the computed policy, as well as the gap associated with the computed policy. Each algorithm is executed 10 times, such that we can report the mean and standard deviation for these measures. Prior to running the algorithms, we intuitively expect that PBS improves the performance of VI since it is likely that it executes fewer backups. Furthermore, we expect that DBBU improves the performance even more, because in that case it iterates over fewer beliefs when computing upper bounds.

The results of our experiment are shown in Tables 4.3 and 4.4, in which each entry represents the mean based on 10 runs of the algorithm, and the small entries denote the standard deviation. Based on the runs in which the time out of 900 seconds was not reached, we can conclude that PBS consistently improves the performance of plain FiVI, meaning that it needs less time to reach a solution with a gap below 0.01. The variants of the algorithm which include DBBU become even faster, and they typically need even less time. This is especially noticable when increasing the horizon to, e.g., $h = 15$ and $h = 20$. In these cases the running time of

		VI	PBS	DBBU10	DBBU20	DBBU30	DBBU40
4x5x2	Time (s)	$0.915_{0.049}$	$0.307_{0.011}$	$0.202_{0.025}$	$0.197_{0.006}$	$0.196_{0.003}$	$0.203_{0.007}$
$h = 5$	LB	$0.429_{0.000}$	$0.429_{0.000}$	$0.429_{0.000}$	$0.429_{0.000}$	$0.429_{0.000}$	$0.429_{0.000}$
	Gap	$0.003_{0.000}$	$0.003_{0.000}$	$0.003_{0.000}$	$0.003_{0.000}$	$0.003_{0.000}$	$0.003_{0.000}$
4x5x2	Time (s)	$13.541_{0.253}$	$5.234_{0.171}$	$2.741_{0.152}$	$2.808_{0.146}$	$2.818_{0.066}$	$2.905_{0.065}$
$h = 10$	LB	$1.119_{0.000}$	$1.119_{0.000}$	$1.119_{0.000}$	$1.119_{0.000}$	$1.119_{0.000}$	$1.119_{0.000}$
	Gap	$0.010_{0.000}$	$0.010_{0.001}$	$0.010_{0.000}$	$0.010_{0.000}$	$0.010_{0.000}$	$0.009_{0.000}$
4x5x2	Time (s)	$112.401_{0.495}$	$39.147_{1.021}$	$17.756_{0.576}$	$17.64_{0.452}$	$17.726_{0.401}$	$18.746_{0.679}$
$h = 15$	LB	$1.619_{0.000}$	$1.619_{0.000}$	$1.619_{0.000}$	$1.619_{0.000}$	$1.619_{0.000}$	$1.619_{0.000}$
	Gap	$0.007_{0.000}$	$0.009_{0.001}$	$0.009_{0.001}$	$0.009_{0.001}$	$0.009_{0.001}$	$0.009_{0.001}$
4x5x2	Time (s)	$346.417_{2.245}$	$141.535_{7.571}$	$70.575_{3.046}$	$68.119_{3.730}$	$67.922_{4.138}$	$70.798_{3.732}$
$h = 20$	LB	$2.256_{0.000}$	$2.256_{0.000}$	$2.256_{0.000}$	$2.256_{0.000}$	$2.256_{0.000}$	$2.256_{0.000}$
	Gap	$0.009_{0.000}$	$0.009_{0.001}$	$0.009_{0.000}$	$0.009_{0.01}$	$0.009_{0.001}$	$0.009_{0.001}$
		VI	PBS	DBBU10	DBBU20	DBBU30	DBBU40
AircraftID	Time (s)	$0.031_{0.029}$	$0.014_{0.004}$	$0.019_{0.014}$	$0.013_{0.001}$	$0.013_{0.002}$	$0.013_{0.002}$
$h = 5$	LB	$-45.393_{0.000}$	$-45.393_{0.000}$	$-45.393_{0.000}$	$-45.393_{0.000}$	$-45.393_{0.000}$	$-45.393_{0.000}$
	Gap	$0.002_{0.000}$	$0.002_{0.001}$	$0.004_{0.001}$	$0.003_{0.001}$	$0.004_{0.001}$	$0.003_{0.001}$
AircraftID	Time (s)	$1.068_{0.059}$	$0.450_{0.047}$	$0.456_{0.060}$	$0.432_{0.024}$	$0.460_{0.052}$	$0.450_{0.034}$
$h = 10$	LB	$-95.240_{0.000}$	$-95.241_{0.000}$	$-95.240_{0.000}$	$-95.240_{0.000}$	$-95.240_{0.000}$	$-95.241_{0.000}$
	Gap	$0.010_{0.000}$	$0.010_{0.001}$	$0.009_{0.000}$	$0.009_{0.001}$	$0.009_{0.000}$	$0.009_{0.001}$
AircraftID	Time (s)	$18.837_{0.168}$	$8.582_{1.033}$	$4.958_{0.413}$	$5.141_{0.236}$	$5.436_{0.451}$	$5.877_{0.623}$
$h = 15$	LB	$-149.467_{0.000}$	$-149.467_{0.000}$	$-149.467_{0.000}$	$-149.467_{0.000}$	$-149.467_{0.000}$	$-149.467_{0.000}$
	Gap	$0.010_{0.000}$	$0.010_{0.000}$	$0.010_{0.001}$	$0.010_{0.000}$	$0.010_{0.000}$	$0.010_{0.000}$
AircraftID	Time (s)	$163.189_{2.356}$	$78.369_{4.647}$	$29.956_{1.658}$	$29.895_{2.247}$	$31.219_{1.746}$	$33.127_{2.192}$
$h = 20$	LB	$-208.013_{0.000}$	$-208.013_{0.000}$	$-208.013_{0.000}$	$-208.013_{0.000}$	$-208.013_{0.000}$	$-208.013_{0.000}$
	Gap	$0.010_{0.000}$	$0.010_{0.000}$	$0.010_{0.000}$	$0.010_{0.000}$	$0.010_{0.000}$	$0.010_{0.000}$

Table 4.3: Algorithm comparison for domains 4x5x2 and AircraftID

DBBU becomes significantly lower than the running time of PBS, which confirms our initial expectations. The variants of the algorithm with PBS and DBBU include randomization, but the low standard deviations of the lower bounds and gaps indicate that it has limited influence on the quality of the solution returned.

As can be seen in the table, the choice of the interval θ influences the performance of DBBU, but the results do not allow us to identify a generic choice for this parameter which provides the best performance throughout all domains. It should be noted, however, that DBBU becomes significantly faster than FiVI with only PBS, regardless of the choice of θ. As a general rule we can say that setting θ too high (e.g., much higher than 40) is unlikely to give good performance because then potential changes in upper bounds are not taken into account quickly during the execution of the algorithm. In the Hallway domain with horizon $h = 5$ we can also see that our heuristics improve the performance of plain FiVI. For the

		VI	PBS	DBBU10	DBBU20	DBBU30	DBBU40
Hallway	Time (s)	$9.843_{0.466}$	$5.045_{0.219}$	$6.358_{0.276}$	$6.370_{0.213}$	$6.523_{0.286}$	$6.498_{0.296}$
$h = 5$	LB	$0.098_{0.000}$	$0.098_{0.001}$	$0.098_{0.001}$	$0.098_{0.000}$	$0.098_{0.000}$	$0.098_{0.000}$
	Gap	$0.009_{0.000}$	$0.009_{0.000}$	$0.009_{0.00}$	$0.009_{0.000}$	$0.009_{0.000}$	$0.009_{0.000}$
Hallway	Time (s)	$909.567_{4.671}$	$903.979_{3.390}$	$904.373_{3.380}$	$904.200_{2.492}$	$903.358_{2.502}$	$902.969_{2.185}$
$h = 10$	LB	$0.327_{0.000}$	$0.334_{0.000}$	$0.335_{0.000}$	$0.335_{0.000}$	$0.334_{0.000}$	$0.334_{0.000}$
	Gap	$0.104_{0.000}$	$0.087_{0.003}$	$0.083_{0.002}$	$0.082_{0.002}$	$0.083_{0.002}$	$0.083_{0.002}$
Hallway	Time (s)	$911.973_{5.962}$	$908.061_{3.659}$	$904.740_{3.136}$	$904.405_{2.191}$	$904.415_{2.638}$	$905.049_{2.830}$
$h = 15$	LB	$0.628_{0.001}$	$0.632_{0.001}$	$0.635_{0.002}$	$0.635_{0.002}$	$0.634_{0.001}$	$0.635_{0.001}$
	Gap	$0.272_{0.001}$	$0.260_{0.003}$	$0.255_{0.002}$	$0.256_{0.003}$	$0.257_{0.002}$	$0.256_{0.003}$
Hallway	Time (s)	$916.264_{5.809}$	$906.485_{4.881}$	$905.811_{2.717}$	$905.738_{2.744}$	$908.350_{5.132}$	$906.481_{3.907}$
$h = 20$	LB	$0.902_{0.000}$	$0.918_{0.003}$	$0.921_{0.002}$	$0.920_{0.003}$	$0.920_{0.003}$	$0.919_{0.002}$
	Gap	$0.430_{0.000}$	$0.403_{0.004}$	$0.398_{0.003}$	$0.399_{0.004}$	$0.399_{0.004}$	$0.400_{0.003}$
		VI	PBS	DBBU10	DBBU20	DBBU30	DBBU40
Network	Time (s)	$0.014_{0.012}$	$0.004_{0.001}$	$0.006_{0.002}$	$0.004_{0.002}$	$0.003_{0.001}$	$0.003_{0.001}$
$h = 5$	LB	$81.137_{0.000}$	$81.137_{0.000}$	$81.137_{0.000}$	$81.137_{0.000}$	$81.137_{0.000}$	$81.137_{0.000}$
	Gap	$0.000_{0.000}$	$0.000_{0.000}$	$0.000_{0.000}$	$0.000_{0.000}$	$0.000_{0.000}$	$0.000_{0.000}$
Network	Time (s)	$0.341_{0.066}$	$0.149_{0.006}$	$0.095_{0.004}$	$0.097_{0.006}$	$0.107_{0.004}$	$0.110_{0.004}$
$h = 10$	LB	$151.18_{0.000}$	$151.18_{0.000}$	$151.18_{0.000}$	$151.18_{0.000}$	$151.18_{0.000}$	$151.18_{0.000}$
	Gap	$0.010_{0.000}$	$0.009_{0.001}$	$0.009_{0.001}$	$0.009_{0.001}$	$0.008_{0.001}$	$0.009_{0.001}$
Network	Time (s)	$32.000_{0.194}$	$22.141_{1.790}$	$5.612_{0.275}$	$5.041_{0.381}$	$5.375_{0.362}$	$5.863_{0.337}$
$h = 15$	LB	$224.616_{0.000}$	$224.616_{0.000}$	$224.616_{0.000}$	$224.616_{0.000}$	$224.616_{0.000}$	$224.616_{0.000}$
	Gap	$0.010_{0.000}$	$0.010_{0.000}$	$0.010_{0.000}$	$0.010_{0.000}$	$0.010_{0.000}$	$0.010_{0.000}$
Network	Time (s)	$901.721_{0.682}$	$901.124_{0.544}$	$267.056_{62.270}$	$204.782_{66.242}$	$114.482_{12.793}$	$142.049_{23.324}$
$h = 20$	LB	$298.149_{0.000}$	$298.149_{0.000}$	$298.149_{0.000}$	$298.149_{0.000}$	$298.149_{0.000}$	$298.149_{0.000}$
	Gap	$0.018_{0.000}$	$0.014_{0.001}$	$0.010_{0.000}$	$0.010_{0.000}$	$0.010_{0.000}$	$0.010_{0.000}$

Table 4.4: Algorithm comparison for domains Hallway and Network

horizons $h = 10, 15, 20$ we observe that the lower bounds and gaps of PBS and DBBU are slightly better when the algorithm reaches the timeout. However, it should be noted that the domain is difficult to solve, which means that it takes a long time to reach a solution with a gap that is lower than the tolerance. This makes it hard to derive conclusions regarding algorithm performance based on those results.

In Figure 4.3 we visualize how the gap decreases over time during one execution of the algorithm. From these graphs we can derive two conclusions about the performance of our two strategies PBS and DBBU. First, in the variants of FiVI which include either PBS or DBBU the gap tends to decrease faster, meaning that it approaches an optimal solution faster. Second, our bound update strategy DBBU almost always improves the performance of FiVI with DBBU. The tables and graphs together confirm our initial expectation that PBS improves the performance of

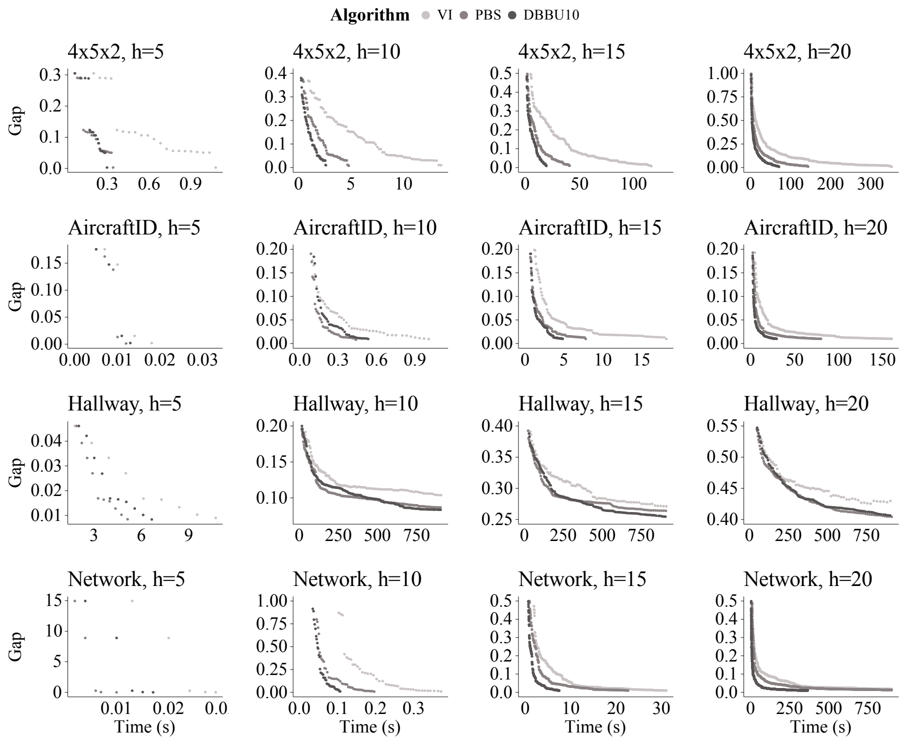


Figure 4.3: Gap during the execution of FiVI

plain FiVI, and our expectation that DBBU improves the performance even more. Furthermore, our experiment shows that FiVI is an effective method to compute finite-horizon solutions while providing guarantees on the quality of the resulting solution.

4.6.2. Number of backups executed by PBS

In our second experiment we study the hypothesis that PBS executes fewer backups due to the potential to skip beliefs for which the value function has improved. We measure the reduction of the number of backups due to PBS as follows. We let #beliefs_total denote the total number of beliefs that has been added so far, counted across all time steps involved. Furthermore, we let #num_backups denote the total number of backups executed by PBS. Now the reduction of the number of backups can be expressed as follows:

$$-100 \times \left(\frac{\#\text{num\_backups} - \#\text{beliefs\_total}}{\#\text{beliefs\_total}} \right). \tag{4.14}$$

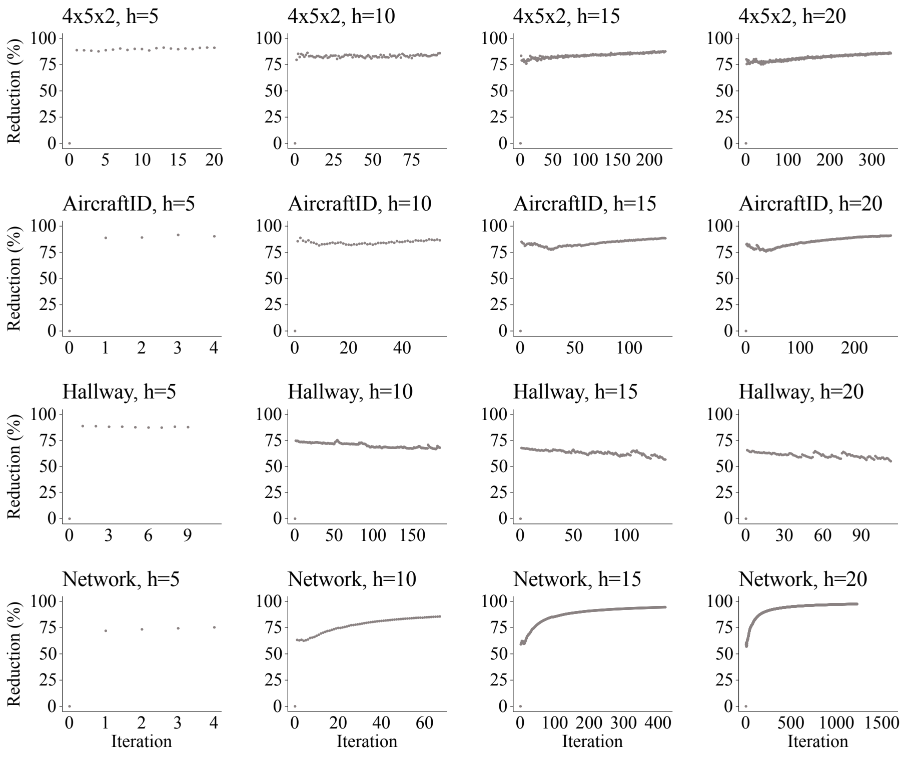


Figure 4.4: Reduction of the number of backups

In Figure 4.4 we use this metric to visualize the reduction of the number of backups for each iteration of FiVI. These results confirm that PBS executes much fewer backups than plain FiVI, which also explains why PBS can run faster than FiVI.

Although PBS accelerates the iterations of FiVI, it is important to note that FiVI with PBS may need more iterations to reach a solution of the same quality. The explanation for this is that executing backups on all beliefs may result in a better value function than the value function obtained when executing backups on only a few beliefs. As we have concluded from Figure 4.4, an iteration with PBS generally runs faster, but as a consequence of the behavior of PBS the total number of iterations may increase. This can be seen in Table 4.5, which shows the number of iterations of FiVI performed until termination. However, based on the initial results in Table 4.3 and Table 4.4 we can conclude that the gain introduced by PBS dominates the additional runtime introduced by running additional iterations.

Domain	h	Iterations VI	Iterations PBS
4x5x2	5	21	21
	10	92	94
	15	225	225
	20	332	345
AircraftID	5	4	5
	10	50	55
	15	138	134
	20	267	267
Hallway	5	9	10
	10	124	186
	15	96	138
	20	81	114
Network	5	5	5
	10	70	68
	15	381	421
	20	997	1229

Table 4.5: Number of iterations of FiVI until termination

4.6.3. Dependency graph construction in DBBU

Our bound update strategy DBBU is based on the intuition that the dependency structure of the beliefs does not change much during the execution of value iteration. In order to test whether this is indeed the case in our domains, we execute an experiment in which we keep track of the number of times that all the dependencies remain constant when constructing a new dependency graph. To be more specific, we let DBBU compute new dependency graphs in each iteration and we measure the number of graph constructions in which the dependencies remain the same compared to the previous iteration. We let #graph denote the total number of graph constructions in an iteration and #graph_unchanged denotes the number of times that the dependency graph associated with a belief does not change. The total number of unchanged dependency graphs within an iteration now corresponds to:

$$100 \times \left(\frac{\#\text{graph\_unchanged}}{\#\text{graph}} \right). \tag{4.15}$$

Figure 4.5 visualizes this percentage as a function of the iterations of FiVI, which confirms that the dependency graphs remain almost always constant during the execution of FiVI. This result implies that the upper bounds returned by `UpperBound` based on B^*_{t+1} are almost always the same as the upper bounds re-

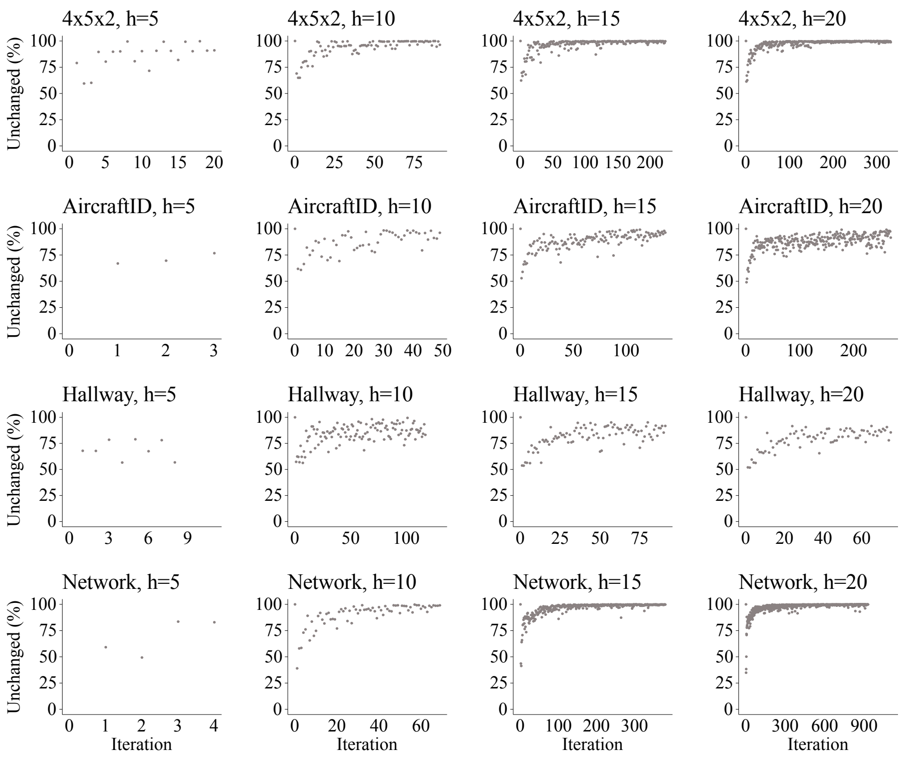


Figure 4.5: Unchanged dependency graphs as a function of the number of iterations

turned by UpperBound based on B_{t+1} (see Algorithm 10), while iterating over only a small subset of beliefs during the computation of upper bound interpolations in UpperBound. From this experiment we can conclude that the upper bound updates in FiVI take less time due to DBBU, and in many cases our strategy does not affect the computed upper bounds.

4.6.4. Comparison with alternative methods

In our final experiment we present a comparison with three alternative methods which may be used to compute solutions to finite-horizon problems. These methods do not provide performance guarantees and they have several limitations. From a practical point of view it can be suitable to use them (e.g., if strict performance guarantees are not required). Therefore, we want to show how such methods perform when solving the instances used in our previous experiments. We want to emphasize that the methods have not been designed for finite-horizon problems, and they have not been presented as such in the literature.

The first method we consider starts with sampling reachable beliefs in the belief

		VI	PBS	DBBU	S1000	RTDP-Bel	GapMin
4x5x2	LB	2.256	2.256	2.256	2.256	2.256	2.118
$h = 20$	Gap	0.009	0.009	0.009	0.237	N/A	N/A
AircraftID	LB	-208.013	-208.013	-208.013	-208.013	-208.765	-208.708
$h = 20$	Gap	0.010	0.010	0.010	1.096	N/A	N/A
Hallway	LB	0.902	0.918	0.921	0.972	0.078	0.974
$h = 20$	Gap	0.430	0.403	0.398	0.293	N/A	N/A
Network	LB	298.149	298.149	298.149	298.128	292.934	291.305
$h = 20$	Gap	0.018	0.014	0.010	13.679	N/A	N/A

Table 4.6: Comparison with random belief sampling, RTDP and infinite-horizon GapMin

space by exploring the environment randomly during a fixed number of episodes. After that, it performs only one iteration of FiVI to compute value functions and upper bounds. It never executes heuristic search to find additional beliefs. This method is simple to execute, but it does not provide any performance guarantees because exploring the environment randomly does not necessarily provide the belief points that are needed to compute a potentially optimal value function. We let the algorithm sample beliefs during 1000 episodes, and the algorithm is denoted by S1000.

The second method we consider is a finite-horizon version of RTDP-Bel (Bonet and Geffner, 2009). The only difference with the infinite-horizon version is that we use separate value functions for each time step. RTDP-Bel discretizes beliefs in order to represent value functions in memory, and due to this discretization it does not necessarily converge to optimality. During our experiments we use discretization parameter $D = 10$ because this parameter setting gives us good results, and we execute the algorithm for 900 seconds, similar to the previous experiments. For more details about RTDP-Bel we refer to our discussion in Section 4.3.

The third method we consider is the infinite-horizon algorithm GapMin which computes an infinite-horizon policy with $\gamma = 0.99$, which we subsequently evaluate during simulation runs. In Section 4.2 we already concluded that this approach is not suitable for finite-horizon problems, and in this section we validate empirically whether this is indeed the case. Similar to RTDP-Bel and the previous experiments, we use a time limit of 900 seconds.

We present the results of our experiment in Table 4.6, which shows the lower bound and the associated gap. The results for VI, PBS and DBBU have been copied from the previous experiments, where we selected the best-performing version of DBBU. For RTDP-Bel and GapMin we do not obtain a lower bound, and therefore the reported number is the mean reward measured during 1 million simulation

		S1000	VI	PBS	DBBU10
4x5x2	Time (s)	26.661	26.661	26.661	26.661
$h = 20$	LB	2.256	2.249	2.256	2.256
	Gap	0.237	0.358	0.181	0.114
AircraftID	Time (s)	25.520	25.520	25.520	25.520
$h = 20$	LB	-208.013	-208.013	-208.013	-208.013
	Gap	1.096	0.035	0.020	0.011
Hallway	Time (s)	830.610	830.610	830.610	830.610
$h = 20$	LB	0.972	0.903	0.916	0.912
	Gap	0.293	0.429	0.407	0.408
Network	Time (s)	5.412	5.412	5.412	5.412
$h = 20$	LB	298.128	298.148	298.149	298.149
	Gap	13.679	0.702	0.596	0.247

Table 4.7: Solution quality for VI, PBS and DBBU10 with time limit equal to S1000 runtime

runs. Below we discuss our most important observations and conclusions for each alternative method.

The mean reward of the solutions computed by RTDP-Bel is close to the lower bounds computed by FiVI, which shows that RTDP-Bel can be an effective and simple method to compute finite-horizon solutions. However, in general the algorithm does not keep track of upper bounds, which means that it is impossible to assess whether a solution computed by RTDL-Bel is close to optimal or not.

The mean reward of the solutions computed by GapMin is slightly lower than the lower bounds computed by FiVI in the domains 4x5x2, AircraftID and Network. This confirms that a policy computed for an infinite-horizon problem does not always perform well if the actual problem has a finite time horizon. In the Hallway domain the infinite-horizon policy performs very well. This seems surprising, but there is a very intuitive explanation which shows why this is the case. Upon reaching the goal the agent restarts from the initial belief, and an infinite-horizon policy tries to reach the goal as many times as possible in order to maximize its expected reward. In practice this means that the infinite-horizon policy tries to reach the goal as fast as possible, which is also the best strategy in case there is only a finite number of steps available. Similar to RTDP-Bel, it is important to note that the algorithm does not compute an upper bound for the finite-horizon case, which prevents assessment of policy quality in general.

For the belief sampling approach S1000 we observe that the lower bounds are very close or identical to the lower bounds found by the variants of FiVI in the domains 4x5x2, AircraftID and Network. However, the associated gap is significantly larger than the gaps returned by FiVI, which means that S1000 did not find beliefs that are needed to reduce the gap effectively. In order to provide a better comparison we execute another experiment in which we give VI, PBS and DBBU a time limit that equals the total runtime of S1000. This allows us to investigate whether the FiVI variants compute a better solution than S1000 within the same amount of time. The results of this comparison are shown in Table 4.7. In the Hallway domain S1000 provides a better solution than the FiVI variants without belief sampling, which means that it has found reachable beliefs leading to a better solution, which were not found yet by the FiVI algorithm within its time limit. In the domains 4x5x2, AircraftID and Network our algorithm computes solutions with a smaller gap than the solution returned by S1000, which means that our algorithm performs better than the belief sampling approach. For S1000 in general we can conclude that it may work well in some domains, but it samples an excessively large number of reachable beliefs, which becomes intractable if the POMDP model is large.

To conclude, our experiment has shown that the belief sampling approach, RTDP-Bel and GapMin may be suitable for planning in finite-horizon settings. However, the algorithms do not provide performance guarantees and sampling a large number of beliefs can be computationally expensive (both the sampling itself and executing backups on those beliefs). FiVI, on the other hand, provides performance guarantees and typically it uses much fewer beliefs to compute the solution. Finally, the experiment confirms our observation that the algorithms are not suitable for solving finite-horizon problems in general.

4.7. Conclusions

Finite-horizon POMDPs naturally arise in application domains in which policies need to be executed during a finite number of time steps. For example, aggregators which control the charging process of electric vehicles typically optimize their decisions for the next 24 hours such that undiscounted charging cost is minimized. Another example can be found in the area of maintenance, in which equipment such as machines should be maintained well during their lifespan. Unfortunately, computing finite-horizon POMDP solutions turns out to be more complicated than intuitively expected. In the literature several approximate POMDP algorithms have been presented which form the current state of the art, such as the class of point-based value iteration algorithms. These algorithms have been designed based on the assumption that an infinite horizon with discounting is considered, but due to this assumption the algorithms do not easily generalize to finite-horizon settings. In other words, state-of-the-art approximate POMDP algorithms cannot be used to solve finite-horizon problems efficiently.

In this chapter we presented FiVI, which is a generic point-based value iteration algorithm for finite-horizon problems. FiVI unifies several insights from existing point-based algorithms SARSOP (Kurniawati, Hsu, and Lee, 2008), HSVI (Smith and Simmons, 2005) and GapMin (Poupart, Kim, and Kim, 2011). FiVI is a point-based value iteration algorithm which computes time-dependent value functions, and it leverages a heuristic search procedure to find new belief points incrementally. It is an anytime algorithm which converges to an optimal finite-horizon POMDP solution. In addition to the algorithm itself we also presented two strategies which further improve the algorithm performance. First, we observed that we can employ randomized backup stages similar to the infinite-horizon algorithm Perseus (Spaan and Vlassis, 2005). Second, we made the updates of value upper bounds more efficient by exploiting the insight that these updates typically depend on only a few beliefs that remain constant during execution. Both strategies are complementary, in the sense that they focus on two different steps in FiVI that are computationally difficult, and therefore the strategies can be used simultaneously.

In a series of experiments we tested the performance and characteristics of FiVI, which shows that the algorithm is an effective method for solving finite-horizon problems. This efficacy is also demonstrated by the next chapter of this dissertation, in which the presented algorithms invoke FiVI many times to obtain subproblem solutions while computing solutions to Constrained POMDPs. This shows that FiVI is not only useful for solving standard finite-horizon POMDPs, but also for other variants of the POMDP model which require a finite planning horizon.

Multiple directions of future work can be pursued to improve our algorithm. The main computational bottleneck of FiVI arises in case the POMDP models have a large number of states, and in case there is a large number of beliefs required to reach a solution within the solution quality tolerance. The first problem can be addressed by implementing the algorithm based on sparse representations for alpha vectors. The second problem is more difficult since the convergence of the algorithm is dependent on the ability to sample belief points that are required to reach an optimal solution. It may be possible to derive a more efficient belief sampling scheme which replaces Algorithm 8. For example, domain-specific knowledge may be used to bias the heuristic search, or multiple promising outcomes can be selected during the forward search, similar to work by Zhang et al. (2015). In infinite-horizon algorithms such as GapMin this is addressed by implementing a breadth-first search which prioritizes beliefs that are encountered early, but in the finite-horizon setting it can be expected that this is less effective since rewards are not discounted. This leaves a gap for additional research. Another interesting avenue of future research is the application to POMDPs with a factored problem representation, which may provide additional computational benefits.

5 Approximate planning for Constrained POMDPs

In several real-world domains it is required to plan ahead while there are finite resources available for executing the plan. The limited availability of resources imposes constraints on the plans that can be executed, which need to be taken into account while computing a plan. A Constrained Partially Observable Markov Decision Process (Constrained POMDP) can be used to model resource-constrained planning problems which include uncertainty and partial observability. Constrained POMDPs provide a framework for computing policies which maximize expected reward, while respecting constraints on a secondary objective such as cost or resource consumption. Column generation for linear programming can be used to obtain Constrained POMDP solutions. This method incrementally adds columns to a linear program, in which each column corresponds to a POMDP policy obtained by solving an unconstrained subproblem. Column generation requires solving a potentially large number of POMDPs, as well as exact evaluation of the resulting policies, which is computationally difficult. We propose a method to solve subproblems in a two-stage fashion using approximation algorithms[1]. First, we the finite-horizon algorithm FiVI from the previous chapter to obtain an approximate subproblem solution. Next, we convert this approximate solution into a policy graph, which we can evaluate efficiently. The resulting algorithm is a new approximate method for Constrained POMDPs in single-agent settings, but also in settings in which multiple independent agents share a global constraint.

[1]This chapter is based on an article that appeared in the Journal of Artificial Intelligence Research (Walraven and Spaan, 2018).

Experiments based on several domains show that our method outperforms the current state of the art.

5.1. Introduction

Decision making under uncertainty subject to constraints on cost or resource consumption occurs in several multi-agent systems in the real world. For example, in condition-based maintenance problems it is required to optimize maintenance on multiple assets while taking into account a global constraint on the total maintenance cost (Jardine, Lin, and Banjevic, 2006). This can be a collection of bridges whose partially observable condition deteriorates stochastically over time. Another constrained planning problem occurs in online advertising (Boutilier and Lu, 2016), in which it is required to assign a finite advertisement budget to online users in order to maximize return on investment. A third example exists in demand-side management for smart energy grids, where independent devices want to achieve a certain goal, while taking into account global capacity constraints imposed by the grid (De Nijs, Spaan, and de Weerdt, 2015). For electric vehicles in a smart grid such a goal can be reaching a fully-charged battery as cheaply as possible, which requires power from the grid. In all these application domains it is required that planning algorithms account for potentially many agents, uncertainty and partial observability.

Markov Decision Processes (Puterman, 1994) and Partially Observable Markov Decision Processes (Kaelbling, Littman, and Cassandra, 1998) have emerged as powerful models for planning under uncertainty and planning under partial observability. However, it is not always possible to integrate additional constraints directly into such models defined for a specific domain. For example, Markov Decision Processes (MDPs) and Partially Observable Markov Decision Processes (POMDPs) can be used to maximize an individual reward signal, but unfortunately additional constraints cannot be included in this signal such that the optimal policy respects the constraints during execution. Optimizing policies in which the cost or resource consumption is simply subtracted from the reward does not produce policies which guarantee that constraints are respected. In the context of multi-objective decision making it is possible to assign weights to the reward objective and cost objective (Roijers et al., 2013), after which single-objective algorithms can be used. However, often there is no a priori assignment of weights to objectives available which ensures that constraints on cost are respected while maximizing the total expected reward. Furthermore, optimizing and evaluating policies for all possible assignments of weights to objectives is only tractable for small instances. Based on the aforementioned considerations we conclude that decision making under uncertainty subject to constraints requires specialized algorithms that account for these constraints during optimization.

In order to deal with additional constraints, MDPs and POMDPs have been extended to Constrained MDPs (Altman, 1999) and Constrained POMDPs (Isom, Meyn, and Braatz, 2008). The main idea is that additional cost functions are added to the models, together with an associated cost limit that should be respected in expectation. Constrained MDP solutions are usually computed using a linear programming formulation for MDPs, in which additional constraints can be easily added to the dual formulation. This insight provided the foundation for several constrained optimization algorithms (Dolgov and Durfee, 2003; Wu and Durfee, 2010; Agrawal, Varakantham, and Yeoh, 2016). Constrained POMDPs, on the other hand, are significantly more difficult to solve and received far less attention than the MDP counterpart. There are only a few algorithms, which typically aim to integrate constraints into a traditional unconstrained POMDP algorithm. Point-based value iteration (Pineau, Gordon, and Thrun, 2003) has been generalized to Constrained POMDPs (Kim et al., 2011). In addition, a method has been proposed to optimize finite-state controllers using approximate linear programming (Poupart et al., 2015), which is also based on the linear program used for Constrained MDPs. The aforementioned approaches have two drawbacks. First, they assume an infinite horizon with discounting, which is typically not desirable in application domains. For example, in maintenance problems it can be required to bound the expected resource usage, but the notion of *discounted* resource usage is not well-defined. The second drawback is the scalability, because typically they can only be applied to relatively small instances and they do not provide sufficient scalability to solve larger (e.g., multi-agent) problems.

Another promising method for Constrained POMDPs, which is not a modification of traditional unconstrained algorithms, is based on column generation for linear programming (Yost and Washburn, 2000). The method is based on a master linear program (LP) in which columns correspond to POMDP policies. These columns are incrementally generated by solving a series of unconstrained subproblems, for which traditional POMDP algorithms can be used. Unfortunately, the method has several shortcomings, preventing us from applying it to larger Constrained POMDPs. Most importantly, its scalability is limited since it relies on exact POMDP algorithms for solving the subproblems, such as incremental pruning (Cassandra, Littman, and Zhang, 1997). Replacing the exact algorithms by approximation algorithms is not trivial because it potentially affects the convergence and it requires exact policy evaluation, which can be an expensive operation.

The shortcomings of constrained point-based value iteration, constrained approximate linear programming and exact column generation leave a gap for the development of more sophisticated Constrained POMDP algorithms for both single-agent and multi-agent problems. We use exact column generation as a starting point, and we improve this algorithm by eliminating the need to solve the series of subproblems to optimality.

5.1.1. Contributions

We present and evaluate a novel algorithm for Constrained POMDPs. In particular, we cast the optimization problem for Constrained POMDPs into a linear program in which columns correspond to POMDP policies, and this enables us to use a variety of techniques for linear programs. Our approach is based on the column generation technique introduced by Yost and Washburn (2000), which we enhance by embedding POMDP approximation algorithms, and we apply this approach in a multi-agent setting where multiple agents share a global constraint.

Compared to constrained point-based value iteration and constrained approximate linear programming, we approach optimization for Constrained POMDPs from a rather different angle. Instead of modifying POMDP algorithms to let them take into account constraints, our methods naturally split the optimization problem into a sequence of regular POMDPs that can be solved using traditional unconstrained POMDP algorithms. This gives us several computational advantages and it opens the door to a new class of novel approximation algorithms for solving Constrained POMDPs. To be more specific, the contributions are the following.

First, we define an extension of the standard single-agent Constrained POMDP model, which supports multi-agent planning problems in which multiple agents act independently while taking a global constraint into account. This makes it possible to model constrained planning problems with loosely-coupled agents. Yost and Washburn (2000) described this multi-agent problem as planning for multiple objects. Other Constrained POMDP literature does not refer to such model extensions, and therefore we provide a formal introduction. In contrast to existing Constrained POMDP literature, our model assumes a finite planning horizon, which aligns with many Constrained POMDP application domains.

Second, we revisit a column generation algorithm which can be used to find optimal Constrained POMDP solutions. It does so by generating policies incrementally, for which new columns can be added to a linear program which takes care of the constraints. We provide a new theoretical analysis to further understand the characteristics of the algorithm, which also proves its correctness.

Third, we improve the column generation algorithm by integrating our finite-horizon algorithm FiVI for solving subproblems. This algorithm first computes a vector-based value function, after which we translate the solution into a policy graph. Furthermore, we show how an upper bound on the expected value can be calculated while running the adapted algorithm, which enables us to assess solution quality.

Fourth, we provide an experimental evaluation which shows that our algorithm significantly outperforms the current state of the art. In particular, we describe several problem domains and we present the results of a series of experiments for both single-agent as well as multi-agent problems.

5.1.2. Chapter outline

In Section 5.2 we introduce Constrained POMDPs and an extension suitable for multi-agent planning. In Section 5.3 we introduce an exact algorithm for solving Constrained POMDPs based on column generation for LPs, and we further analyze this algorithm in order to understand its characteristics. In Section 5.4 we describe techniques to solve column generation subproblems using approximate POMDP algorithms, which significantly improves the performance of column generation. In Section 5.5 we provide the results of our experimental evaluation. In Section 5.6 and Section 5.7 we describe related work, our conclusions and future work.

5.2. Constrained POMDPs

In this section we provide a formal introduction to Constrained POMDPs. In a fully observable setting the Constrained MDP framework can be used to model constrained stochastic decision making problems (Altman, 1999). This framework augments a default MDP with an additional cost function and an upper bound on the expected cost incurred during execution. The Constrained POMDP formalism is based on a similar idea and it models constrained stochastic decision making problems which include partial observability (Isom, Meyn, and Braatz, 2008). In this chapter we consider finite-horizon planning problems, because this naturally aligns with Constrained POMDP applications domains, which typically have a finite horizon.

We define a Constrained POMDP using a tuple $M = \langle S, A, O, T, \Omega, R, C, L, b_1, h\rangle$. This tuple is identical to the tuple M used for POMDPs, except that it contains an additional cost function $C : S \times A \rightarrow \mathbb{R}$ and a cost limit L. When executing an action $a \in A$ in state $s \in S$, the agent incurs cost $C(s,a)$. Similar to the reward function, the expected sum of costs $C^{\pi}(t,b)$ incurred by the agent when following policy π from starting from belief b at time t is defined as:

$$C^{\pi}(t,b) = E_{\pi}\left[\sum_{t'=t}^{h} C(b_{t'}, \pi(t', b_{t'})) \;\middle|\; b_t = b \right], \tag{5.1}$$

where $b_{t'}$ is the belief at time t' and $C(b_{t'}, \pi(t', b_{t'})) = \sum_{s \in S} C(s, \pi(t', b_{t'})) b_{t'}(s)$. The cost function C and the limit L reflect the constrained nature of the problem, because the agent aims to maximize the expected sum of rewards while ensuring that the expected sum of costs is upper bounded by L. This optimization problem can be formally stated as follows:

$$\begin{aligned} &\max_{\pi} \; V^{\pi}(1, b_1) \\ &\text{s.t. } C^{\pi}(1, b_1) \leq L. \end{aligned} \tag{5.2}$$

Similar to Constrained MDPs, an optimal policy for a Constrained POMDP may need to randomize over different actions in order to find an appropriate balance between reward and cost (Altman, 1999). It can be shown that the best possible deterministic policy for a Constrained POMDP may be suboptimal (Kim et al., 2011).

In contrast to the fully observable counterpart, Constrained POMDPs received limited attention in the literature. Isom, Meyn, and Braatz (2008) presented an exact dynamic programming update for the constrained setting, which keeps track of both reward and cost. Moreover, it is shown that the pruning operator that is typically found in exact algorithms requires a mixed-integer linear program, rather than the linear program from the non-constrained solution algorithm. In order to address the intractability of exact methods, a constrained variant of point-based value iteration, also known as CPBVI, has been proposed which keeps track of admissible cost while executing backups (Kim et al., 2011). The algorithm CALP aims to approximate the Constrained POMDP using a Constrained MDP defined over belief states, and eventually it produces a finite-state controller respecting the imposed constraint (Poupart et al., 2015). More details about the algorithms and their characteristics are provided in Section 5.3.

5.2.1. Multi-agent Constrained POMDPs

So far we discussed Constrained POMDPs from the perspective of one individual agent which needs to respect a constraint on expected cost. However, we address a larger class of decision making problems which involves multiple independent agents with a shared constraint on cost. These agents are only coupled through their shared constraint, which allows for scalable optimization techniques.

We consider n independent agents that share a common constraint on cost, each of which is modeled using a POMDP which includes cost. For agent i we define the decision making process using a tuple $M_i = \langle S_i, A_i, O_i, T_i, \Omega_i, R_i, C_i, b_{1,i}, h\rangle$, similar to the tuple M used for Constrained POMDPs. It should be noted that the models of the individual agents are completely separated, and the existing definitions from the previous sections can be applied directly to each individual agent. Therefore, the additional subscript i will be used to refer to a specific agent throughout the chapter. The main idea is to find policies $\pi_1, \dots, \pi_n$ for the agents, such that the total expected reward is maximized while the expected sum of costs is bounded:

$$\begin{aligned} \max_{\{\pi_1,\dots,\pi_n\}} \quad & \sum_{i=1}^{n} V_i^{\pi_i}(1, b_{1,i}) \\ \text{s.t.} \quad & \sum_{i=1}^{n} C_i^{\pi_i}(1, b_{1,i}) \leq L. \end{aligned} \tag{5.3}$$

We want to emphasize that the multi-agent formulation above is equivalent to the standard Constrained POMDP model if there is only 1 agent. This means that all presented techniques also apply to Constrained POMDPs with only 1 agent.

5.3. Column Generation for Constrained POMDPs

Approximation algorithms for POMDPs have been widely studied, but the constrained counterpart received only limited attention. Typically, algorithms for Constrained POMDPs have been created by adapting traditional POMDP algorithms for unconstrained problems, and by generalizing algorithms for Constrained MDPs to Constrained POMDPs. An example of the former is CPBVI (Kim et al., 2011), which generalizes point-based value iteration to constrained problems. An example of the latter is CALP (Poupart et al., 2015), which uses solution concepts for Constrained MDPs to create an algorithm which supports partial observability. Unfortunately, both algorithms are potentially affected by scalability problems. CPBVI keeps track of admissible cost while executing point-based backups. This requires solving many linear programs, which slows down the algorithm. CALP defines a linear program over a potentially large number of beliefs, which potentially introduces scalability problems due to the size of this linear program. In both cases the scalability of the algorithms potentially limits the application of existing approximate algorithms for Constrained POMDPs.

Besides the aforementioned scalability problems there is another significant drawback. The algorithms assume that the expected sum of *discounted* costs of the solution should be bounded, but unfortunately this type of constraint is often not useful from a practical point of view. For example, in problems with a constraint on the amount of resources, it would be intuitive to define a constraint on the expected resource consumption. However, the notion of *discounted* resource consumption is typically not well-defined, which means that algorithms for Constrained POMDPs with discounting cannot be applied. Another example consists in domains where it is suitable to use constraints to impose a bound on the probability of an event occurrence. Such constraints can be expressed in the Constrained POMDP formalism, but algorithms which assume discounting in the constraints cannot be used for such problems.

To address both the scalability problems and the problems due to discounting, we build upon a collection of techniques proposed by Yost and Washburn (2000), which approach optimization for Constrained POMDPs from a different angle. They show how the optimization problem can be seen as a linear program defined over the entire policy space, which can be subsequently solved using a column generation algorithm for linear programs. Based on this linear program it is possible to formulate a solution algorithm which does not assume discounting in the constraint. The application of column generation is attractive because it makes it possible to solve a constrained problem as a sequence of unconstrained problems. In the remainder of this section we provide an introduction to the algorithm, and we present an additional mathematical analysis to further understand the characteristics of the algorithm. In Section 5.4 we describe how the scalability of the column generation algorithm can be improved by integrating approximate POMDP algorithms.

5.3.1. Exact column generation for Constrained POMDPs

Optimization problems formulated as an LP can be solved using a conventional LP solver based on, e.g., simplex (Dantzig, 1963) and interior-point methods (Karmarkar, 1984). However, due to the large size of problem formulations it is not always tractable to solve an LP as one individual problem. The main idea of column generation is that large LPs contain only a few variables (i.e., columns) that become non-zero in an optimal solution. Theoretically, only these variables are necessary to characterize an optimal solution. A column generation algorithm incrementally computes columns having the potential to improve the objective function, rather than initializing all the columns immediately. Typically, a column generation algorithm is based on a master LP, which contains only a subset of columns from the original LP. A subproblem is used to identify columns which improve the objective value of the master problem. Column generation can be particularly useful in case the total number of columns is exponential, while searching for new columns can be executed without full enumeration of the exponential column space. The column generation technique was first described by Gilmore and Gomory (1961). For more details about column generation in general we refer to a book by Desrosiers and Lübbecke (2005).

A column generation approach for Constrained POMDPs has been proposed by Yost and Washburn (2000). It uses an LP formulation which defines a probability distribution over policies for each agent, rather than one individual policy for each agent. The LP can be stated as follows:

$$\begin{aligned} \phi = \max & \sum_{i=1}^{n} \sum_{\pi_i \in K_i} V_i^{\pi_i} \cdot x_{i,\pi_i} & & \\ \text{s.t.} & \sum_{i=1}^{n} \sum_{\pi_i \in K_i} C_i^{\pi_i} \cdot x_{i,\pi_i} \leq L & & \text{(dual variable: } \lambda) \\ & \sum_{\pi_i \in K_i} x_{i,\pi_i} = 1 & \forall i \quad & \text{(dual variables: } \lambda_i) \\ & x_{i,\pi_i} \geq 0 & \forall i, \pi_i. & \end{aligned} \tag{5.4}$$

For each agent i the set K_i represents the finite policy space of its finite-horizon POMDP model. The variables x_{i,π_i} represent decision variables corresponding to the probability that agent i uses policy $\pi_i \in K_i$ during execution. The objective function represents the total expected sum of rewards collected by the agents, in which we use $V_i^{\pi_i}$ as a shortcut for $V_i^{\pi_i}(1, b_{1,i})$. Note that this term is a coefficient associated with a variable, and not a variable of the LP. In a similar way the first constraint ensures that the total expected sum of costs is upper bounded by L. Here we use $C_i^{\pi_i}$ as a shortcut for $C_i^{\pi_i}(1, b_{1,i})$. The remaining constraints ensure that the variables constitute valid probability distributions for each agent. For convenience we let ϕ denote the optimal objective value. For each constraint there is a corresponding dual variable, which represent the solution to the dual of the problem. The value assigned to such variables can be obtained from the LP solver after solving the linear program.

The linear program cannot be solved directly because it is intractable to enumerate all possible policies $\pi_i \in K_i$ for each agent. However, a column generation algorithm can be used to generate the policies incrementally, and typically such algorithms require enumerating only a relatively small number of columns. The algorithm maintains a lower bound ϕ_l and an upper bound ϕ_u on the optimal objective value ϕ. A lower bound ϕ_l can be obtained by solving the LP in (5.4) with only a subset of columns. An upper bound ϕ_u can be derived using the following Lagrangian relaxation:

$$\begin{aligned}\phi_u = \max & \sum_{i=1}^{n} \sum_{\pi_i \in K_i} V_i^{\pi_i} \cdot x_{i,\pi_i} + \lambda \left(L - \sum_{i=1}^{n} \sum_{\pi_i \in K_i} C_i^{\pi_i} \cdot x_{i,\pi_i} \right) \\ \text{s.t.} & \sum_{\pi_i \in K_i} x_{i,\pi_i} = 1 \quad \forall i \\ & x_{i,\pi_i} \geq 0 \qquad \forall i, \pi_i,\end{aligned} \tag{5.5}$$

in which λ is the Lagrangian multiplier corresponding to the first constraint in (5.4). Since the constraints only affect individual agents, the upper bound can also be written as:

$$\phi_u = \lambda \cdot L + \sum_{i=1}^{n} \left[\max_{\pi_i \in K_i} \left(V_i^{\pi_i} - \lambda \cdot C_i^{\pi_i} \right) \right]. \tag{5.6}$$

It turns out that the upper bound is easy to compute if we observe that the computation decouples into n separate subproblems. For each agent i the maximization over its policy space can be executed by running a regular POMDP solver, which uses the reward function:

$$G_i(s,a) = R_i(s,a) - \lambda \cdot C_i(s,a). \tag{5.7}$$

After solving these subproblems separately for each agent, we can compute the upper bound ϕ_u. Note that the subproblems of the agents can be solved in parallel.

The full column generation algorithm is shown in Algorithm 11. On lines 2-7 the algorithm starts with initializing the LP shown in (5.4) with only one column for each agent, which we refer to as the master LP. In order to ensure initial feasibility of the master LP, it is assumed that we can always obtain a policy for each agent with minimum expected cost (e.g., always executing the action with lowest cost).

For example, in practice this can be a policy which always executes the action that does not consume any resources. Within the algorithm the sets K_i are used to keep track of the policies for which columns have been added. On lines 8-20 the algorithm repeatedly solves the master LP to obtain dual price λ, after which new policies can be generated for each agent. This procedure repeats until the dual price λ converges, because in that case the new policies generated by the algorithm do not change anymore. Finally, the algorithm returns a set Y_i for each agent, which represents a probability distribution over policies. The description in Algorithm 11 also illustrates how column generation keeps track of the upper bound ϕ_u during execution.

The application of column generation in this context is convenient because it enables us to approach a constrained optimization problem as a sequence of unconstrained optimization problems. Additionally, we want to emphasize that the column generation algorithm produces optimal solutions for Constrained POMDPs. The formulation in Equation 5.4 defines that expected sum of rewards is maximum while the expected sum of costs remains bounded. As we will show in the next section, column generation converges to an optimal solution to the LP in Equation 5.4. Prior to execution each agent i should sample a policy based on the probability distribution defined by Y_i to ensure that the expected cost during execution is bounded while maximizing the reward that is collected in expectation. Agents do not need to communicate with each other during the execution of the selected policies. Moreover, there will be at most 1 agent which needs to randomize its policy choice, as we will show in the analysis in the next section.

5.3.2. Analysis of exact column generation

In this section we study the characteristics of column generation for the setting where exact POMDP solvers are used for solving the subproblems. Our analysis gives additional insight into the behavior of the algorithm, and it was not provided by Yost and Washburn (2000). Moreover, the additional understanding is required in the next sections where solutions to subproblems are computed using approximate algorithms, because such approximate solutions may influence the characteristics of column generation.

Our analysis is based on the concept of reduced cost (Dantzig, 1963; Bradley, Hax, and Magnanti, 1977), which we explain using the following LP formulation in standard form:

$$\begin{aligned} \max\ & c^\top x \\ \text{s.t. } & Ax \leq b \\ & x \geq 0, \end{aligned} \tag{5.8}$$

in which the symbol $\top$ denotes the transpose operator. Note that we use conventional LP notation, which is conflicting with the notation in the definition of

Algorithm 11: Column generation

```
input  : POMDP $M_i$ for each agent $i$, limit $L$
output : probability distribution $Y_i$ over policies for each $M_i$
1  $\phi_l \leftarrow -\infty,\ \phi_u \leftarrow \infty,\ \lambda' \leftarrow \infty,\ \lambda \leftarrow \infty$
2  initialize empty master LP: $K_i \leftarrow \emptyset \ \forall i$
3  foreach $i = 1, \dots, n$ do
4      $\pi_i \leftarrow$ policy for $M_i$ with lowest expected cost
5      compute $V_i^{\pi_i}$ and $C_i^{\pi_i}$ using $\pi_i$
6      add column: $K_i \leftarrow K_i \cup \{\pi_i\}$
7  end
8  do
9      $\lambda' \leftarrow \lambda$
10     solve the master LP to obtain new $\lambda$
11     $\phi_l \leftarrow$ current objective value of the master LP
12     $\phi_u \leftarrow \lambda \cdot L$
13     foreach $i = 1, \dots, n$ do
14         $G_i(s,a) \leftarrow R_i(s,a) - \lambda \cdot C_i(s,a) \quad \forall s \in S_i,\ a \in A_i$
15         solve $M_i$ using $G_i$ to obtain $\pi_i$
16         compute $V_i^{\pi_i}$ and $C_i^{\pi_i}$ using $\pi_i$
17         add column: $K_i \leftarrow K_i \cup \{\pi_i\}$
18         $\phi_u \leftarrow \phi_u + (V_i^{\pi_i} - \lambda \cdot C_i^{\pi_i})$
19     end
20 while $\lambda \neq \lambda'$;
21 $Y_i \leftarrow \{(\pi_i, x_{i,\pi_i}) \mid \pi_i \in K_i$ and $x_{i,\pi_i} > 0\} \quad \forall i$
22 return $\{Y_1, \dots, Y_n\}$
```

POMDPs, but its meaning in this section will be clear from context. We can define a reduced cost vector $\bar{c}$:

$$\bar{c} = c - A^{\top} y, \tag{5.9}$$

in which y is a vector containing the dual prices of the constraints. The reduced cost vector contains a reduced cost value for each column of the LP. The reduced cost of a column j, which is denoted by $\bar{c}_j$, can be interpreted as the rate of change in the objective function when increasing the value assigned to the corresponding variable x_j (Bradley, Hax, and Magnanti, 1977). If the reduced cost of column j is greater than zero (i.e., $\bar{c}_j > 0$), then it holds that the variable x_j has the potential to increase the objective value.

We observe that the LP defined in (5.4) is in standard form if we transform the constraint $\sum_{\pi_i \in K_i} x_{i,\pi_i} = 1$ into two constraints $\sum_{\pi_i \in K_i} x_{i,\pi_i} \leq 1$ and $\sum_{\pi_i \in K_i} -1 \cdot$

$x_{i,\pi_i} \leq -1$ for each agent i. The corresponding dual prices are denoted by $\lambda_{i,0}$ and $\lambda_{i,1}$, respectively. However, we do not need to treat the dual prices of these constraints separately, since the original dual price λ_i of the equality constraint of agent i is defined by $\lambda_i = \lambda_{i,0} - \lambda_{i,1}$. The reason is that increasing the right hand side of the first constraint by 1 corresponds to decreasing the right hand side of the second constraint by 1. Since the dual price corresponds to the rate of change in the objective function, the rate of change when increasing the right hand side of the original equality constraint equals $\lambda_{i,0} - \lambda_{i,1}$.

By applying the definitions of reduced cost to the columns in (5.4), we derive that the reduced cost of a policy π_i is equal to:

$$\begin{aligned}
\bar{c}_{\pi_i} &= V_i^{\pi_i} - \lambda \cdot C_i^{\pi_i} - \lambda_{i,0} \cdot 1 - \lambda_{i,1} \cdot (-1) && (5.10)\\
&= V_i^{\pi_i} - \lambda \cdot C_i^{\pi_i} - \lambda_{i,0} + \lambda_{i,1} && (5.11)\\
&= V_i^{\pi_i} - \lambda \cdot C_i^{\pi_i} - (\lambda_{i,0} - \lambda_{i,1}) && (5.12)\\
&= V_i^{\pi_i} - \lambda \cdot C_i^{\pi_i} - \lambda_i. && (5.13)
\end{aligned}$$

This enables us to establish a relationship between the concept of reduced cost and the computed policies. Below we show that the subproblems solved by Algorithm 11 can be interpreted as computing columns which maximize reduced cost.

Lemma 2. *In each iteration, Algorithm 11 computes a policy π_i for each agent i which maximizes reduced cost.*

Proof. Without loss of generality we consider an arbitrary agent i. In each iteration the algorithm computes a policy π_i for this agent which maximizes:

$$\begin{aligned}
G_i^{\pi_i} &= E_{\pi_i}\left[\sum_{t=1}^{h} G_i(b_t, \pi_i(t, b_t)) \,\middle|\, b_1 = b_{1,i}\right] && (5.14)\\
&= E_{\pi_i}\left[\sum_{t=1}^{h} R_i(b_t, \pi_i(t, b_t)) \,\middle|\, b_1 = b_{1,i}\right] - \lambda \cdot E_{\pi_i}\left[\sum_{t=1}^{h} C_i(b_t, \pi_i(t, b_t)) \,\middle|\, b_1 = b_{1,i}\right] && (5.15)\\
&= V_i^{\pi_i} - \lambda \cdot C_i^{\pi_i}, && (5.16)
\end{aligned}$$

where $G_i(b_t, \pi_i(t, b_t)) = \sum_{s \in S_i} G_i(s, \pi_i(t, b_t)) b_t(s)$. From Equation 5.13 we know that the reduced cost of the newly generated policy π_i is equal to $V_i^{\pi_i} - \lambda \cdot C_i^{\pi_i} - \lambda_i$. Since the last term is a constant regardless of the computed policy π_i, we can conclude that the algorithm computes a policy for agent i which maximizes reduced cost. □

By maximizing reduced cost the algorithm tries to find policies with positive reduced cost, which have the potential to improve the objective of the master LP. It should be noted that finding such columns is equivalent to Dantzig's pivot rule for selecting entering variables in the simplex algorithm (Papadimitriou and Steiglitz, 1982). Before we can show that the column generation algorithm progresses towards an optimal solution, it is important to know whether policies can be generated twice, and how many policies we can potentially generate. This is characterized in Lemma 3 and Lemma 4.

Lemma 3. *If Algorithm 11 generates a policy π_i for which the reduced cost $\bar{c}_{\pi_i}$ is strictly positive, then the policy has not been generated before.*

Proof. Without loss of generality we consider an arbitrary agent i. We assume that Algorithm 11 solves the master LP to optimality and subsequently it generates a policy π_i with strictly positive reduced cost (i.e., $\bar{c}_{\pi_i} > 0$). The reduced cost of policies that have been generated before is zero or negative, which follows from the definition of reduced cost. This is the case because the optimal objective value cannot increase further, and therefore the reduced cost of existing columns cannot be positive. Since the reduced cost of π_i is positive, it follows that π_i has not been generated before. □

Lemma 4. *The master LP in Equation 5.4 has a finite number of distinct columns.*

Proof. A column is defined by the expectations $V_i^{\pi_i}$ and $C_i^{\pi_i}$, which are calculated using Equation 2.23 and Equation 5.1. We consider the computation of the expectation $V_i^{\pi_i}$, which enumerates all reachable beliefs under the execution of π_i starting from the initial belief. We can interpret $V_i^{\pi_i}$ as a function of the beliefs reachable in the POMDP model and the policy π_i used in evaluation. The number of reachable beliefs is finite because we consider a finite-horizon POMDP. During evaluation the policy $\pi_i : \{1, \dots, h\} \times \Delta(S) \rightarrow A$ is invoked based on a finite number of beliefs, and the horizon and the number of actions are finite as well. Both observations together imply that there is a finite number of distinct expectations $V_i^{\pi_i}$ that can be constructed by varying the policy π_i. The same line of reasoning applies to $C_i^{\pi_i}$. Since there is only a finite number of distinct expectations $V_i^{\pi_i}$ and $C_i^{\pi_i}$, it follows that there is a finite number of distinct columns. □

Algorithm 11 terminates if the dual price λ has converged. Before we can prove that the algorithm computes an optimal Constrained POMDP solution, we present two lemmas which we can use to characterize the correct termination of the algorithm.

Lemma 5. *If the master LP solution does not correspond to the optimal Constrained POMDP solution after adding new columns, then the dual price λ changes due to adding the new columns.*

Proof. We consider a setting in which the algorithm retrieves the dual price λ' from the master LP, generates new columns using λ', after which the dual price becomes λ. We assume that the master LP solution does not correspond to the optimal Constrained POMDP solution after generating the new columns, which implies that at least one new column with positive reduced cost exists. We show by contradiction that $\lambda' \neq \lambda$. We assume that $\lambda' = \lambda$. From this it follows that the columns found in the next iteration are identical to the columns found in the previous iteration. The reduced cost of such existing columns is zero or negative. The subproblems in column generation maximize reduced cost (Lemma 2), which implies that new columns with positive reduced cost do not exist. This is a contradiction, because we concluded that there is at least one such column if the master LP solution does not correspond to the optimal Constrained POMDP solution. We can conclude that $\lambda' \neq \lambda$, which means that the dual price changes due to adding new columns. □

Lemma 6. *If the master LP solution corresponds to the optimal Constrained POMDP solution, then the dual price λ becomes constant during the execution of Algorithm 11.*

Proof. The dual price λ follows from the dual solution of the master LP. Since the master LP solution is optimal and its primal solution remains constant in subsequent iterations, it follows that the dual price λ also remains constant in subsequent iterations. □

Based on the lemmas we can prove the correct termination and optimality of Algorithm 11, as shown in Theorems 5 and 6 below.

Theorem 5. *Algorithm 11 terminates if and only if it has found an optimal Constrained POMDP solution.*

Proof. This follows immediately from Lemma 5 and Lemma 6. If the solution to the master LP does not correspond to the optimal Constrained POMDP solution after generating columns, then the dual price λ changes (Lemma 5), which means that the algorithm does not terminate and proceeds with generating columns. If the solution to the master LP corresponds to the optimal Constrained POMDP solution, then the dual price λ will become constant (Lemma 6), which leads to termination. □

Theorem 6. *Algorithm 11 computes an optimal Constrained POMDP solution.*

Proof. Based on Theorem 5 we know that Algorithm 11 keeps generating new columns until reaching an optimal solution, and it never terminates before reaching an optimal solution. Therefore, we only need to show that the algorithm is guaranteed to converge to lower bound $\phi_l = \phi$ in a finite number of iterations. Suppose that it does not, which means that it reaches a lower bound $\phi_l < \phi$ which never further increases in subsequent iterations. The master LP solution does not correspond to the optimal Constrained POMDP solution, which implies that there is at least one new column to be added with positive reduced cost. The algorithm is guaranteed to generate all columns with positive reduced cost in a finite number of iterations because subproblems maximize reduced cost (Lemma 2), columns with positive reduced cost are always new (Lemma 3) and the number of columns with positive reduced cost is finite (Lemma 4). Now it follows that it is guaranteed that ϕ_l eventually increases further. This is a contradiction, because earlier we concluded that the lower bound ϕ_l never increases further in remaining iterations. Now we can conclude that Algorithm 11 is guaranteed to converge to a lower bound $\phi_l = \phi$ in a finite number of iterations, which means that it computes an optimal Constrained POMDP solution. □

As noted earlier, in an optimal solution computed by exact column generation there is a probability distribution over policies for each agent. This means that agents may need to randomize their policy choice prior to execution. In practice it turns out that randomization is limited because we can derive an upper bound on the total number of policies which get a non-zero probability assigned. This is formalized in the theorem below, which shows that there is at most one agent which needs to randomize its policy choice in the final solution.

Theorem 7. *Algorithm 11 computes a solution in which at most one agent needs to randomize its policy choice.*

Proof. For each agent the probability distribution over policies is determined based on a solution satisfying the constraints in the LP defined in (5.4). There are $n+1$ constraints in total, which implies that only $n+1$ variables in the master LP can become non-zero. The reason is that only basic variables of a linear program can take non-zero values, and the number of basic variables is upper-bounded by the number of constraints (Papadimitriou and Steiglitz, 1982). Now it follows that there is at most one agent which has two policies with non-zero probability. □

To summarize, in our analysis in this section we have shown that Algorithm 11 finds optimal Constrained POMDP solutions in which at most one agent randomizes its policy choice. Solving subproblems to optimality quickly becomes intractable, however, due to the limited scalability of exact POMDP algorithms. In the next section we show how a tailored approximate algorithm can be used, in order to mitigate potential scalability problems, and we discuss how this affects the convergence characteristics of the algorithm.

5.4. Approximate algorithms for subproblems

There are several limitations which prevent us from using exact column generation to solve Constrained POMDPs. Exact column generation uses an exact POMDP

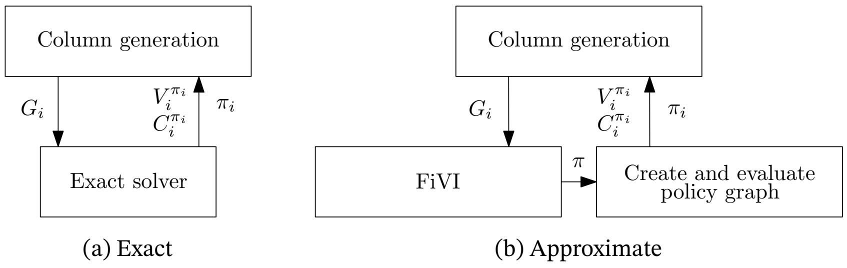


Figure 5.1: Overview of exact column generation and column generation with point-based methods and policy graph generation.

algorithm to solve the subproblems, which may require a significant amount of time and therefore this quickly becomes intractable. Besides the scalability problems, the column generation algorithm assumes that the LP coefficients $V_i^{\pi_i}$ and $C_i^{\pi_i}$ can be computed for a given policy π_i that maximizes $G_i^{\pi_i}$, which we defined in Equation 5.14. These coefficients are required in the LP objective function and the cost constraint, respectively. However, policy evaluation is typically expensive and it may be intractable in practice. Intuitively, the scalability problems can be addressed by solving the subproblems using an approximate POMDP algorithm. However, it still requires policy evaluation, and even in the approximate case this is not always trivial to execute. Additionally, the upper bound ϕ_u computed in Equation 5.6 becomes too tight if the approximate algorithm does not find an optimal solution to the subproblem. This would lead to a situation in which the upper bound computed by the column generation algorithm becomes invalid.

We address the limitations of exact column generation by presenting a two-stage approach to compute solutions to subproblems, based on approximate POMDP algorithms. In particular, we use our finite-horizon algorithm FiVI to obtain an approximate solution to the subproblems. This algorithm provides improved scalability, but obtaining the expected reward and cost of the resulting policies (i.e., the coefficients that we need to insert in the LP) remains expensive. Therefore, we describe a method which converts the solution computed by FiVI into a policy graph, which allows for exact policy evaluation. Finally, we discuss how the techniques can be integrated in the column generation algorithm, and how it keeps track of valid upper bounds ϕ_u while optimizing. A high-level overview of the resulting approach is shown in Figure 5.1, which indicates the differences between exact column generation and column generation based on point-based algorithms and policy graphs. In Figure 5.1a an exact subproblem solution is computed, which immediately gives the coefficients required in the LP. In Figure 5.1b FiVI produces an intermediate policy π, which is converted to a policy graph and subsequently evaluated. The final policy π_i and the LP coefficients are returned to the column generation procedure.

The remainder of this section is structured as follows. In Sections 5.4.1 and 5.4.2 we introduce policy graphs and we describe how they can be created and evaluated. In Section 5.4.3 we discuss our modified column generation algorithm, which is called CGCP. An additional analysis of the graph construction is provided in Section 5.4.4.

5.4.1. Policy graphs as policy representation

Computing value functions using FiVI is relatively efficient compared to exact value iteration. However, given a policy π_i induced by vector sets $\Gamma_1, \dots, \Gamma_h$, it is computationally difficult to obtain the expectations $V_i^{\pi_i}$ and $C_i^{\pi_i}$. It requires evaluation of a tree consisting of all reachable beliefs, and even in the finite-horizon case the construction of this tree can be intractable in terms of both memory and time. Performing such an evaluation many times during the execution of column generation is clearly not possible. It should also be noted that it is not possible to keep track of cost as part of the vectors while executing backups, because this does not provide us with an exact expectation of cost. Such expectations only become exact if the backups are executed on all reachable beliefs, but point-based value iteration algorithms do not guarantee that all these beliefs are enumerated.

We use policy graphs as an alternative to vector-based policies (Kaelbling, Littman, and Cassandra, 1998; Hansen, 1998; Poupart and Boutilier, 2003). Such graphs provide a general formalism for representing POMDP solutions. They consist of a set of nodes, each of which has associated actions and node transitions, which together represent a finite-state controller. After executing the action corresponding to the current node and receiving an observation from the environment, the controller transitions to another node, after which the process repeats. Both the action selection and the node transitions can be stochastic, but we exclusively use deterministic policy graphs. The main motivation for using policy graphs is that policy evaluation is relatively cheap to perform, which enables us to obtain $V_i^{\pi_i}$ and $C_i^{\pi_i}$ without enumerating all reachable beliefs.

Formally, we represent the policy π_i of an agent i using a set of nodes $\mathcal{G}$. Typically we represent a node using the label $q_{t,j} \in \mathcal{G}$, where t refers to a time step and j is the index of the node. The action to be executed in node $q_{t,j}$ is $q_{t,j}^a \in A_i$, and after receiving observation $o \in O_i$ the controller node transitions deterministically to node $q_{t,j}^o \in \mathcal{G}$. This means that $q_{t,j}^o$ refers to another node of the controller, whose time step is $t+1$. Prior to execution the controller starts in node $q^s \in \mathcal{G}$.

An example policy graph is shown in Figure 5.2 for a POMDP with observation set $O_i = \{o_1, o_2, o_3, o_4\}$. Execution starts in node $q_{1,1}$, which is also known as the start node q^s. In this node the agent always executes the action $q_{1,1}^a \in A_i$. For each observation the graph defines a transition to a node in the next layer, corresponding to the next time step. For the example graph it holds that $q_{1,1}^{o_1} = q_{2,1}$, $q_{1,1}^{o_2} = q_{2,3}$,

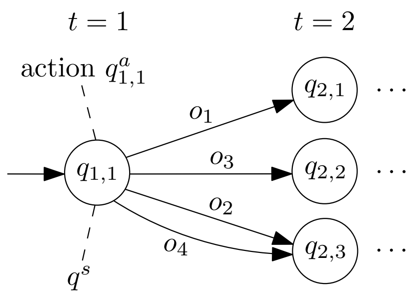


Figure 5.2: Policy graph example

$q^{o_3}_{1,1} = q_{2,2}$ and $q^{o_4}_{1,1} = q_{2,3}$. If the agent executes action $q^a_{1,1}$ and observes o_3, then it transitions to node $q_{2,2}$. The figure shows the graph for just one transition, but the remaining transitions for subsequent steps are defined in a similar manner.

5.4.2. Creating and evaluating a policy graph

A policy graph $\mathcal{G}$ can be constructed in several different ways. There are algorithms which optimize finite-state controllers directly (Poupart and Boutilier, 2003; Grześ, Poupart, and Hoey, 2013; Amato, Bernstein, and Zilberstein, 2010), and they iteratively update a controller in order to improve its quality. They resemble policy iteration techniques, which iteratively evaluate and update a policy. Unfortunately, several of these algorithms can get trapped in a local optimum (Poupart and Boutilier, 2003), they tend to be computationally expensive, and most algorithms have been developed for infinite-horizon problems. Since we need to solve a potentially large number of subproblems during the execution of column generation, we do not want to rely on such expensive algorithms for solving subproblems. Another issue is that our adapted column generation algorithm requires an upper bound on the value of a computed policy, which cannot be easily obtained using algorithms which optimize policy graphs directly. Instead of computing a policy graph directly, we use a method which converts a vector-based policy into a policy graph. By doing so, we maintain the convenient characteristics of point-based value iteration and the value upper bound it produces, while being able to perform policy evaluation efficiently using the policy graph.

We convert the value function induced by $\Gamma_1, \dots, \Gamma_h$ into an approximately equivalent policy graph $\mathcal{G}$, in which each node $q_{t,j} \in \mathcal{G}$ corresponds to a vector $\alpha^j \in \Gamma_t$ from the original solution (Grześ et al., 2015). Algorithm 12 shows how the alpha vectors $\Gamma_1, \dots, \Gamma_h$ can be translated into a policy graph $\mathcal{G}$. The action to be executed in the node $q_{t,j}$ is identical to the action associated with the vector $\alpha^j \in \Gamma_t$. Each node has an outgoing transition for each observation $o \in O_i$. For each action-observation pair, the outgoing transition leads to the node corresponding to the vector providing the highest value for the resulting belief. The policy graph is

Algorithm 12: Generate policy graph (`GeneratePolicyGraph`)

```
input  : POMDP model $M_i$, alpha vectors in a sets $\Gamma_1, \dots, \Gamma_h$
output : policy graph $\mathcal{G}$, start node $q^s$
1  $\mathcal{G} \leftarrow \emptyset$
2  for $t = h, h-1, \dots, 1$ do
3      for $j = 1, \dots, |\Gamma_t|$ do
4          create node $q_{t,j}$
5          $\mathcal{G} \leftarrow \mathcal{G} \cup \{q_{t,j}\}$
6          $a \leftarrow$ action associated with $\alpha^j \in \Gamma_t$
7          $q^a_{j,t} \leftarrow a$
8          $b \leftarrow$ belief using which $\alpha^j \in \Gamma_t$ was generated
9          if $t < h$ then
10             foreach $o \in O_i$ do
11                 if $P(o \mid b, a) > 0$ then
12                     $k \leftarrow \arg\max_{\{\alpha^k \in \Gamma_{t+1}\}_k} \alpha^k \cdot b^o_a$
13                     $q^o_{j,t} \leftarrow q_{t+1,k}$
14                 else
15                     $q^o_{j,t} \leftarrow q_{t+1,1}$
16                 end
17             end
18         end
19     end
20 end
21 $k \leftarrow \arg\max_{\{\alpha^k \in \Gamma_i\}_k} \alpha^k \cdot b_{1,i}$
22 $q^s \leftarrow q_{1,k}$
23 return $(\mathcal{G}, q^s)$
```

equivalent to the original value function in case the policy induced by the vectors is finitely transient (Sondik, 1971; Cassandra, 1998), but in general it is not guaranteed that the policy quality remains the same. An additional discussion regarding policy quality will be provided in Section 5.4.4.

A convenient property is that we can evaluate the quality of the policy graph using a recurrence. We let $V_R(q_{t,j}, s)$ denote the expected sum of rewards received by the agent when the current node is $q_{t,j} \in \mathcal{G}$, the current state is $s \in S_i$, and the agent follows the policy induced by the policy graph afterwards. We can compute this expectation as follows:

$$\mathcal{V}_R(q_{t,j},s) = \begin{cases} R_i(s,q^a_{t,j}) + \sum_{o\in o_i, s'\in S_i} P(s' \mid s, q^a_{t,j}) P(o \mid q^a_{t,j}, s') \mathcal{V}_R(q^o_{t,j}, s') & t < h \\ R_i(s,q^a_{t,j}) & t = h \end{cases}. \tag{5.17}$$

Now we can obtain the exact expected sum of rewards of the policy π_i represented by the policy graph:

$$V_i^{\pi_i} = \sum_{s\in S_i} \mathcal{V}_R(q^s,s)\cdot b_{1,i}(s), \tag{5.18}$$

where $b_{1,i}(s)$ corresponds to the probability that s is the initial state of agent i. In a similar fashion we can obtain the expected sum of costs using the following recurrence:

$$\mathcal{V}_C(q_{t,j},s) = \begin{cases} C_i(s,q^a_{t,j}) + \sum_{o\in o_i, s'\in S_i} P(s' \mid s, q^a_{t,j}) P(o \mid q^a_{t,j}, s') \mathcal{V}_C(q^o_{t,j}, s') & t < h \\ C_i(s,q^a_{t,j}) & t = h \end{cases}. \tag{5.19}$$

The exact expected sum of costs of the policy π_i represented by the policy graph equals:

$$C_i^{\pi_i} = \sum_{s\in S_i} \mathcal{V}_C(q^s,s)\cdot b_{1,i}(s). \tag{5.20}$$

To summarize, for a given policy π_i represented by a policy graph we can use a recurrence to obtain the LP coefficients $V_i^{\pi_i}$ and $C_i^{\pi_i}$, which we can use to generate a new column during the execution of column generation. This evaluation is exact, and it does not require full enumeration of reachable beliefs. The fact that policy evaluation is exact ensures that the newly added column is a valid column of the original master LP in the column generation algorithm. A theoretical analysis of the policy graph construction is provided in Section 5.4.4. In the next section we first describe how FiVI and the policy graphs are integrated in the column generation algorithm.

5.4.3. Adapted column generation algorithm

Exact column generation in Algorithm 11 iteratively generates new columns until the optimal solution has been found. When generating columns using approximate methods, it is no longer guaranteed that the algorithm reaches an optimal solution. Generating policies with approximate methods implies that Lemma 2 is no longer valid because computed policies do not necessarily maximize reduced cost. Lemma 3 is still valid because we are always able to determine the reduced cost $\bar{c}_{\pi_i}$ of a new policy. Lemma 4 is still valid because it does not depend on the solution algorithm used. When using approximate methods the dual price may become constant, even if the algorithm did not reach an optimal solution, which means that Lemma 5 is no longer valid. Lemma 6 is still valid and the dual price will remain constant after reaching optimality, but it is not guaranteed that the algorithm actually reaches such a solution. We conclude that introducing approximate methods for solving subproblems affects the correctness and termination of Algorithm 11, which means that several modifications need to be made. In the remainder of this section we discuss how we modify the traditional column generation algorithm, in such a way that the algorithm is guaranteed to terminate while keeping track of valid lower bounds and upper bounds.

Our first observation is that FiVI may need a significant amount of time to compute a solution to a subproblem. Since we need to solve potentially many subproblems, we want to be able to control the time spent on solving subproblems. Especially during early iterations we do not want to invest a significant amount of time in computing nearly-optimal solutions, because typically the policies generated during early stages (i.e., when λ is not stable yet) do not always occur in the final solution. In general there is a tradeoff between the quality of the subproblem solutions and the running time required to obtain such solutions. In our case we prefer quick computation and evaluation of subproblem solutions over solution quality. Therefore, we introduce a time limit τ for the point-based algorithm FiVI, which we gradually increase during the execution of column generation by adding τ^+ once the objective of the master LP does not improve anymore. In practice this means that the algorithm runs FiVI only for a short period of time during early iterations, such that it is able to compute several initial columns quickly. If the lower bound ϕ_l does not change anymore (i.e., when λ remains constant), we increase the time limit. After increasing the time limit it may be able to compute better policies which it could not generate before. This eventually leads to policies which improve the objective of the master LP. Besides the FiVI time limit τ we also introduce a global time limit $\mathcal{T}$ which ensures that the entire algorithm terminates.

Our second observation is that the upper bound defined in Equation 5.6 is no longer valid since it is not guaranteed that FiVI finds the maximizing policy $\pi_i \in K_i$. However, given the upper bound $\hat{G}_i^{\pi_i}$ computed by FiVI we derive:

$$\phi_u = \lambda \cdot L + \sum_{i=1}^{n} \left[\max_{\pi_i \in K_i} \left(V_i^{\pi_i} - \lambda \cdot C_i^{\pi_i} \right) \right] \leq \lambda \cdot L + \sum_{i=1}^{n} \hat{G}_i^{\pi_i}. \tag{5.21}$$

Note that the upper bound $\hat{G}_i^{\pi_i}$ is denoted by the variable v_u in the description of FiVI Based on the new upper bound we can modify the computation of ϕ_u in the column generation algorithm, such that we obtain a valid upper bound. These bounds are not always tight, especially when the point-based algorithm runs for a short period of time. However, it can be expected that the quality of the upper bound becomes better once FiVI runs longer during later stages of the column generation algorithm.

In Algorithm 13 we present the modified Column Generation algorithm for Constrained POMDPs, which we call CGCP. On line 19 the algorithm invokes FiVI

Algorithm 13: Adapted column generation (CGCP)

```
input  : POMDP $M_i$ $\forall i$, time limit $\mathcal{T}$, FiVI time limit $\tau$, increment
         time $\tau^+$, precision $\rho$, limit $L$
output : probability distribution $Y_i$ over policies for each $M_i$
1  $\phi_l \leftarrow -\infty$, $\phi_u \leftarrow \infty$
2  initialize empty master LP: $K_i \leftarrow \emptyset$ $\forall i$
3  foreach $i = 1, \dots, n$ do
4      $\pi_i \leftarrow$ policy for $M_i$ with lowest expected cost
5      compute $V_i^{\pi_i}$ and $C_i^{\pi_i}$ using $\pi_i$
6      add column: $K_i \leftarrow K_i \cup \{\pi_i\}$
7  end
8  $\mathcal{T}' \leftarrow 0$
9  $\lambda' \leftarrow \infty$
10 do
11     solve the master LP to obtain $\lambda$
12     $\phi_l \leftarrow$ current objective value of the master LP
13     $\phi_u \leftarrow \lambda \cdot L$
14     if $\lambda = \lambda'$ then
15         $\tau \leftarrow \tau + \tau^+$
16     end
17     foreach $i = 1, \dots, n$ do
18         $G_i(s,a) \leftarrow R_i(s,a) - \lambda \cdot C_i(s,a)$ $\forall s \in S_i$, $a \in A_i$
19         $(\Gamma_1, \dots, \Gamma_h, \hat{G}_i^{\pi_i}) \leftarrow$ FiVI($M_i, \rho, \tau, G_i$)
20         $\pi_i \leftarrow$ GeneratePolicyGraph($M_i, \Gamma_1, \dots, \Gamma_h$)
21         compute $V_i^{\pi_i}$ and $C_i^{\pi_i}$ using $\pi_i$ and Equations 5.17-5.20
22         add column: $K_i \leftarrow K_i \cup \{\pi_i\}$
23         $\phi_u \leftarrow \phi_u + \hat{G}_i^{\pi_i}$
24     end
25     $\mathcal{T}' \leftarrow$ elapsed time since the start of the algorithm
26     $\lambda' \leftarrow \lambda$
27     $\phi_a \leftarrow 10^{\lceil \log_{10}(\max(|\phi_l|, |\phi_u|)) \rceil - \rho}$
28 while $\mathcal{T}' < \mathcal{T} \wedge \phi_u - \phi_l > \phi_a$;
29 $Y_i \leftarrow \{(\pi_i, x_{i,\pi_i}) \mid \pi_i \in K_i$ and $x_{i,\pi_i} > 0\}$ $\forall i$
30 return $\{Y_1, \dots, Y_n\}$
```

based on the modified reward function G_i and time limit τ, which gives vector sets $\Gamma_1, \dots, \Gamma_h$ and an upper bound $\hat{G}_i^{\pi_i}$. The policy graph is generated on line 20, which invokes Algorithm 12. After policy evaluation using a recurrence, the policy

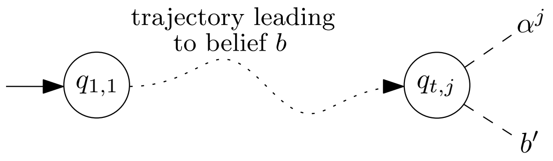


Figure 5.3: Graph trajectory leading to a node $q_{t,j}$

can be added and the upper bound ϕ_u is updated according to Equation 5.21. If the dual price λ remains identical, then the algorithm does not generate new policies anymore, and therefore the time limit of the point-based algorithm is increased on line 15 in those cases. Since it is not guaranteed that the bounds eventually coincide, we use the same termination condition as the gap-based condition in FiVI. The algorithm also terminates if the time limit $\mathcal{T}$ has passed. It can be convenient to use this time limit in case a finite computation time is available and in case optimality is not required.

5.4.4. Analysis of the policy graph construction

In this section we provide an analysis of the translation of vectors $\Gamma_1, \dots, \Gamma_h$ into a policy graph $\mathcal{G}$. This translation is not exact, which means that the solution quality of the policy induced by $\Gamma_1, \dots, \Gamma_h$ is not necessarily the same as the solution quality of $\mathcal{G}$. This can be explained as follows. Algorithm 12 creates a node $q_{t,j}$ corresponding to the vector $\alpha^j \in \Gamma_t$, based on the belief point b' using which α^j was generated. However, if the agent reaches the node $q_{t,j}$ during execution then it may be the case that its current belief b is not identical to b', as visualized in Figure 5.3. For the next time step the policy graph defines a value-maximizing action that was selected for b' rather than b, which can be a different action compared to the action defined by the vector-based policy. This may lead to different behavior in subsequent time steps. This effect becomes less noticeable if FiVI adds more belief points reachable under policy execution to the sets B_t. Then each such reachable belief will have a value-maximizing vector, and the algorithm defines the appropriate node transitions accordingly. Below we offer two approaches to quantify the potential quality difference, as well as a formal theorem to characterize the equivalence of vectors and policy graphs.

Our first approach allows us to quantify the quality difference caused by the translation from vectors to policy graph. FiVI solves a subproblem based on the modified reward function G_i, and in Lemma 2 we concluded that this is equivalent to maximizing reduced cost (i.e., maximizing the potential to improve the LP objective value). For a vector-based policy π_i the expression $\bar{G}_i^{\pi_i} - \lambda_i$ gives us a lower bound on the reduced cost of π_i (see Lemma 2). For the corresponding policy graph $\mathcal{G}$ we can define the actual reduced cost as:

$$\left(\sum_{s\in S_i} \mathcal{V}_G(q^s,s)\cdot b_{1,i}(s)\right)-\lambda_i, \tag{5.22}$$

where

$$\mathcal{V}_G(q_{t,j},s)=\begin{cases} G_i(s,q^a_{t,j})+\sum_{o\in O_i,s'\in S_i} P(s'\mid s,q^a_{t,j})P(o\mid q^a_{t,j},s')\mathcal{V}_G(q^o_{t,j},s') & t<h \\ G_i(s,q^a_{t,j}) & t=h \end{cases}. \tag{5.23}$$

The latter computes the expected value of $\mathcal{G}$ based on the function G_i. Now we can express the change in the lower bound on reduced cost as follows:

$$\left(\sum_{s\in S_i} \mathcal{V}_G(q^s,s)\cdot b_{1,i}(s)\right)-\lambda_i-(\bar{G}^{\pi_i}_i-\lambda_i) \tag{5.24}$$

$$=\left(\sum_{s\in S_i} \mathcal{V}_G(q^s,s)\cdot b_{1,i}(s)\right)-\bar{G}^{\pi_i}_i, \tag{5.25}$$

which can be positive as well as negative. The expression in Equation 5.25 enables us to measure the change in policy quality after translating the vectors $\Gamma_1,\dots,\Gamma_h$ into a policy graph $\mathcal{G}$. Ideally, we want this quantity to be close to zero, and in practice this turns out to be the case, as we will show empirically in our experimental evaluation. Moreover, we want to emphasize that a minor quality loss is acceptable, because the resulting policy always represents a valid column of the master LP.

Our second approach measures the probability that the policy graph $\mathcal{G}$ defines an action which would not be defined by the policy induced by $\Gamma_1,\dots,\Gamma_h$. More formally stated, it measures the probability $\mathcal{P}$ that a graph node $q_{t,j}$ is encountered where the current belief b deviates from b' (see Figure 5.3), and where the prescribed action $q^a_{t,j}$ is not identical to the action defined by the vector $\arg\max_{\alpha\in\Gamma_t} \alpha\cdot b$. An algorithmic procedure to compute the probability is defined in Algorithm 14. The algorithm traverses the beliefs that are reachable during execution of $\mathcal{G}$, and on line 9 it checks whether the belief and action deviate. In those cases it updates the probability $\mathcal{P}$ and no subsequent beliefs are considered (i.e., the execution trajectory terminates). The algorithm shows a breadth-first search, but a depth-first variant can also be implemented if only limited memory is available.

Finally, we formally show that the translation to a policy graph $\mathcal{G}$, defined by Algorithm 12, does not introduce a quality loss if all beliefs reachable during the execution of $\mathcal{G}$ have been sampled. For a policy graph node $q_{t,j}$ it can be observed that the term $\mathcal{V}_G(q_{t,j},\cdot)$, defined in Equation 5.23, represents a vector with an entry for each state. Based on this insight, we can show that $b\cdot\mathcal{V}_G(q_{t,j},\cdot)=b\cdot\alpha_{q_{t,j}}$ for each node $q_{t,j}$, where $\alpha_{q_{t,j}}\in\Gamma_t$ is the vector corresponding to $q_{t,j}$ and b is the belief using which both $\alpha_{q_{t,j}}$ and q were generated.

Algorithm 14: Get the probability that the action defined by $\mathcal{G}$ and $\Gamma_1, \dots, \Gamma_h$ is not identical

```
input  : POMDP model $M_i$, alpha vectors in a sets $\Gamma_1, \dots, \Gamma_h$, graph $\mathcal{G}$
output : probability $\mathcal{P}$
$\mathcal{X}_t \leftarrow \emptyset \quad \forall t, \quad \mathcal{P} \leftarrow 0$
$\mathcal{X}_1 \leftarrow \mathcal{X}_1 \cup \{(q^s, b_{1,i}, 1)\}$
for $t = 1, \dots, h$ do
    for $(q_{t,j}, b, p) \in X_t$ do
        $\alpha^j \leftarrow$ vector $\alpha \in \Gamma_t$ corresponding to $q_{t,j}$
        $b' \leftarrow$ belief using which $\alpha^j$ was generated
        $a' \leftarrow$ action associated with $\arg\max_{\alpha \in \Gamma_t} \alpha \cdot b$
        $a \leftarrow q^a_{t,j}$
        if $b \neq b' \wedge a \neq a'$ then
            $\mathcal{P} \leftarrow \mathcal{P} + p$
        else if $t < h$ then
            for $o \in O_i$ do
                if $P(o \mid b, a) > 0$ then
                    $\mathcal{X}_{t+1} \leftarrow \mathcal{X}_{t+1} \cup \{(q^a_{t,j}, b^o_a, p \cdot P(o \mid b, a))\}$
                end
            end
        end
    end
end
return $\mathcal{P}$
```

Lemma 7. *Given belief sets $B_1, \dots, B_h$, vector sets $\Gamma_1, \dots, \Gamma_h$ and the corresponding policy graph $\mathcal{G}$. If it holds for each $t = 1, \dots, h-1$ that all beliefs reachable from B_t are present in B_{t+1}, then it holds for each node $q_{t,j} \in \mathcal{G}$ that $b \cdot \mathcal{V}_G(q_{t,j}, \cdot) = b \cdot \alpha_{q_{t,j}}$ In this equation the vector $\alpha_{q_{t,j}} \in \Gamma_t$ denotes the vector corresponding to node $q_{t,j}$ and b is the belief using which the node and vector were generated.*

Proof. We prove this by mathematical induction over time steps $t = h, h-1, \dots, 1$. As a base case we consider $t = h$. For each $b_j \in B_h$ the point-based algorithm produces a vector α^j using Equation 4.6, which yields the immediate reward vector for a value-maximizing action. We denote this action by a^*, and the node corresponding to b_j is denoted by $q_{h,j}$. Based on the construction of the graph we know that it holds that $q^a_{h,j} = a^*$. When computing the vector $\mathcal{V}_G(q_{t,j}, \cdot)$ using Equation 5.23 we derive the same immediate reward vector. It follows that $b_j \cdot \mathcal{V}_G(q_{t,j}, \cdot) = b_j \cdot \alpha^j$, which means that the theorem holds for $t = h$.

In our induction hypothesis (IH) we assume that the theorem holds for nodes $q_{t+1,j} \in \mathcal{G}$ (i.e., for step $t+1$). Assuming that the hypothesis holds for step $t+1$, we show that the theorem also holds for step t. For each node $q_{t,j} \in \mathcal{G}$ at time t we can derive $b \cdot \mathcal{V}_G(q_{t,j}, \cdot) = b \cdot \alpha_{q_{t,j}}$, where b is the belief using which both $q_{t,j}$ and $\alpha_{q_{t,j}}$ were generated. Without loss of generality we use q as a shortcut for $q_{t,j}$ in the derivation for readability reasons.

$$
\begin{aligned}
& b \cdot \mathcal{V}_G(q, \cdot) && \\
&= \sum_s b(s) \mathcal{V}_G(q,s) && \\
&= \sum_s b(s) \left(G(s,q^a) + \sum_{o,s'} P(s'|s,q^a) P(o|q^a,s') \mathcal{V}_G(q^o,s') \right) && \text{def. } \mathcal{V}_G \\
&= \sum_s b(s) G(s,q^a) + \sum_s b(s) \sum_{o,s'} P(s'|s,q^a) P(o|q^a,s') \mathcal{V}_G(q^o,s') && \\
&= b \cdot G(q^a) + \sum_s \sum_o \sum_{s'} P(s'|s,q^a) P(o|q^a,s') b(s) \mathcal{V}_G(q^o,s') && \\
&= b \cdot G(q^a) + \sum_o \sum_{s'} P(o|q^a,s') \sum_s P(s'|s,q^a) b(s) \mathcal{V}_G(q^o,s') && \\
&= b \cdot G(q^a) + \sum_o \sum_{s'} \mathcal{V}_G(q^o,s') P(o|q^a,s') \sum_s P(s'|s,q^a) b(s) && \\
&= b \cdot G(q^a) + \sum_o \sum_{s'} \mathcal{V}_G(q^o,s') P(o|b,q^a) b^o_{q^a}(s') && \text{def. belief update} \\
&= b \cdot G(q^a) + \sum_o \sum_{s'} P(o|b,q^a) b^o_{q^a}(s') \mathcal{V}_G(q^o,s') && \\
&= b \cdot G(q^a) + \sum_o P(o|b,q^a) \sum_{s'} b^o_{q^a}(s') \mathcal{V}_G(q^o,s') && \\
&= b \cdot G(q^a) + \sum_o P(o|b,q^a) (b^o_{q^a} \cdot \mathcal{V}_G(q^o,\cdot)) && \\
&= b \cdot G(q^a) + \sum_o P(o|b,q^a) (b^o_{q^a} \cdot \alpha_{q^o}) && \text{IH, graph construction} \\
&= b \cdot G(q^a) + \sum_o \sum_{s'} P(o|b,q^a) b^o_{q^a}(s') \alpha_{q^o}(s') && \\
&= b \cdot G(q^a) + \sum_o P(o|b,q^a) \sum_{s'} b^o_{q^a}(s') \alpha_{q^o}(s') && \\
&= b \cdot G(q^a) + \sum_o P(o|b,q^a) \max_{\alpha \in \Gamma_{t+1}} \sum_{s'} b^o_{q^a}(s') \alpha(s') && \text{graph construction} \\
&= b \cdot G(q^a) + \sum_o P(o|b,q^a) V_{t+1}(b^o_{q^a}) && \text{def. value function} \\
&= \max_a \left[b \cdot G(a) + \sum_o P(o|b,a) V_{t+1}(b^o_a) \right] && q^a \text{ is maximizing} \\
&= b \cdot \texttt{backup}(b) && \text{backup using } G(s,a) \\
&= b \cdot \alpha_q && \alpha_q \text{ generated with } b
\end{aligned}
$$

Non-trivial steps have been justified in the right column. Below we further eleborate on the step where the induction hypothesis (IH) is invoked.

We assume that Algorithm 12 breaks ties on line 12 by selecting the vector for which the corresponding belief is the closest to b_a^o (e.g., using the absolute difference). This may occur if there are multiple value-maximizing vectors providing exactly the same value in b_a^o. We can invoke the induction hypothesis (IH) based on the following line of reasoning:

1. We know that node q^o is a node at time step $t+1$, because q is a node at time t and q^o is a shortcut for a node in the next layer of the policy graph.
2. We know that node q^o has a corresponding vector $\alpha_{q^o} \in \Gamma_{t+1}$. This follows immediately from the policy graph construction procedure.
3. We know that node q was generated with b. Therefore, we also know that node q^o corresponds to the node at time $t+1$ for which α_{q^o} is value-maximizing in belief $b_{q_a}^o$. This follows from the description of Algorithm 12 and the fact that action q_a is associated with q.
4. The theorem assumes that all beliefs reachable from B_t are present in B_{t+1}. Under this assumption, and under the assumption that the algorithm breaks ties as described above, we know that α_{q^o} was generated using $b_{q_a}^o$.
5. There is a direct correspondence between nodes and vectors, and therefore we know that q^o was generated with $b_{q_a}^o$ as well.
6. Both q^o and α_{q^o} belong to time $t+1$ and were generated using $b_{q^a}^o$, and hence we can write $b_{q^a}^o \cdot V_G(q^o, \cdot) = b_{q^a}^o \cdot \alpha_{q^o}$, following our induction hypothesis.

All other derivation steps follow from definitions, the description in Algorithm 12, and the fact that actions associated with nodes and vectors are value-maximizing. In the call to `backup` in the derivation we discarded additional arguments because it is only relevant that the backup is executed on b for time t. Based on the induction principle we conclude that the theorem holds for all time steps $t = 1, 2, \ldots, h$. □

Theorem 8. *Given belief sets $B_1, \ldots, B_h$, vector sets $\Gamma_1, \ldots, \Gamma_h$ and the corresponding policy graph $\mathcal{G}$. If it holds for each $t = 1, \ldots, h-1$ that all beliefs reachable from B_t are present in B_{t+1}, then the reduced cost of the policy induced by $\Gamma_1, \ldots, \Gamma_h$ is the same as the reduced cost of $\mathcal{G}$.*

Proof. From Lemma 2 we know that the point-based algorithm maximizes reduced cost, and $\bar{G}_i^{\pi_i} - \lambda_i$ provides a lower bound on the exact reduced cost of the vector-based policy. It is assumed that all beliefs reachable from the initial belief are present in the belief sets $B_1, \ldots, B_h$, and therefore the lower bound becomes tight, which means that $\bar{G}_i^{\pi_i} - \lambda_i$ represents the actual reduced cost of the vector-based policy. Now we derive $\bar{G}_i^{\pi_i} - \lambda_i = \max_{\alpha \in \Gamma_1} b_{1,i} \cdot \alpha - \lambda_i = b_{1,i} \cdot \alpha_{q^s} - \lambda_i = b_{1,i} \cdot \mathcal{V}_G(q^s, \cdot) - \lambda_i$, in which we use $b_{1,i} \cdot \mathcal{V}_G(q^s, \cdot) = b_{1,i} \cdot \alpha_{q^s}$ based on Lemma 7. The final term in the derivation is identical to the exact reduced cost of the graph-based policy, as defined in Equation 5.22, and therefore we can conclude that the reduced cost of both representations is identical. □

To summarize, we have analyzed the potential difference in policy quality introduced by the translation from vectors to graph. For a given set of vectors and the corresponding graph we can compute the difference in reduced cost, which directly relates to the potential to improve the objective of the master LP. Moreover, our lemma and theorem state under which conditions Algorithm 12 provides an exact translation to a policy graph.

5.5. Experiments

In this section we present our experimental evaluation based on single-agent and multi-agent planning problems which include constraints. For single-agent problems we evaluate CGCP by comparing it with a finite-horizon version of CALP (Poupart et al., 2015). Originally CALP has been designed for infinite-horizon problems, but the algorithm can be easily generalized to finite-horizon settings. Additional details about this generalization can be found at the end of this section. The algorithm CPBVI (Kim et al., 2011) has been designed for infinite horizons and it does not guarantee that constraints are respected. Therefore, it is not considered in our evaluation. It should be noted, however, that CALP outperforms existing solution algorithms for Constrained POMDPs and thus we compare with the current state of the art. We implemented the algorithms using Java version 8, and the experiments were executed on an Intel Xeon 3.70 GHz CPU with a 5 GB memory limit. For solving LPs we use Gurobi version 6.5.2. For multi-agent problems we only consider CGCP, because CALP has been designed for single-agent problems. Solving multi-agent problems using CALP would lead to underlying MDP models which scale exponentially in the number of agents. More details about the problem domains and the experimental setup are provided in subsequent sections.

5.5.1. Single-agent planning: robot navigation domains

We first consider single-agent robot navigation problems from `pomdp.org`. In these domains a robot is tasked to reach the goal state, which gives reward 1000. The robot is unable to execute an infinite number of actions (e.g., due to a limited battery capacity), and therefore we aim to bound the expected number of actions executed. The domains have been modified, such that the goal state leads to a trap state that cannot be left. Otherwise the robot would restart from the initial belief,

			CGCP			CALP		
Domain	h	L	R	Gap	Time (s)	R	Gap	Time (s)
MiniHall	10	1	283.33	0.00	0.47	283.33	0.00	**0.33**
	10	2	472.22	0.00	**0.21**	472.22	0.00	0.22
	10	3	630.95	0.00	0.19	630.95	0.00	**0.15**
	10	4	773.81	0.00	0.17	773.81	0.00	**0.16**
Cheese	10	1	325.00	0.00	**0.34**	325.00	0.00	0.37
	10	2	575.00	0.00	0.15	575.00	0.00	**0.14**
	10	3	780.00	0.00	**0.08**	780.00	0.00	0.12
	10	4	950.00	0.00	**0.07**	950.00	0.00	0.18
4x3	10	1	**258.88**	**0.05**	**1.33**	255.85	3.05	22.64
	10	2	**462.90**	**0.27**	**1.36**	458.26	4.66	26.68
	10	3	**645.46**	**0.12**	**1.18**	639.38	6.09	31.67
	10	4	**815.56**	**0.14**	**0.96**	811.17	4.53	28.40
Maze20	10	1	**60.22**	**0.01**	**398.07**	46.73	44.62	774.64
	10	2	**118.66**	**0.04**	**301.31**	64.58	104.62	623.88
	10	3	**159.70**	**7.70**	**1107.91**	67.94	163.99	1821.55
	10	4	**182.49**	**14.48**	**1106.59**	62.01	199.29	2306.13
Hallway	10	1	**110.88**	**77.37**	**1025.98**	42.82	165.00	1924.76
	10	2	**166.65**	**94.44**	**1026.10**	68.93	236.63	2026.85
	10	3	**206.54**	**101.54**	**1122.61**	84.03	278.66	2420.72
	10	4	**240.16**	**102.25**	**1024.78**	88.71	313.80	2096.91

Table 5.1: Comparison of CGCP and CALP on navigation domains

which is not desirable in our experiments. A full description of the modifications to the benchmark domains is discussed in Section 5.5.5.

We compare CGCP and the finite-horizon version of CALP. We run CGCP with a time limit of 1000 seconds (i.e., $\mathcal{T} = 1000$). We solve the subproblems initially for at most 100 seconds, and upon convergence this is incremented by 100 seconds (i.e., $\tau = \tau^+ = 100$). We use a time limit because otherwise CALP runs much longer on several domains, and in that case the algorithm typically runs out of memory, which would not give a fair comparison between the two algorithms. We use precision $\rho = 3$, and the time limit of CALP is set to the actual runtime of CGCP on the same instance, with a minimum of 20 seconds. This ensures that CALP has at least as much time available as CGCP. The reported running times of CALP in the table may be higher, since it ends execution with a binary search to create the final solution, and the running time of this search is also included in the reported running time. The reported running times of CGCP may be higher, because we only terminate after completion of an iteration of CGCP. We assume that the robot is able to execute at most h actions, which can be `IDLE` or a `MOVE` action. We also impose the upper bound L on the expected number of `MOVE` actions.

Table 5.1 shows the results, where the column R refers to the exact expected reward collected by the robot. Note that this is a lower bound on the optimal expected reward that can be collected. Gap refers to the difference between the expected reward and the upper bound on the expected reward of an optimal solution. A gap of 0 implies that the computed solution is optimal since the lower bound and upper bound coincide, and in general a smaller gap represents a better solution. It should be noted that in some domains, such as Hallway, the worst-case path in the maze is longer than 4 steps. However, it is still possible to have a positive expected reward because the initial position is defined by the initial belief over states, and from some of these positions in the maze the goal can be reached within L steps. For all instances the constraint on expected cost is tight, which means that there is no gap between L and the expected cost of the solution. The column Time shows the measured running times in seconds. In each row the bold entries indicate the best-performing algorithm.

We observe that both CGCP and CALP are able to compute optimal solutions in the small domains MiniHall and Cheese. In larger domains the CGCP method starts to perform significantly better than CALP. As can be seen in the table, the expected reward (i.e., the column R) of the solutions is much higher, the gap is significantly smaller, and CGCP required less time to compute the solution. In these domains CGCP clearly outperforms CALP on all fronts.

5.5.2. Multi-agent planning: condition-based maintenance

Condition-based maintenance is an emerging practice to reduce the operational cost and maintenance cost for systems whose condition and operating performance deteriorates over time (Jardine, Lin, and Banjevic, 2006). Rather than performing scheduled maintenance on a regular basis, inspections and sensor diagnostics can provide information based on which maintenance can be scheduled before critical components fail. Maintenance cost can be reduced and utilization of personnel and resources can be improved by executing maintenance at the right time and only when necessary. Examples of condition-based maintenance include the maintenance of wind turbines (Byon and Ding, 2010), railway equipment (Fararooy and Allan, 1995), bridges (Neves and Frangopol, 2005) and aircraft components (Harman, 2002).

Our Multi-Agent Constrained POMDP model naturally applies to condition-based maintenance tasks in which multiple objects should be kept in a good condition while bounding the expected maintenance cost. From a planning point of view the current condition of these objects is partially observable (Kim, Choi,

n	L_m	L	R	Gap	Std reward	C	Std cost	Time (s)
2	0.8	667.23	17001.14	6.95	797.21	667.23	118.89	484
	0.6	500.42	16861.91	13.81	810.76	500.42	68.41	362
	0.4	333.61	16437.60	39.26	870.69	333.65	67.16	485
	0.2	166.81	14920.06	77.21	1066.55	166.83	37.29	484
3	0.8	1074.75	25533.55	80.56	965.99	1074.75	146.78	535
	0.6	806.06	25335.12	61.88	994.61	806.06	99.43	540
	0.4	537.37	24862.29	41.44	1076.49	537.37	79.09	716
	0.2	268.69	23072.89	24.41	1243.07	268.67	56.64	717
4	0.8	1337.98	34043.17	14.34	1125.47	1337.98	150.34	968
	0.6	1003.48	33793.92	37.41	1130.76	1003.48	117.64	726
	0.4	668.99	33000.47	24.10	1233.51	668.99	92.52	968
	0.2	334.49	29821.64	8.54	1515.81	334.49	58.76	1210
5	0.8	1664.06	42747.63	92.30	1175.27	1664.06	120.28	908
	0.6	1248.04	42207.48	69.68	1223.84	1248.04	125.92	906
	0.4	832.03	41151.80	35.46	1311.70	832.03	96.16	1207
	0.2	416.01	36899.35	9.23	1501.08	416.01	68.37	1508
6	0.8	2289.34	51619.98	5.23	1256.40	2289.34	142.98	1452
	0.6	1717.00	51065.92	56.83	1327.70	1717.00	158.44	1090
	0.4	1144.67	49830.45	37.40	1427.64	1144.67	110.78	1453
	0.2	572.34	45704.99	69.22	1635.37	572.34	82.43	1451

Table 5.2: Performance of CGCP on maintenance instances

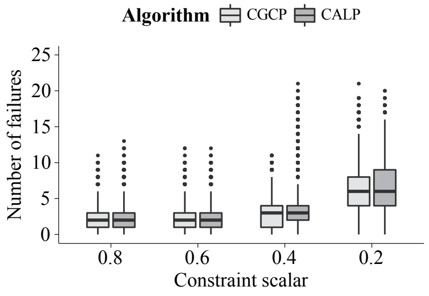
and Lee, 2018), because noisy sensor readings do not provide perfect information regarding the actual condition. If there are no sensor readings at all, then the actual condition is also unknown. The actual condition only becomes available when performing a manual inspection. Moreover, the multi-agent aspect is highly relevant if organizations perform maintenance on multiple different objects which deteriorate independently. A concrete example is a road authority which performs maintenance on several bridges that are part of the road infrastructure. If only a finite maintenance budget is available, then the question becomes how this budget should be distributed over the bridges in order to perform the required maintenance.

In order to demonstrate the efficacy of our CGCP algorithm, we consider the aforementioned maintenance problem in which a road authority is tasked to perform maintenance on several bridges, such that they remain in a good condition. The authority aims to execute the maintenance in the best possible way, given a constraint on the expected amount of money it spends on all maintenance operations during, e.g., one year. In particular, we consider the partially observable bridge repair problem introduced by Ellis, Jiang, and Corotis (1995), for which a description can be found on `pomdp.org`. We use this description to model n bridges requiring maintenance, in which the reward is proportional to the current condition. In other words, the road authority has an incentive to perform maintenance on the collection of bridges. Three repair actions are available, each of which has a cost associated with it, and the road authority aims to upper bound the total expected cost by L. There is no cost associated with the action that does not perform maintenance at all. We added noise to the transition model of the bridges to ensure that they have slightly different state transition characteristics defining the deterioration process. More details regarding the domain can be found in Section 5.5.5.

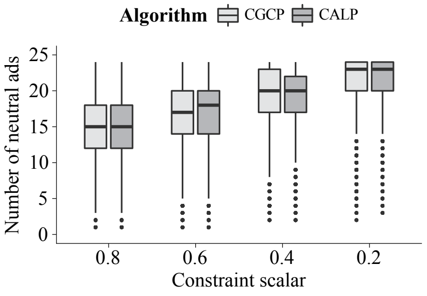
We create instances with an increasing number of agents n. For each instance, we first compute the expected cost C_u of the unconstrained problem, which we can use to define several constrained instances. To be more specific, we can define the cost limit $L = C_u \cdot L_m$, where L_m is a scalar in the range between 0 and 1. This way, we can naturally parameterize the constraint of the instance using a number in a fixed range. We run CGCP with a time limit of 3600 seconds (i.e., $\mathcal{T} = 3600$), and for FiVI we initially solve during 60 seconds, which can be incremented by 60 seconds after converging (i.e., $\tau = \tau^{+} = 60$). We use precision $\rho = 3$.

The results of our evaluation are shown in Table 5.2 for $h = 24$, $n \in \{2, 3, \ldots, 6\}$ and $L_m \in \{0.2, 0.4, 0.6, 0.8\}$. The column R denotes the exact expected reward, which is a lower bound on the optimal expected reward, and Gap indicates the gap between the computed lower bound and upper bound. The column C indicates the exact expected cost of the computed solution, and the column Time shows the required running time in seconds. Note that the expectations in the columns R and C are exact. We measured the standard deviation for both reward and cost using 10^6 simulation runs, and these statistics have also been added to the table. We conclude that CGCP is able to compute high-quality solutions while bounding the expected cost. Considering the relatively small gap compared to the expected reward, we can conclude that the algorithm computes solutions which are close to optimal. The running time increases when increasing the number of agents, but it should be noted that we solved all subproblems sequentially in our evaluation. Since all subproblems are independent, they can be parallelized given sufficient cores, which alleviates this increase in runtime. From a practical point of view, we conclude that our algorithm is able to compute near-optimal maintenance policies while bounding the total expected cost spent on maintenance.

Our previous experiment determines policy quality in terms of the expected reward collected by the agents during execution. This enables us to assess the optimality of the policies, but it does not provide much insight into the actual



(a) Bridge failures



(b) Neutral advertising

Figure 5.4: Behavior of policies during simulation

policy behavior when making the constraint more tight. We execute an additional experiment for one bridge with horizon 24 (i.e., $h = 24$), and during simulation of the resulting policy we measure the number of times the fail state is reached. Intuitively, we expect that this occurs more often if we decrease the budget available for maintenance. The results are shown in Figure 5.4a, which visualizes the number of failures for several cost constraints defined using the constraint scalar L_m. Since we consider only one bridge, we also include policies computed by CALP in our evaluation. We can conclude that a decrease in maintenance budget leads to more failures of the bridge, which is natural since less maintenance can be performed. For this single-agent instance we observe that CGCP and CALP provide similar performance. However, if it is required to perform maintenance on multiple bridges, then it is no longer convenient to use CALP due to the exponential scalability, as discussed before.

5.5.3. Multi-agent planning: online advertising

Online advertising involves deciding which advertisements need to be presented to multiple target customers in order to maximize profit (Boutilier and Lu, 2016). As an example we can consider a company which sells several types of products. Each user can be interested in a subset of these products, or it has no interest at all, and therefore we want to present relevant advertisements to the user with an appropriate intensity. Effects of these advertisements on the user behavior are stochastic, and the type of interest of the user is partially observable since it needs to be inferred from, e.g., browsing behavior. Typically there is a finite budget for advertising, which makes the problem constrained. A similar recommendation problem occurs in systems which recommend points of interest to tourists. In such systems points of interest have limited capacity and the user type needs to be learned from observed user behavior (De Nijs et al., 2018).

We can use our Multi-Agent Constrained POMDP model to formalize an online

n	L_m	L	R	Gap	Std reward	C	Std cost	Time (s)
2	0.8	16.34	1.94	0.00	0.04	16.34	5.70	485
	0.6	12.25	1.93	0.00	0.03	12.25	5.34	399
	0.4	8.17	1.92	0.00	0.03	8.17	5.34	353
	0.2	4.08	1.90	0.00	0.04	4.08	3.60	343
3	0.8	35.47	2.92	0.00	0.05	35.47	7.72	606
	0.6	26.60	2.91	0.00	0.04	26.60	8.58	879
	0.4	17.73	2.90	0.00	0.04	17.73	6.86	691
	0.2	8.87	2.87	0.00	0.04	8.87	5.38	545
4	0.8	37.24	3.89	0.00	0.05	37.24	8.12	814
	0.6	27.93	3.87	0.01	0.05	27.93	7.98	645
	0.4	18.62	3.85	0.01	0.05	18.62	7.09	659
	0.2	9.31	3.82	0.01	0.05	9.30	5.33	506
5	0.8	55.15	4.87	0.00	0.06	55.15	9.72	960
	0.6	41.36	4.86	0.00	0.06	41.36	8.77	1461
	0.4	27.57	4.83	0.00	0.05	27.57	8.89	1150
	0.2	13.79	4.79	0.00	0.05	13.79	6.55	1047
6	0.8	90.42	5.85	0.01	0.07	90.42	9.85	1388
	0.6	67.82	5.83	0.01	0.06	67.82	8.90	1992
	0.4	45.21	5.81	0.01	0.06	45.21	10.40	2577
	0.2	22.61	5.78	0.01	0.05	22.61	8.37	1640

Table 5.3: Performance of CGCP on advertising instances

advertising problem involving multiple users. Each user is modeled as a POMDP, in which the state represents the user type reflecting the level of interest in a certain product. Actions correspond to several different advertising campaigns, which affect the level of interest of the user and its willingness to buy a product. The observations of the model represent the observable user behavior, such as the page it selects or the search query it executes. The advertising campaigns have cost associated with them, and we would like to incentivize users to buy the products while bounding the expected amount of money spent on advertising.

In our experiment we demonstrate that our CGCP algorithm can be used to solve online advertising instances involving several partially observable users. We use the web-ad domain description from `pomdp.org` to model an individual user. In particular, we create n users which independently browse on a website, and we impose an upper bound L on the total expected advertising cost. Actions corresponding to advertising for specific products have cost associated with it, and it is assumed that neutral advertising has cost zero. This way, the cost associated with advertisements for specific products can be interpreted as the additional cost compared to displaying neutral advertisements all the time. Similar to the condition-based maintenance experiment we added noise to the state transitions to ensure that users have slightly different transition dynamics. More details about the domain can be found in Section 5.5.5.

Similar to the previous experiment, we create instances with an increasing number of agents n. We run CGCP with a time limit of 3600 seconds (i.e., $\mathcal{T} = 3600$), and for the FiVI we initially solve during 60 seconds, which can be incremented by 60 seconds after converging (i.e., $\tau = \tau^+ = 60$). We use planning horizon $h = 24$ and precision $\rho = 3$. The results are shown in Table 5.3 for $n \in \{2, 3, \ldots, 6\}$ and $L_m \in \{0.2, 0.4, 0.6, 0.8\}$. Similar to the condition-based maintenance experiment, we want to emphasize that subproblems can be solved in parallel if there are more target customers involved. The table shows that our algorithm effectively bounds the expected cost, and the gap close to zero shows that the computed solutions are nearly optimal.

Similar to the condition-based maintenance experiment, we also study the behavior of policies when making the constraint on expected cost more tight. Figure 5.4b shows the number of neutral ads shown to a user as a function of the constraint scalar L_m, for both CGCP and CALP. As expected, we can see that a decrease in available budget leads to an increase in neutral (i.e., cheap) advertising. When almost no budget is available (i.e., $L_m = 0.2$), the policy gets extremely conservative and it displays neutral ads almost always. As can be seen in the figure, in this domain the policies computed by both CGCP and CALP behave similarly. However, again it should be noted that CALP does not provide immediate support for instances involving multiple users, which is always the case in realistic online advertising problems.

5.5.4. Translation of vectors into policy graph

In Section 5.4.4 we explained why the translation from vectors into a policy graph may lead to a decrease in solution quality. Moreover, we described a method to quantify the quality difference for a given vector set and its corresponding policy graph. In particular, Equation 5.25 describes how the change in the value lower bound can be computed. This allows us to assess the quality differences introduced by the policy graph translation. For the domains Hallway, Hallway2, Maze20, 4x3, condition-based maintenance and advertising we took an unconstrained instance, which we solved using FiVI. In some domains convergence to optimality takes a long time, and therefore we terminate the algorithm after 200 iterations. In each iteration of the algorithm we store the current vector set, and we translate this set of vectors into a policy graph. Subsequently, we compute the change in the value lower bound as a percentage, which is visualized in Figure 5.5. Each dot in the

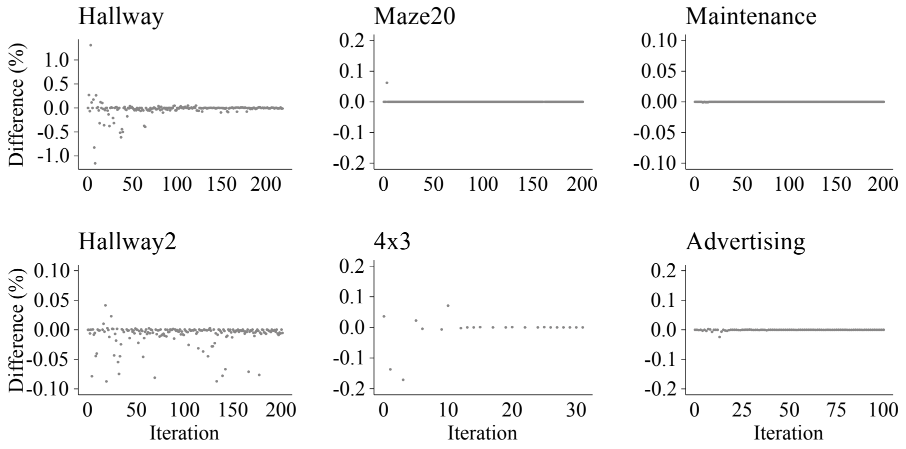


Figure 5.5: Lower bound difference as a function of the number of iterations

figure represents the quality difference encountered in a particular iteration, and ideally these points are all close to zero. As can be seen, the quality differences introduced by the policy graph translation are negligible. For example, in Hallway, Hallway2 and 4x3 differences in solution quality can be observed, but these dots correspond to a quality change that is less than 1 percent. In the other domains the quality difference is even lower.

The results indicate that there are some differences in solution quality, as we expected, but the solution quality of the policy graph is approximately the same. This is an important observation, because it means that we can expect that the translation to policy graphs only introduces a minor solution quality change during the execution of CGCP. It is important to note that the coefficients that we insert in the master LP always remain exact, because for policy graphs we execute exact policy evaluation.

The second approach for assessing the quality difference, as described in Algorithm 14, defines the probability that the policy graph specifies an action to be executed, which would not be proposed by the vector-based policy. However, if the translation to a policy graph is near-exact (e.g., as in our case), then it boils down to enumeration of all beliefs reachable under policy execution. For the domains we tested we found that it is intractable to compute this metric in each iteration of value iteration. However, if our quality difference metric does not suffice (e.g., when the quality of the policy graph deviates a lot), then Algorithm 14 may be used in small domains for additional assessment of the policy quality change.

Domain	$\|S\|$	$\|A\|$	$\|O\|$
MiniHall	13	3	9
Cheese	11	4	7
4x3	11	4	6
Maze20	20	6	8
Hallway	60	5	21
Maintenance	5	12	5
Online advertising	4	3	5

Table 5.4: Size of the benchmark domains used in experiments

5.5.5. Additional details benchmark domains

For the robot navigation domains we took the corresponding POMDP domain description from `pomdp.org`. Note that the Maze20 domain corresponds to milos-aaai97. We made the following changes to the domains. We added a trap state which ensures that the robot transitions to the trap state after reaching the goal. We also added an action which enables the robot to be idle for one time step, and we created a cost function in which all non-idle actions have cost 1. Table 5.4 shows the size of the domains considered.

For the condition-based maintenance domain we use the bridge-repair domain provided on `pomdp.org`. In this domain the actions correspond to specific types of maintenance, followed by inspection. It is assumed that `clean-paint` actions have cost 10, and that `paint-strengthen` actions have cost 20. A `structural-repair` is even more expensive and has cost 30. The original domain is described in terms of cost rather than reward, and therefore we converted the reward structure. To be more specific, we want to ensure that high-cost actions in the original domain have low reward in our experiments, and hence we inverted the reward structure. For each (s,a)-pair we define the new reward as $(1+R_{\max}-R(s,a))\times 0.1$, where $R_{\max}$ is the maximum cost in the original instance, and the final 0.1 serves as a scalar. This way, the rewards in the modified instance are always strictly positive numbers.

For the online advertising problems we use the web-ad domain from `pomdp.org`. In this domain the actions correspond to advertising for specific types of products, or neutral advertising. We associated cost 1 with each action that advertises for a specific product type. It is assumed that neutral advertising has no cost.

In the multi-agent experiments we want to have agents with slightly different transition dynamics. In these experiments we first initialize all the agents based on the same POMDP model, after which we add noise to probabilities defining the state transition function. In particular, for each transition probability that is non-zero in the original problem, we add a number between 0 and 0.5, sampled uniformly at random. After that, we normalize the distributions such that the probabilities sum to 1. Note that this procedure ensures that the reachability of states is not affected.

Implementation detail: finite-horizon CALP

CALP (Poupart et al., 2015) computes an approximate Constrained POMDP solution using a constrained belief-state MDP, in which each state corresponds to a belief point of the original POMDP. The MDP is solved using an LP formulation for Constrained MDPs (Altman, 1999). The description of the algorithm assumes an infinite horizon with discounting, but the algorithm can also be used for finite-horizon problems if the following LP is used:

$$\begin{aligned}
\max\ & \sum_{t=1}^{h} \sum_{s\in S} \sum_{a\in A} x_{t,s,a} \cdot R(s,a) \\
\text{s.t.}\ & \sum_{a'\in A} x_{t+1,s',a'} = \sum_{s\in S} \sum_{a\in A} x_{t,s,a} \cdot P(s'|s,a) \quad && \forall s' \in S,\ \forall t \in \{1,\dots,h\} \\
& \sum_{a\in A} x_{1,s,a} = P(s_0 = s) && \forall s \in S \\
& \sum_{t=1}^{h} \sum_{s\in S} \sum_{a\in A} x_{t,s,a} \cdot C(s,a) \leq L.
\end{aligned} \tag{5.26}$$

The variable $x_{t,s,a}$ represents the probability that state s is reached at time t and action a is executed. In the second constraint s_0 refers to the initial state. Note that the LP is defined in terms of MDP states, whereas CALP uses the LPs for states representing a belief state of the POMDP. The LP solution represents a stochastic finite-state controller, which can be evaluated using a recurrence similar to the recurrence used in our work, and similar to the linear constraint system used by the original version of CALP.

5.6. Related work

Several algorithms have been proposed for solving Constrained POMDPs in the single-agent setting. It has been shown that the exact dynamic programming update for POMDPs can be generalized to Constrained POMDPs (Isom, Meyn, and Braatz, 2008). The vector pruning operation that is typically used in optimal POMDP algorithms can be executed by solving mixed-integer linear programs, whereas the traditional algorithms for POMDPs rely on a linear program (Cassandra, Littman, and Zhang, 1997; Walraven and Spaan, 2017). The method is computationally hard to execute on larger instances due to its exact nature. In order to address the computational difficulties, Constrained Point-Based Value Iteration (CPBVI) has been proposed (Kim et al., 2011). This algorithm executes point-based backups based on pairs consisting of a belief and the admissible cost that can be incurred when executing starting from that belief. Constrained Approximate Linear Programming (Poupart et al., 2015) has shown to outperform both the exact algorithm and CPBVI. It solves a belief-state MDP for an incrementally growing subset of belief points, based on the LP that is typically used in Constrained MDPs (Altman, 1999). Besides the actual solution and its expected value, it also computes an upper bound on the expected value, similar to our CGCP algorithm. None of the existing methods have discussed potential applications to multi-agent planning problems. Although it was not explicitly stated, this observation was made for the first time by Yost and Washburn (2000), which refer to multiple objects modeled as a POMDP. We generalized their approach and enhanced its scalability by using approximate algorithms. At the same time, our work also bridges the gap between CPBVI, CALP and the branch of work on column generation for Constrained POMDPs.

Planning for Constrained POMDPs is related to more general methods for resource allocation in stochastic environments. The aforementioned Constrained MDPs (Altman, 1999) provide the foundation for this work, which use the dual of an LP to bound the expected cost of the resulting solution. Even though the model is defined based on one individual agent, it can be generalized to multi-agent settings by concatenating the models of multiple agents into one LP. In domains where a limited amount of resources is available to the agents it is not always sufficient to bound the expected resource consumption, because this may lead to resource violations during execution time. Instead, mixed-integer linear programming can be used to compute static allocations of resources to agents prior to execution (Wu and Durfee, 2010), and its tractability can be increased by using a Lagrangian relaxation (Agrawal, Varakantham, and Yeoh, 2016). A drawback is that resources are allocated before execution, which leads to conservative resource consumption in uncertain domains because the allocation cannot be changed during execution. To address the limitations of preallocating resources, a conditional preallocation strategy has been proposed for Multi-Agent MDPs, which also employs column generation techniques (De Nijs et al., 2017). Our work extends this branch of work by considering partial observability, and we study how approximate solvers can be used for subproblems. In contrast, we do not consider strict resource limits, and our master LP only bounds the total expected cost.

In our work we use policy graphs as a representation for subproblem solutions. This is related to a larger branch of work on policy iteration techniques for optimizing finite-state controllers. Hansen (1998) discussed an approach to repeatedly evaluate and improve a finite-state controller. Later this approach was adopted by Poupart and Boutilier (2003) to create BPI, which aims to improve a finite-state controller using policy iteration while keeping its size fixed. Unfortunately, the algorithm can get trapped in a locally-optimal solution, and therefore a so-called escape technique has been proposed which gradually increases the size of the controller in order to improve it. More recent techniques employ local search (Braziunas and Boutilier, 2004), mixed-integer programming (Kumar and Zilberstein, 2015) and branch-and-bound techniques (Grześ, Poupart, and Hoey, 2013) to optimize controllers. Besides POMDPs, optimization of finite-state controllers has also been studied in the context of Decentralized POMDPs using non-linear programs (Amato, Bernstein, and Zilberstein, 2010). In our work we do not want to rely on such specific methods for optimizing finite-state controllers because some of these methods do not compute an upper bound on the expected value, which we need to use in the column generation algorithm. Additionally, some of these methods are computationally demanding, which is not practical if we want to compute subproblem solutions quickly in early iterations of the column generation algorithm.

There are relations between Decentralized POMDPs (Oliehoek and Amato, 2016) and the multi-agent model considered in this chapter. Our algorithm computes a solution for each agent which respects the constraint on cost, even if the agents do not communicate with each other during execution. This means that policy execution is fully decentralized. In our work the agents have an individual state space and their own set of actions, which means that the state transitions and the reward signal remain individual. In contrast, the Decentralized POMDP formalism defines the transitions and rewards based on joint states and joint actions, which allows one to model a larger class of decision making problems.

5.7. Conclusions

The Constrained POMDP framework provides a general formalism for sequential decision making under uncertainty subject to additional constraints. It extends the traditional Constrained MDP framework with partial observability, which becomes relevant in domains where an agent is unable to fully observe its environment. Existing research on Constrained POMDPs mainly focuses on adapting approximate algorithms for unconstrained POMDPs to constrained settings, or it aims to generalize existing work on Constrained MDPs to a partially observable setting. In this work we chose a rather different angle, in which optimization for Constrained POMDPs is seen as a global linear optimization problem defined over the entire policy space. A convenient property is that it enabled us to compute solutions by solving a series of unconstrained POMDPs using approximate algorithms.

We described a column generation technique for Constrained POMDPs, in which new policies are incrementally generated by solving subproblems. Since exact optimization can only be applied to tiny instances, we proposed a two-stage approach to solve subproblems using approximate methods. In particular, we use FiVI to compute an approximate subproblem solution, and we described a technique to translate the resulting solution into a policy graph. This representation is convenient because its quality can be evaluated using a simple recurrence. In a series of experiments we have demonstrated that the resulting CGCP algorithm outperforms the current state of the art in the field of Constrained POMDPs. Moreover, it is the only algorithm that provides immediate support for problems in which a global constraint is shared by multiple independent agents in a partially observable environment.

In future work it can be investigated whether and how the master LP of the column generation algorithm can be eliminated in multi-agent domains. Currently this central optimization problem does not form a bottleneck during the execution of the algorithm, but in larger multi-agent systems it can be convenient to replace this optimization problem by a distributed optimization scheme. We also observe that the current upper limit L only bounds cost in expectation. However, in several domains it can be desirable to have guarantees on the actual probability of a constraint violation. Existing techniques to achieve this in Constrained MDPs rely on solving many subproblems (De Nijs et al., 2017). In the partially observable setting it would be necessary to enhance these techniques because solving a large number of POMDPs can be computationally demanding. Furthermore, enforcing constraints on the actual cost incurred during policy execution, rather than the expected cost, remains an open challenge.

6
Constrained planning under uncertainty in smart grids

The power distribution grid serves as a backbone of our society. The current grid has been designed decades ago, and it faces major changes in the upcoming years due to new developments. For example, conventional power plants are being replaced by renewable alternatives, and there is an increasing number of electric vehicles (EVs). These developments introduce two major problems. First, power production of renewables brings more uncertainty in the distribution grid, which makes it more difficult to balance demand and supply. Second, distribution grids do not have sufficient capacity to accommodate the increase in the number of EVs, which leads to congestion. Both problems raise the question how uncertain demand and uncertain supply can be balanced in such a way that congestion is prevented. We propose an AI-based solution to congestion management problems, based on Constrained Multi-agent Markov Decision Processes (CMMDPs)[1]. In particular, we provide a novel mapping from power grid constraints to CMMDP constraints, which allows us to take realistic power grid constraints into account. Unfortunately CMMDPs only impose constraints on expectations, and therefore we present techniques to prevent violations of power grid constraints during execution. In a series of experiments we demonstrate the effectiveness of our approach in an IEEE distribution grid with uncertain production from renewables, uncertain consumption of EVs and a large number of households. Our results show that AI-based planning algorithms are able to manage congestion in smart distribution

[1]The research described in this chapter was performed in collaboration with Nils van der Blij, who was responsible for the electrical aspects of the proposed solutions and the simulations. The author of this dissertation was responsible for the algorithmic aspects and had the lead in writing the paper.

grids, and at the same time this shows that AI techniques are able to support the changes that distribution grids in our society are facing in the upcoming years.

6.1. Introduction

The power grid is the infrastructure that transports electric energy from producers to consumers in cities, neighborhoods and streets (Schavemaker and Van der Sluis, 2008). It currently consists of conventional generators which produce electric power, and lines that carry this power to consumers. The power grid can be seen as a hierarchical infrastructure, in which the transmission grid transports power over large distances, and the distribution grid delivers power to individual buildings and consumers. The grid serves as a main backbone of our society and economy, and it is a crucial infrastructure for daily needs such as communication, health care, transportation and our food supply chain.

There are two major developments which affect the operation of the grid in the upcoming years. First, conventional power generators are gradually being replaced by renewable generators at the distribution level (Weitemeyer et al., 2015). For example, solar panels and wind turbines provide power immediately to the distribution grid, whereas conventional generators only supply power through the transmission grid. Second, there is a significant increase in the number of electric vehicles expected, which requires charging using charging stations at the distribution level (Veldman and Verzijlbergh, 2015). Both developments contribute to the global energy transition, which aims to reduce the total emissions of the energy system (Breyer et al., 2017). As a result, it reduces the dependence of our society on energy sources such as oil, natural gas and coal.

Although renewable generators and electrification of transportation contribute to the transition to sustainable energy, it also introduces major problems related to the operation of the current grid. Renewable generators such as wind turbines and solar panels are uncontrollable and create more uncertainty regarding the amount of power produced, which makes it more difficult for optimization methods to reason about renewable production. This can be relevant when solving optimization problems for balancing demand and supply (Morales-España et al., 2016). Another problem arises due to the large number of electric vehicles in neighborhoods and streets. If a large number of vehicles in neighborhoods becomes electric then it is no longer feasible to charge them simultaneously due to the limited capacity of lines. It can be concluded that the transition to sustainable energy makes it more difficult to match demand and supply, and it introduces the risk that the grid becomes congested due to increased demand.

Both issues associated with the energy transition raise the question how the distribution grid, consumers and producers should change in the future. Reinforcing the current grid by increasing its capacity would be a straightforward solution to congestion problems, but this represents major maintenance work and it is considered expensive (Lunde, Røpke, and Heiskanen, 2016). Moreover, it should be noted that grid reinforcements do not address problems related to the uncertainty in renewable generation. An alternative solution to congestion problems uses the flexibility provided by consumers, and it aims to lower the total instantaneous demand by shifting demand to other periods of the day with renewable generation and excess grid capacity (Sundstrom and Binding, 2012; Walraven and Spaan, 2016). Rather than reinforcing the grid, it requires active and smart control of power consumption, and it can be seen as the transformation of the grid into a so-called smart grid which integrates intelligence in generators and appliances. The development of congestion management methods for smart grids is an area where artificial intelligence becomes important due to the ability to perform automated control and reasoning under uncertainty (Ramchurn et al., 2012). We take a major step in this direction by presenting the first AI-based framework for congestion management in distribution grids while considering uncertainty in, e.g., renewables and electric vehicle demand.

6.1.1. Contributions

In this chapter we present a novel approach to congestion management in distribution grids based on multi-agent planning under uncertainty. We introduce a framework based on Constrained Multi-agent Markov Decision Processes (CMMDPs) which can be used to control individual agents in a distribution grid while taking grid constraints and potential congestion into account. To be more specific, the contributions of the chapter are the following.

First, we provide a general overview of artificial intelligence literature focusing on applications in smart grids. The purpose of this overview is twofold. It allows us to identify general categories of smart grid applications in which artificial intelligence can play a key role. This includes congestion management in distribution grids, but also topics such as energy markets and the technical operation of the grid. Furthermore, our overview enables us to study whether realistic grid constraints have been considered in existing artificial intelligence methods for congestion management.

Second, we present a multi-agent planning framework suitable for congestion management, which is based on the Constrained Multi-agent Markov Decision Processes (Altman, 1999). The framework itself naturally supports constraints, and therefore we present a novel mapping of grid constraints to CMMDP constraints. This enables us to perform multi-agent planning under uncertainty while considering the characteristics of a distribution grid.

Third, we present techniques to ensure that solutions to CMMDPs respect grid constraints during execution. Traditional algorithms for CMMDPs ensure that the resulting policies respect the constraints only in expectation, which may lead to undesirable grid constraint violations during policy execution. In order to address this, we describe multiple methods to reduce or prevent such violations. These techniques turn out to be crucial, because they make it possible to use CMMDPs for congestion management.

Fourth, we present the results of a series of simulation experiments in which we demonstrate how our framework can be used for congestion management in a realistic IEEE distribution grid. In this study we consider uncertain power production provided by renewables, as well as control of flexible electric vehicles which require charging. Besides the computational results, the experiments confirm that artificial intelligence techniques can be used for congestion management and for building future power grids that serve our society.

6.1.2. Chapter outline

The structure of our chapter is as follows. In Section 6.2 we start with an overview of applications of artificial intelligence in the context of smart grids, which shows us that existing methods cannot be used for congestion management. In Section 6.3 we provide technical background information about power flows in distribution grids. These power flows are used in Section 6.4, where we present the CMMDP framework and the mapping of grid constraints to CMMDP constraints. Furthermore, we describe techniques to prevent constraint violations during policy execution. Section 6.5 presents the results of our simulation experiments. In Section 6.6 we discuss our conclusions and future work.

6.2. Applications of AI in smart grids

Artificial intelligence techniques can potentially contribute to the transformation of the passive energy grid into a smart energy grid. In this section we investigate to what extent artificial intelligence has been used for smart grids already, and we study whether existing algorithms are suitable for congestion management while dealing with uncertainty.

To be more specific, our literature study aims to identify whether existing work in the artificial intelligence community considers the capacity constraints imposed by the distribution grid. We consider all smart grid literature that appeared in major artificial intelligence conferences and journals. This specific focus narrows the scope of our literature study to a feasible size, but at the same time it enables us to see whether new techniques developed in the artificial intelligence community are suitable for congestion management. In this section we provide a high-level overview which highlights the main applications of artificial intelligence in smart grids. A full overview of the relevant papers is provided in Appendix 6A.

The structure of this section is based on four main categories of smart grid applications that we identified. In Section 6.2.1 we start with AI-based algorithms for control of flexible loads that are connected to the grid. In Section 6.2.2 we discuss applications in the context of energy markets, energy tariffs and trading. Grid control and integration of renewable generators using artificial intelligence is discussed in Section 6.2.3. We discuss AI-based methods for smart meters and forecasting in Section 6.2.4. Finally, in Section 6.2.5 we provide a discussion about grid constraints in optimization algorithms from a more general perspective, and we explain why artificial intelligence can be essential for solving constrained problems.

6.2.1. Control of flexible loads

The first category of applications involves loads that are connected to the grid, which naturally have power consumption that is temporally flexible. This means that the load needs to consume power in order to run, but the actual consumption can be shifted in time within a certain time window. The total power consumption and the size of the time window determine how much flexibility the load offers to the grid. As a concrete example we can consider charging of electric vehicles (De Weerdt et al., 2018). Such a vehicle consumes power in order to charge its battery, but when the vehicle actually charges is not relevant as long as the desired battery level is reached before departure. This means that the desired battery level and the intended departure time define the total flexibility that is available. Other examples of flexible loads include home appliances such as heating systems and refrigerators (Van Den Briel, Scott, and Thiébaux, 2013; De Nijs, Spaan, and de Weerdt, 2015).

Exploiting the flexible nature of loads in a distribution grid is relevant for multiple reasons. Most importantly, flexibility of loads can be used to prevent grid congestion. For example, if multiple electric vehicles charge simultaneously then the total consumption may exceed the capacity of the lines in the grid. The peak power consumption of the vehicles can be reduced by shifting the consumption from periods with high demand to periods with lower demand. Exploiting flexibility also becomes relevant when matching demand and renewable supply. For example, renewable generators are typically weather-dependent and do not provide power at all times. Flexible demand of loads can be controlled in such a way that power is consumed when renewable generators produce power, which means that power produced by renewables can be used locally in the distribution grid. This local usage of renewable power prevents disuptive power flows through the entire grid, and it reduces the usage of power produced by conventional generators.

Artificial intelligence methods have a huge potential for application in the context of controlling flexible loads because of the following key characteristics. Demand and supply in the grid is inherently uncertain due to the uncertain nature of the behavior of households as well as the uncertain power production of renewable generators. Furthermore, the grid can be seen as a large multi-agent system with multiple cooperative or self-interested households. The combination of uncertainty and multiple agents gives rise to the application of AI-based algorithms for multi-agent sequential decision making under uncertainty, which are a natural fit for control of flexible loads. Moreover, AI-based methods can model decision making problems with uncertainty that cannot be modeled by alternative methods such as multi-stage stochastic programming, because AI-based methods can be used to model settings where decisions influence the process representing the uncertainty (e.g., charging decisions which affect the uncertain demand the next day).

Based on the literature we found it can be concluded that AI-based algorithms for control of flexible loads received significant attention. A full overview of this literature is given in Appendix 6A. Unfortunately, it turns out that none of the existing work considers realistic grid constraints, which means that there is no suitable AI-based method that can be used to perform congestion management by controlling flexible loads. As a final remark we want to mention that new developments in the area of safe reinforcement learning are potentially relevant to prevent constraint violations during exploration, but to the best of our knowledge such methods have not been combined with realistic grids yet.

6.2.2. Energy markets, tariffs and trading

The second category of applications considers control of loads from a market perspective, in which markets and prices steer the behavior of appliances of users. To some extent this line of work can be seen as indirect control, because the combination of market mechanisms, pricing and user response achieves a certain behavior (e.g., shifting of loads). This is different compared to the methods discussed in the previous section, which control the loads directly.

Real-time pricing mechanisms have been proposed to provide incentives to consumers to shift their consumption from peak hours to off-peak hours (Bandyopadhyay et al., 2015). These incentives are provided by making the electricity price dependent on time. Most work considers ex-ante pricing mechanisms, which means that the prices are determined and announced before the relevant time window starts. This allows consumers to shift their flexible consumption to cheap periods of the day. The main challenge in this line of work is setting the actual prices, in such a way that they realize the desired effects. It should be noted that there can be an interplay with direct control of flexible loads as discussed in the previous section, because pricing mechanisms typically assume that flexible consumption is shifted when consumers respond to prices.

Due to the integration of renewables, households also start to produce energy besides the traditional consumption of energy. This gives rise to the participation in markets and trading of energy with other market participants (Ramchurn et al., 2012). Furthermore, multiple loads can be aggregated into groups, for which energy can be purchased in the market simultaneously. This coordinated purchasing based on larger groups can give additional advantages due to increased competition, prices that are more responsive to market conditions, and the ability to predict demand more reliably (Perrault and Boutilier, 2015).

Artificial intelligence can play a key role in this type of applications because setting suitable prices requires reasoning about uncertain behavior of consumers and producers. Moreover, algorithms for autonomous trading and market participation can be created by formulating planning problems in which actions define the properties of the bids performed by agents (Urieli and Stone, 2016). In Table 6.6 in Appendix 6A we provide a full overview of artificial intelligence literature related to markets, trading and pricing mechanisms. In this table we also indicate whether the work considers realistic grid constraints. For example, the marketplace proposed by Cerquides, Picard, and Rodríguez-Aguilar (2015) does refer to such grid constraints, but the actual modeling of the flows through the lines is not based on the actual characteristics of the lines. There is only one mechanism that considers realistic grid constraints (Burgess, Chapman, and Scott, 2018), but this mechanism cannot be used to perform congestion management using flexible loads in a distribution grid. From these observations we can conclude that artificial intelligence found a large number of applications in the context of markets, tariffs and trading, but none of these methods is suitable for congestion management while considering realisitc grid constraints.

6.2.3. Grid control and integration of renewables

The third category of applications focuses on the operation of the grid. In contrast to the flexible loads and markets discussed in Section 6.2.1 and 6.2.2, this category includes power generators and the actual components of the grid itself, rather than loads at the level of individual households. As a first example we consider power supply restoration problems in which it is required to reconfigure the grid after failures of lines. This means that one or more line switches need to be opened or closed in order to restore the power supply to the nodes in the grid (Agrawal, Kumar, and Varakantham, 2015). Another example is the dispatch of generators which supply the power that is required to make sure that the power demand always perfectly matches the power supply (Miller, Ramchurn, and Rogers, 2012). This requires solving a complex decision making problem in which it is decided when generators are running, and how much power they provide to the grid. For renewable generators such as solar panels and wind turbines the power production cannot be controlled explicitly, and therefore it is sometimes required to shut down (i.e., curtail) the generators to prevent excess power supply (Bandyopadhyay, Kumar, and Arya, 2016). The latter typically involves solving planning and scheduling problems.

The aforementioned problems all involve decision making under uncertainty, which makes it a natural application domain for techniques developed in the artificial intelligence community, for similar reasons as those discussed in Section 6.2.1. Table 6.4 in Appendix 6A provides an overview of the algorithms that appeared in the artificial intelligence literature. This table also shows us that a few algorithms consider realistic grid constraints during execution, which requires some additional explanations to indicate the differences with the work presented in this chapter. Coninx and Holvoet (2016) mention the technical constraints that need to be respected when wind power generation increases, but the work does not describe how such constraints are respected. Agrawal, Kumar, and Varakantham (2015) describe power flow conservation constraints in the context of power supply restoration, but the computed flows are not based on the actual characteristics of the lines. Piacentini et al. (2013) combine PDDL-based planning with an external power flow solver, which calculates the power flows given the line characteristics and the power demand and supply. Our work is fundamentally different because we provide an integrated solution which considers grid constraints while optimizing, whereas the PDDL-based planning method only performs multiple flow feasibility checks. Furthermore, it is important to mention that the PDDL-based planning method does not consider uncertainty, which is important in distribution grids.

6.2.4. Load forecasting and smart meter data

The final category of applications focuses on forecasting and measuring of power demand and supply. Forecasting becomes relevant if demand and supply of power need to be actively matched (Chen et al., 2013). In order to do this, algorithms need to reason about the uncertain demand and supply in the future, and therefore more accurate forecasts enable the algorithms to match demand and supply more effectively. Techniques such as Gaussian processes, neural networks and conditional random fields have been used to facilitate forecasting. Artificial intelligence techniques have also been used to measure power by means of data collection and analysis using smart meters. Most notably, energy disaggregation techniques extract load profiles of individual appliances from the aggregate load profile of a household (Parson et al., 2012), which can be useful to provide insight into consumption behavior and load shifting opportunities. A full overview of the relevant methods is provided in Table 6.5 in Appendix 6A. Typically, the presented methods only provide or analyze data, which means that there is no need for the algorithms to account for grid characteristics. It can be expected that new machine learning and data analysis methods developed in the artificial intelligence community can also be applied to this specific set of applications in the future.

6.2.5. Grid constraints in optimization algorithms

Integration of grid constraints in optimization methods has been studied in several different contexts, such as optimization in the operations research and power systems fields. In this section we describe these lines of work, and we highlight the benefits of AI-based algorithms. We only describe key techniques and applications that have been considered, and we do not aim to give a complete overview of all the work because it is beyond the scope of this dissertation.

From an operations research point of view there has been work on integration of grid constraints and enhancing the efficiency when solving optimization problems in power systems that include such constraints. For example, Coffrin and Van Hentenryck (2014) present techniques to linearize and approximate power grid constraints in such a way that the models also capture reactive power and voltage magnitudes. These properties can also be potentially integrated in our work, but since we are interested in congestion management it is sufficient to consider constraints on flows. Furthermore, there has been work on SDP relaxations (Hijazi, Coffrin, and Van Hentenryck, 2016) and QC relaxations (Coffrin, Hijazi, and Van Hentenryck, 2016) in the context of optimal power flow problems, in which it is determined how conventional power generation units should be operated in order to meet the total demand while minimizing the total operating cost. There are three key properties which motivate why artificial intelligence based methods can be beneficial compared to the aforementioned techniques. First, AI techniques have shown to be able to model hard constraints, as well as constraints for which minor violations are allowed. The latter can be beneficial in distribution grids because typically minor violations are allowed. We further elaborate on this in Section 6.4.6. Second, AI-based algorithms have been used to create systems in which agents operate in a decentralized way, which is relevant to reduce the amount of communication that is required in a smart grid. Third, multiple types of uncertainty can be integrated in AI-based planning algorithms, whereas traditional optimization algorithms typically support only one type of uncertainty, which we further explain below.

Optimization algorithms which include both grid constraints and uncertainty have been considered in the context of the unit commitment, which is a critical task in the operation of power systems. Unit commitment involves scheduling the dispatch of power generation units in order to satisfy the demand (Padhy, 2004). If the unit commitment problem includes the full characteristics of the grid, then it actually contains the aforementioned optimal power flow problem. Typically, uncertainty is included by formulating the problem as a stochastic or robust optimization problem, in which multiple scenarios characterize the uncertainty that can be encountered over time (Morales-España, Lorca, and de Weerdt, 2018). This type of uncertainty characterization is suitable for exogenous sources of uncertainty such as wind and solar power, but it is important to note that it does not allow for modeling of problems in which decisions influence uncertainty encountered later in time. For example, one can think about electric vehicle charging problems in which charging decisions influence the demand uncertainty encountered during the next day. AI planning algorithms based on Markov Decision Processes, on the other hand, can be used to model such problems, which means that these algorithms can be applied to a broader class of decision making problems with uncertainty in smart grids (Defourny, Ernst, and Wehenkel, 2012).

6.2.6. Summary

We identified four categories of smart grid applications and we discussed how artificial intelligence can play an essential role in these applications. We found that intelligent decision making algorithms developed in the AI community can be used for control of flexible loads, market participation, pricing, grid control and analyzing user demand and supply. In these applications it is crucial to perform automated reasoning for a large number of consumers and producers, while taking several sources of uncertainty into account. These characteristics make smart grids an attractive application domain for artificial intelligence methods. This is a promising conclusion, but unfortunately we found that none of the existing AI-based methods is currently suitable to perform congestion management in these applications because realistic power flows in lines are typically not considered. In this chapter we close this knowledge gap by presenting algorithms for decision making under uncertainty which consider realistic grid constraints. Furthermore, our algorithms naturally support decentralized decision making, exogenous uncertainty and endogenous uncertainty, which is typically not supported simultaneously by traditional optimization techniques.

6.3. Power flows in distribution grids

In this section we present background information about modeling of smart grids and we introduce a mathematical model to calculate power flows. The model is widely used in industry and academic literature, mainly because of the linearity, acceptable accuracy and the low computational burden compared to an exact non-linear model (Stott, Jardim, and Alsac, 2009). The model approximates the power flow by assuming that the resistance in the network is negligible, and the voltage and voltage angle differences between nodes are small, which is valid for distribution systems. This model is suitable for analyzing power flows, which makes it a suitable model for congestion management methods. In this section we provide a description of the computation of the power flows for an Alternating Current (AC) distribution grid. It turns out that Direct Current (DC) distribution grids can be modeled in a similar fashion, and this eventually leads to the same equations, which means that our congestion management algorithms are sufficiently generic to be used in both types of grids. More details about modeling of DC distribution grids are provided in the manuscript corresponding to this chapter.

We model a distribution grid, to which we simply refer as grid, which is the infrastructure to which power-consuming and power-producing entities such as households are connected. In such a grid the consumption of power leads to flow of power through the lines. The flow through a line should not exceed the total capacity of the line (Schavemaker and Van der Sluis, 2008). More formally, the grid can be modeled as an undirected graph consisting of m nodes, indexed from 1 to m. A line $(\rho,\tau) \in \mathcal{L}$ connects node ρ to node τ, and the capacity limit, resistance and reactance of this line are denoted by $\ell_{\rho,\tau}$, $r_{\rho,\tau}$ and $x_{\rho,\tau}$, respectively. The capacity limit, resistance and reactance are given properties of the line. The amount of power injected in node ν is denoted by $\mathcal{P}(\nu)$, which becomes a negative number in case power is consumed from the grid. Without loss of generality it is assumed that the grid is connected to a transmission grid through node 1, which supplies power to the distribution grid. At all times it is required that

$$-1 \cdot \mathcal{P}(1) = \sum_{\nu=2}^{m} \mathcal{P}(\nu), \tag{6.1}$$

which means that power supply and demand are always balanced.

It is not sufficient to ensure that power demand and supply are equal. It is also important that the power flow in each line does not exceed the capacity limit of the line. Power injected in node 1 and consumed in node ν flows through one or more paths through the grid, but this is not necessarily the shortest path. The actual path depends on the grid topology and the properties of the lines.

In Section 6.6 it is derived that the power flow $\mathcal{P}_{\rho,\tau}$ through the line from node ρ to node τ is equal to:

$$\mathcal{P}_{\rho,\tau} = \frac{\overline{U}^2}{x_{\rho,\tau}}(\theta_\rho - \theta_\tau), \tag{6.2}$$

where θ_ρ and θ_τ denote the voltage angles in node ρ and τ, respectively, and $\overline{U}$ denotes the nominal voltage. The voltage angle is a property of the nodes relating voltage to current. Based on the equation we can see that the problem of computing flows reduces to computing voltage angles. For each node ν it should hold that injected power in node ν equals the amount of power flowing to other nodes:

$$\mathcal{P}(\nu) = \sum_{j=1,\ j\neq\nu,\ (\nu,j)\in\mathcal{L}}^{m} \mathcal{P}_{\nu,j} = \sum_{j=1,\ j\neq\nu,\ (\nu,j)\in\mathcal{L}}^{m} \frac{\overline{U}^2}{x_{\nu,j}}(\theta_\nu - \theta_j), \tag{6.3}$$

where the condition under the summation ensures that pairs of nodes without a line are excluded. This set of equations can be written in matrix form as follows:

$$\begin{bmatrix} \mathcal{P}(1) \\ \vdots \\ \mathcal{P}(m) \end{bmatrix} = \overline{U}^2 \mathbf{M} \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_m \end{bmatrix}, \tag{6.4}$$

where $\mathbf{M}$ is an $m \times m$ matrix. The element $\mathbf{M}_{k,l}$ at row k and column l is defined by:

$$\mathbf{M}_{k,l} = \begin{cases} -1/x_{k,l} & \text{if } k \neq l \\ \sum_{j=1,\ j \neq k,\ (k,j) \in \mathcal{L}}^{m} 1/x_{k,j} & \text{otherwise} \end{cases}. \tag{6.5}$$

We can obtain the voltage angles by computing

$$\begin{bmatrix} \theta_1 \\ \vdots \\ \theta_m \end{bmatrix} = \frac{1}{\overline{U}^2} \mathbf{M}' \begin{bmatrix} \mathcal{P}(1) \\ \vdots \\ \mathcal{P}(m) \end{bmatrix}, \tag{6.6}$$

where $\mathbf{M}'$ is the inverse of $\mathbf{M}$. Since the matrix $\mathbf{M}$ is singular, the inverse does not exist. Therefore, we set θ_1 equal to 0 and solve the following set of linear equations:

$$\begin{bmatrix} \theta_2 \\ \vdots \\ \theta_m \end{bmatrix} = \frac{1}{\overline{U}^2} \mathbf{Z} \begin{bmatrix} \mathcal{P}(2) \\ \vdots \\ \mathcal{P}(m) \end{bmatrix}, \tag{6.7}$$

where $\mathbf{Z}$ is an $(m-1) \times (m-1)$ matrix obtained by removing the first row and the first column of $\mathbf{M}$, and then the inverse of the resulting matrix is computed. The power flow in each line can be computed using the voltage angles and Equation 6.2.

Computations can be further simplified by observing that the voltage angles can be written as follows:

$$\theta_k = \frac{1}{\overline{U}^2} \sum_{j=2}^{m} \mathbf{Z}_{k-1,j-1} \cdot \mathcal{P}(j) \quad \forall k \in \{2, \dots, m\}. \tag{6.8}$$

Note that we subtract 1 in the indices of $\mathbf{Z}$ because the voltage angle for node k is computed using row $k-1$ of $\mathbf{Z}$ due to the matrix size reduction. Substituting this expression in Equation 6.2 yields the following if $\rho > 1$ and $\tau > 1$:

$$\mathcal{P}_{\rho,\tau} = \frac{\overline{U}^2}{x_{\rho,\tau}} (\theta_\rho - \theta_\tau) \tag{6.9}$$

$$= \frac{\sum_{j=2}^{m} \mathbf{Z}_{\rho-1,j-1} \mathcal{P}(j) - \sum_{j=2}^{m} \mathbf{Z}_{\tau-1,j-1} \mathcal{P}(j)}{x_{\rho,\tau}} \tag{6.10}$$

$$= \sum_{j=2}^{m} \mathcal{P}(j) \cdot \frac{1}{x_{\rho,\tau}} \cdot (\mathbf{Z}_{\rho-1,j-1} - \mathbf{Z}_{\tau-1,j-1}). \tag{6.11}$$

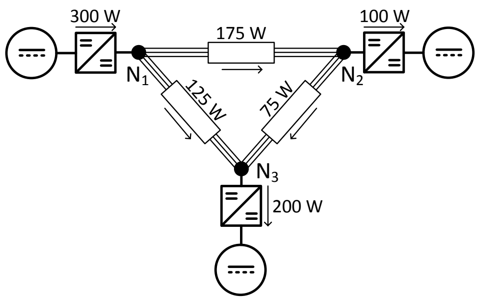


Figure 6.1: Example distribution grid with computed power flows

As we can see, it turns out that the flow between node ρ and node τ can be calculated as a linear function of the power $\mathcal{P}(j)$ injected in the nodes, weighted by a term that is not dependent on the injected power. These terms can be precomputed, and thus we can compute the flow $\mathcal{P}_{\rho,\tau}$ as follows:

$$\mathcal{P}_{\rho,\tau} = \sum_{j=2}^{m} \mathcal{P}(j) \cdot S_{\rho,\tau,j}, \tag{6.12}$$

with

$$S_{\rho,\tau,j} = \begin{cases} \frac{\mathbf{Z}_{\rho-1,j-1} - \mathbf{Z}_{\tau-1,j-1}}{x_{\rho,\tau}} & \rho,\tau > 1, \rho \neq \tau \\ -\mathbf{Z}_{\tau-1,j-1} \,/\, x_{\rho,\tau} & \rho = 1, \tau > 1 \\ \mathbf{Z}_{\rho-1,j-1} \,/\, x_{\rho,\tau} & \rho > 1, \tau = 1 \\ 0 & \text{otherwise} \end{cases}. \tag{6.13}$$

The first case follows immediately from the derivation, and the second and the third case follow from a similar derivation, where 0 is substituted for either θ_ρ or θ_τ. The number $S_{\rho,\tau,j}$ is a Power Transfer Distribution Factor (PTDF), representing the fraction of power from node 1 to node j that flows through the line from node ρ to node τ. The PTDFs are convenient because they can be precomputed, and flows can be easily computed using the injected power $\mathcal{P}(2), \ldots, \mathcal{P}(m)$. Congestion arises if the flow in the line from node ρ to node τ exceeds the line capacity $\ell_{\rho,\tau}$.

As a numerical example we consider a grid consisting of 3 nodes, as shown in Figure 6.1, in which the arrows indicate the direction of flow. The power consumption in node 2 is 100 W, and the power consumption in node 3 is 200 W. This means that $\mathcal{P}(1) = 300$, $\mathcal{P}(2) = -100$ and $\mathcal{P}(3) = -200$. For the reactance we assume that $x_{1,2} = 0.1$, $x_{1,3} = 0.2$ and $x_{2,3} = 0.1$. Intuitively, one may think that $\mathcal{P}_{1,2} = 100$, $\mathcal{P}_{1,3} = 200$ and $\mathcal{P}_{2,3} = 0$, but this is not the case. The matrix $\mathbf{M}$ and the corresponding matrix **Z** are as follows:

$$\mathbf{M} = \begin{bmatrix} 15 & -10 & -5 \\ -10 & 20 & -10 \\ -5 & -10 & 15 \end{bmatrix} \quad \mathbf{Z} = \begin{bmatrix} 0.075 & 0.05 \\ 0.05 & 0.1 \end{bmatrix}.$$

We show how $\mathcal{P}_{2,3}$ is computed. It requires $S_{2,3,2} = (\mathbf{Z}_{1,1} - \mathbf{Z}_{2,1})\ /\ x_{2,3} = (0.075 - 0.05)\ /\ 0.1 = 0.25$ and $S_{2,3,3} = (\mathbf{Z}_{1,2} - \mathbf{Z}_{2,2})\ /\ x_{2,3} = (0.05 - 0.1)\ /\ 0.1 = -0.5$. Now we derive $\mathcal{P}_{2,3} = (\mathcal{P}(2) \cdot S_{2,3,2}) + (\mathcal{P}(3) \cdot S_{2,3,3}) = (-100 \cdot 0.25) + (-200 \cdot -0.5) = 75$. The flows $\mathcal{P}_{1,2}$ and $\mathcal{P}_{1,3}$ can be computed similarly.

6.4. Congestion management using CMMDPs

In this section we present a framework for congestion management using constrained multi-agent planning under uncertainty. We start with a general motivation that explains why planning under uncertainty is a suitable approach for congestion management in distribution grids. After that, we provide a general description of Constrained Multi-agent Markov Decision Processes (CMMDPs) and the corresponding solution methods. In order to make sure that CMMDP solutions take grid constraints into account, we present a mapping of grid constraints to CMMDP constraints. Finally, we describe additional techniques to ensure that these grid constraints are respected during the execution of the computed policies.

6.4.1. Algorithm requirements for congestion management

Congestion management in distribution grids can be seen as control of power consumption and generation, in such a way that the capacities of the lines are not violated. In Section 6.2 we have seen that flexibility of loads provides an attractive concept to prevent congestion, because such loads can be used to shift consumption to other periods of the day in which sufficient grid capacity is still available. In other words, power consumption can be spread out over the day, rather than consuming power simultaneously.

Deciding how power consumption can be shifted requires decision making algorithms that are able to deal with multiple agents. Besides decision making it is important that algorithms are able to deal with uncertainty. For example, uncontrollable consumption of households and uncontrollable production of renewables is inherently uncertain and this needs to be considered when making decisions about shifting load. Another important characteristic of flexible loads in a distribution grid is that they naturally want to achieve a certain goal. For example, heating systems in buildings aim to keep the temperature close to a given setpoint, whereas electric vehicles want to eventually reach a fully-charged battery. This gives rise to the application of decision making algorithms that maximize a notion of reward. In heating systems this reward can be related to the deviation from the setpoint, and electric vehicles get rewarded when they reach their desired battery level.

The line of reasoning above shows us that congestion management in distribution grids requires algorithms for multi-agent planning under uncertainty which maximize the notion of reward while respecting the constraints imposed by the lines of the grid. Constrained Multi-agent Markov Decision Processes include all basic requirements that we need, and therefore we select this model to formalize congestion management problems. Furthermore, it turns out that the model can be used for decentralized decision making, which is convenient in smart distribution grids because it reduces communication requirements.

6.4.2. Constrained Multi-agent Markov Decision Processes

We consider a Constrained Multi-agent Markov Decision Process that consists of n agents. We introduced Constrained Markov Decision Processes in Section 2.1, but in this section we provide an additional introduction because we use a slightly different model that includes both constraints and multiple agents. Each agent i is modeled as a finite-horizon Markov Decision Process (Puterman, 1994) and there are resource constraints which force the agents to coordinate. For agent i the MDP is defined by the tuple $M_i = (S_i, A_i, T_i, R_i, s_{i,1}, h)$, where S_i denotes a finite set of states, and the finite set A_i contains actions. The planning horizon is denoted by h, which is identical for each agent, and decisions are made at the timesteps $1, \dots, h$. The function $T_i : S_i \times A_i \times S_i \times \{1, \dots, h\} \rightarrow [0,1]$ defines stochastic state transitions, where $T_i(s, a, s', t)$ represents the probability that the state changes from $s \in S_i$ to $s' \in S_i$ after executing action $a \in A_i$ at time t. The function $R_i : S_i \times A_i \times \{1, \dots, h\} \rightarrow \mathbb{R}$ encodes the reward $R_i(s, a, t)$ that is received when executing action $a \in A_i$ in state $s \in S_i$ at time t. The initial state is denoted by $s_{i,1}$. A solution is represented by a policy $\pi_i : \{1, \dots, h\} \times S_i \rightarrow A_i$ for each agent i, such that $\pi_i(t, s)$ defines the action $a \in A_i$ to be executed at time t if the state is $s \in S_i$. The objective is to maximize the total expected reward received by the agents:

$$\sum_{i=1}^{n} \left(E_{\pi_i} \left[\sum_{t=1}^{h} R_i(s_{i,t}, \pi_i(t, s_{i,t}), t) \,\middle|\, s_{i,1} \right] \right), \tag{6.14}$$

in which $s_{i,t}$ denotes the state of agent i at time t.

Constraints on resources, indexed by z, require coordination of the decisions made by the agents. For each agent i the consumption of resource z is defined by $C_{i,z} : S_i \times A_i \rightarrow \mathbb{R}$, such that $C_{i,z}(s, a)$ denotes the instantaneous consumption of resource z when executing action $a \in A_i$ in state $s \in S_i$. The resource limit is defined by L_z, which is violated at time t if $\sum_{i=1}^{n} C_{i,z}(s_{i,t}, a_{i,t}) > L_z$, where $s_{i,t}$ and $a_{i,t}$ denote the state and executed action of agent i at time t, respectively.

The multi-agent planning problem outlined above can be solved using linear programming in case it is sufficient to satisfy the constraints in expectation. For this purpose the Constrained MDP framework (Altman, 1999) defines a linear program, which imposes constraints on the policies of the agents, as shown below.

$$\begin{aligned}
\max \sum_{i=1}^{n}\sum_{t=1}^{h}\sum_{s\in S_i}\sum_{a\in A_i} & x^i_{t,s,a}\cdot R_i(s,a,t) && \\
\text{s.t.} \sum_{a'\in A_i} x^i_{t+1,s',a'} &= \sum_{s\in S_i}\sum_{a\in A_i} x^i_{t,s,a}\cdot T_i(s,a,s',t) && \forall i,t,s'\in S_i \\
\sum_{a\in A_i} x^i_{1,s,a} &= P(s_{i,1}=s) && \forall i,s\in S_i \\
\sum_{i=1}^{n}\sum_{s\in S_i}\sum_{a\in A_i} x^i_{t,s,a}\cdot C_{i,z}(s,a) &\leq L_z && \forall z,t \\
0 \leq x^i_{t,s,a} &\leq 1 && \forall i,t,s,a
\end{aligned} \tag{6.15}$$

The decision variable $x^i_{t,s,a}$ denotes the probability that agent i reaches state s at time t, and subsequently executes action a. Agent i selects action a in state s at time t with probability $x^i_{t,s,a} \,/\, \sum_{a'\in A_i} x^i_{t,s,a'}$. The policies of the agents may be stochastic, and they have a very attractive property in the multi-agent setting. The policy of agent i does not depend on the policies of other agents and therefore it can be executed in a decentralized way. This means that the agents can execute the policies without any communication while making sure that the constraints are not violated in expectation.

6.4.3. Agent goals and guarantees

As discussed in Section 6.4.1, flexible loads in a distribution grid aim to reach a certain goal, such as a fully-charged battery. The Constrained Multi-agent Markov Decision Process framework allows us to formalize this notion of goals using the reward functions of the agents. For each agent i with a specific goal, we define a reward function R_i which gives reward 1 when reaching the goal of the agent, and it defines reward 0 otherwise. This type of reward function is convenient, because the expected reward

$$\mathcal{E}_i = \sum_{t=1}^{h}\sum_{s\in S_i}\sum_{a\in A_i} x^i_{t,s,a}\cdot R_i(s,a,t) \tag{6.16}$$

becomes equivalent to the probability that agent i reaches its goal. Furthermore, maximizing expected reward based on such a reward function corresponds to maximizing the probability that the agent reaches its goal. Based on the expected reward of the agents we can define the total expected reward of the agents with a goal as follows:

$$\mathcal{E} = \sum_{\{i=1,\dots,n \mid \text{agent } i \text{ has a goal}\}} \mathcal{E}_i. \tag{6.17}$$

Given a set of policies computed using the linear program in Equation 6.15, it is important to know whether the agents will reach their goal during the execution of their policy. The expectation $\mathcal{E}$ allows us to check whether it is guaranteed that all agents reach their goal, as formalized in the theorem below.

Theorem 9. *If $\mathcal{E}$ is equal to the total number of agents with a goal, then it is guaranteed that all the agents reach their goal during policy execution.*

Proof. We assume that $\mathcal{E}$ equals the number of agents with a goal, and we prove by contradiction that it implies that all the agents reach their goal. Suppose the contrary, namely that it is not guaranteed that all the agents reach their goal. In that case there is at least one agent i for which the probability to reach the goal is less than 1, which means that $\mathcal{E}_i < 1$. Since there is at least one $\mathcal{E}_i$ in the sum in Equation 6.17 which is less than 1, it follows that $\mathcal{E}$ is less than the number of agents with a goal. This leads to a contradiction, because we assumed that $\mathcal{E}$ equals the number of agents with a goal. We conclude that all the agents reach their goal if $\mathcal{E}$ equals the number of agents with a goal. □

Our definitions and analysis regarding agent goals become relevant when integrating grid constraints because of two reasons. Most importantly, we can use it to assess whether the agents are guaranteed to reach their goal while respecting grid constraints. Furthermore, our techniques to enforce grid constraints make use of the expectation $\mathcal{E}$ to decide whether optimization can be terminated or not.

6.4.4. Integrating grid constraints in Constrained MMDPs

The CMMDP model provides a general framework for planning under uncertainty subject to constraints. In this section we establish a connection between this model and constraints on power flows in distribution grids. This enables us to solve CMMDPs in such a way that the resulting policies ensure that the agents can reach their goals while respecting the constraints imposed by the lines of the power grid.

The resource consumption functions discussed in Section 6.4.2 can be used to model constraints. To make sure that CMMDP constraints correspond to grid constraints, we derive a relationship between the flows and the resource consumption functions. It turns out that the linearity of the flows in Section 6.3 gives us a convenient mapping from a grid constraint to a collection of CMMDP constraints.

The agents are connected to nodes in the grid, and the lines of the grid impose constraints on the behavior of the agents. Agent i is connected to node $2 \leq g(i) \leq m$, in which it can either inject or subtract power. We let $c_i(s,a)$ denote the power consumption of agent i when executing action a in state s, which corresponds to the amount of power subtracted from the grid. Note that agents cannot be connected to node 1 because it represents the connection to the transmission grid, as discussed in Section 6.3.

Considering a line $(\rho,\tau) \in \mathcal{L}$, we can see in Equation 6.12 that each agent contributes a certain amount of flow to the line, regardless of the location of the agent in the grid. For agent i the flow contributed to line $(\rho,\tau) \in \mathcal{L}$ is equal to $S_{\rho,\tau,g(i)} \cdot \mathcal{P}(g(i))$, in which the last term represents the amount of power agent i injects in the node $g(i)$ to which it is connected. For example, when executing action a in state s, then agent i would contribute flow $-1 \cdot c_i(s,a) \cdot S_{\rho,\tau,g(i)}$ to the total flow through the line. Note that we multiply by -1 because $c_i(s,a)$ represents the power subtracted from the grid while $\mathcal{P}(g(i))$ represents the power injected into the grid.

The grid constraints can be integrated in the MDP formulation of the agents by adding resource consumption functions and corresponding constraints. For each line $(\rho,\tau) \in \mathcal{L}$ we create two resources $z_{\rho,\tau}$ and $z_{\tau,\rho}$. Both resources correspond to the same line, but they represent two directions of flow. We need two resources to model a constraint defining that the flow in the line is upper bounded by $\ell_{\rho,\tau}$, because we do not know beforehand whether the power flows from node ρ to τ, or from τ to ρ. For resource $z_{\rho,\tau}$ the consumption function associated with agent i is defined as follows:

$$C_{i,z_{\rho,\tau}}(s,a) = -1 \cdot c_i(s,a) \cdot S_{\rho,\tau,g(i)} \qquad \forall s,a. \tag{6.18}$$

The final constraints can be imposed by adding the following constraints to the linear program shown in Equation 6.15:

$$\sum_{i=1}^{n} \sum_{s \in S_i} \sum_{a \in A_i} x^i_{t,s,a} \cdot C_{i,z_{\rho,\tau}}(s,a) \leq \ell_{\rho,\tau} \quad \forall (\rho,\tau) \in \mathcal{L},\ t. \tag{6.19}$$

The consumption function and constraints for resource $z_{\tau,\rho}$ are identical, except that ρ and τ are reversed. The constraints merge the individual consumption functions of the agents, and they ensure that the *expected* power flow is upper bounded by $\ell_{\rho,\tau}$. The current formulation ensures that the constraints of either $z_{\rho,\tau}$ or $z_{\tau,\rho}$ are violated in case the expected power flow in the line $(\rho,\tau) \in \mathcal{L}$ exceeds $\ell_{\rho,\tau}$. The actual constraint that is violated depends on the direction of the power flow. Constraints cannot be simplified into one constraint by taking the absolute value of the consumption (e.g., to eliminate direction), because it leads to incorrect expectations if positive and negative terms cancel out.

The mapping presented in this section can be used to solve CMMDPs in such a way that the resulting stochastic policies satisfy the grid constraints in expectation. However, in one single execution run constraints may be violated, which is not desirable in a realistic distribution grid. In the next two sections we present multiple strategies to reduce or prevent constraint violations during policy execution.

6.4.5. Preventing violations of constraints: preallocation

The resource consumption functions and the LP constraints defined in the previous section ensure that the *expected* power flow of the lines does not exceed the total capacity of the lines. However, given a computed solution it is unclear to what extent constraints are violated during execution, even if the expected power flows are bounded. In this section we present a preallocation technique to prevent constraint violations completely. Since enforcing hard constraints is not always necessary, we present additional techniques in the next section which can be used in case minor violations are allowed.

The LP solution defines variables $x^i_{t,s,a}$ for agent i, which correspond to the probability that agent i reaches state s at time t and subsequently chooses action a. Intuitively, agent i *may* execute action a in state s at time t if $x^i_{t,s,a} > 0$, but because of this uncertainty we cannot use variables $x^i_{t,s,a}$ directly to create constraints related to worst-case scenarios. We propose to use a strategy which preallocates grid capacity to the agents while solving the planning problem, similar to Wu and Durfee (2010) and De Nijs, Spaan, and de Weerdt (2018). We first introduce binary variables $\bar{x}^i_{t,s,a} \in \{0,1\}$, which indicate that agent i is allowed to execute action a in state s at time t. These variables can be used in additional constraints:

$$x^i_{t,s,a} \leq \bar{x}^i_{t,s,a} \quad \forall i,t,s,a, \tag{6.20}$$

which enforce that agents are only able to execute actions in case this is explicitly allowed.

The binary variables can be used in new grid constraints which replace the constraints shown in Equation 6.19. Considering a specific line $(\rho,\tau) \in \mathcal{L}$ and its resource $z_{\rho,\tau}$, the worst-case flow increase when $\bar{x}^i_{t,s,a}$ turns from 0 to 1 is equal to:

$$\bar{C}_{i,z_{\rho,\tau}}(s,a) = \max\{C_{i,z_{\rho,\tau}}(s,a), 0\}. \tag{6.21}$$

Taking the maximum is necessary because $C_{i,z_{\rho,\tau}}(s,a)$ may be negative, which would be a flow decrease rather than an increase. For resource $z_{\tau,\rho}$, corresponding to the reverse direction, the worst-case flow increase can be defined similarly by swapping ρ and τ. The hard grid constraints now become:

$$\sum_{i=1}^{n} \sum_{s\in S_i} \sum_{a\in A_i} \bar{x}^i_{t,s,a} \cdot \bar{C}_{i,z_{\rho,\tau}}(s,a) \leq \ell_{\rho,\tau} \quad \forall(\rho,\tau) \in \mathcal{L},\ t. \tag{6.22}$$

As before, the constraints for $z_{\tau,\rho}$ are identical with ρ and τ reversed. When using the constraints together with the constraints defined by Equation 6.20, the linear program becomes a mixed-integer linear program (MILP) which preallocates grid capacity to the agents. The agents use the resulting stochastic policy induced by the variables $x^i_{t,s,a}$ during execution without communicating with each other. Most importantly, constraints are never violated during execution, which we formally prove below.

Theorem 10. *The constraints defined by Equations 6.20-6.22 enforce that the agents do not violate grid constraints while executing the stochastic policy induced by the variables $x^i_{t,s,a}$.*

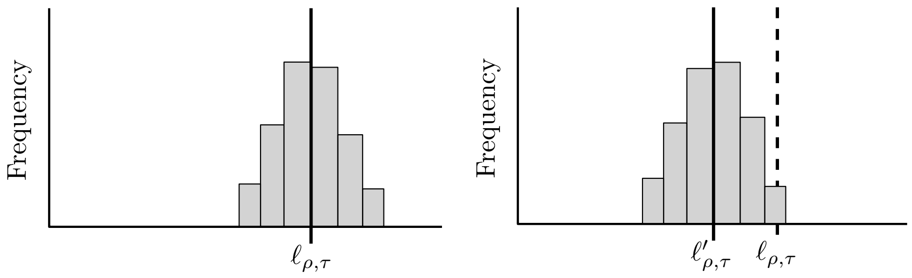
Proof. Without loss of generality we consider timestep t and a line $(\rho,\tau) \in \mathcal{L}$ with its resource $z_{\rho,\tau}$ and limit $\ell_{\rho,\tau}$. Assuming that constraints defined by the Equations 6.20-6.22 hold, we show that $\mathcal{P}_{\rho,\tau} \leq \ell_{\rho,\tau}$:

$$\begin{aligned}
\mathcal{P}_{\rho,\tau} &= \sum_{j=2}^{m} \mathcal{P}(j) \cdot S_{\rho,\tau,j} && (6.23)\\
&= \sum_{i=1}^{n} \mathcal{P}(g(i)) \cdot S_{\rho,\tau,g(i)} && (6.24)\\
&\leq \sum_{i=1}^{n} \sum_{\{(s,a) \mid x^i_{t,s,a} > 0\}} -c_i(s,a) \cdot S_{\rho,\tau,g(i)} && (6.25)\\
&\leq \sum_{i=1}^{n} \sum_{\{(s,a) \mid x^i_{t,s,a} > 0\}} \max\{-c_i(s,a) \cdot S_{\rho,\tau,g(i)}, 0\} && (6.26)\\
&\leq \sum_{i=1}^{n} \sum_{s \in S_i} \sum_{a \in A_i} \bar{x}^i_{t,s,a} \cdot \max\{-c_i(s,a) \cdot S_{\rho,\tau,g(i)}, 0\} && (6.27)\\
&= \sum_{i=1}^{n} \sum_{s \in S_i} \sum_{a \in A_i} \bar{x}^i_{t,s,a} \cdot \bar{C}_{i,z_{\rho,\tau}}(s,a) && (6.28)\\
&\leq \ell_{\rho,\tau}. && (6.29)
\end{aligned}$$

In the derivation step 6.24 holds because we can take the sum over the agents rather than nodes. Step 6.25 holds because we take the sum over all realizations of pairs (s,a) that may arise, even though only one of them will occur. In step 6.26 the max operator only discards negative numbers from the sum, and hence this sum cannot decrease. Step 6.27 follows from the constraints defined by Equation 6.20. Step 6.28 and step 6.29 follow from Equation 6.21 and Equation 6.22, respectively. Proving that $\mathcal{P}_{\tau,\rho} \leq \ell_{\rho,\tau}$ works similarly. □

6.4.6. Reducing violations using empirical bounding

The techniques introduced in the previous section ensure that power grid constraints are respected at all times, considering all possible realizations of the stochas-



(a) Planning with actual limit $\ell_{\rho,\tau}$ (b) Planning with reduced limit $\ell'_{\rho,\tau}$

Figure 6.2: Reducing the number of violations by reducing the limits used for planning.

tic behavior of the agents. Imposing such a hard constraint on the violation of a line limit means that always the absolute worst case is considered. However, due to the large uncertainty and the large number of contributors in the grid, it is unlikely that the worst case scenario occurs. Nowadays, the unlikely coincidence of (household) load peaks is taken into account using a diversity factor for sizing components of the distribution grid[2]. For example, according to the French NFC14-100 standard the diversity factor is between 1 and 0.4 depending on the number of households considered. Consequently, there is a small chance that the peak load of all households coincide and the distribution line constraints are violated. Therefore, instead of hard constraints, it is more realistic to compute solutions for which the chance that a constraint is violated is very low. Based on these considerations we present an approach to compute policies for which the number of constraint violations is lower than a given threshold, rather than preventing violations completely. Intuitively, we expect that it makes policies less conservative and that the policies give higher expected reward.

Our approach is based on the idea that we can plan with capacity limits $\ell'_{\rho,\tau}$ that are lower than the actual capacity limits $\ell_{\rho,\tau}$ used in the original constraints[3]. Intuitively, this gives solutions in which the expected power flows are lower, and this also reduces the likelihood that the actual capacity limits $\ell_{\rho,\tau}$ are violated during policy execution. Figure 6.2 provides an informal illustration of reducing the limits. In Figure 6.2a planning is performed using a constraint which defines that the expected flow should be at most $\ell_{\rho,\tau}$. Therefore, the actual realization of the flow centers around this expectation, as illustrated using the distribution represented by the gray bars. As can be seen, violations of the limit $\ell_{\rho,\tau}$ may still

[2]See the following IEEE standard: IEEE Std 141-1993 – IEEE Recommended Practice for Electric Power Distribution for Industrial Plants, 1994.

[3]A similar concept was introduced for planning with resource limits using column generation for Constrained MDPs (De Nijs et al., 2017). However, the method cannot be applied because it assumes that the resource limits can be set arbitrarily low, which is not the case in this work. Instead, we derive a lower bound on the limits that can be set, and we present an alternative approach to relax the limits.

occur. In Figure 6.2b the reduced limit $\ell'_{\rho,\tau}$ is used during planning, which leads to a lower mean and fewer violations. In the remainder of this section we first formalize reduced limits, after which we introduce an approach to decide how far the limits should be reduced in order to reduce the number of violations.

Without loss of generality we consider a line $(\rho,\tau) \in \mathcal{L}$ at a timestep t, together with its capacity limit $\ell_{\rho,\tau}$. Recall that the constraints for this line ensure that the expected power flow does not exceed $\ell_{\rho,\tau}$ at time t. In order to understand to what extent we can reduce the capacity limit used during planning, we consider the visualization in Figure 6.3, which shows the expected flow $\mathcal{F}_{\rho,\tau,t}$ in line (ρ,τ) created by uncontrollable loads as a function of time t. The dashed line represents the true capacity limit $\ell_{\rho,\tau}$. We only consider the power flow in the line due to loads that we cannot control using planning, because this provides us with information about the lowest possible resource limit $\ell'_{\rho,\tau}$ that we can set in Equation 6.19. The resource limits $\ell'_{\rho,\tau}$ that we use for planning cannot be lower than the expectations visualized in the figure, because then the planning problem immediately becomes infeasible. In other words, the expected flow created by uncontrollable loads can be interpreted as a lower bound on $\ell'_{\rho,\tau}$.

The expected flow $\mathcal{F}_{\rho,\tau,t}$ can be easily obtained by solving the linear program in Equation 6.15 without grid constraints, after which the expectation can be computed as follows:

$$\mathcal{F}_{\rho,\tau,t} = \left| \sum_{\{i=1,\dots,n \mid \text{agent } i \text{ is uncontrollable}\}} \sum_{s\in S_i} \sum_{a\in A_i} x^i_{t,s,a} \cdot C_{i,z_{\rho,\tau}}(s,a) \right|. \tag{6.30}$$

We take the absolute value to eliminate the flow direction, because when defining the reduced limit $\ell'_{\rho,\tau}$ the direction is not relevant. The gap between $\ell_{\rho,\tau}$ and $\mathcal{F}_{\rho,\tau,t}$ indicates to what extent we can reduce the limit, as depicted in Figure 6.3. We compute the reduced resource limit $\ell'_{\rho,\tau}$ as follows:

$$\ell'_{\rho,\tau} = \left(\max_t \mathcal{F}_{\rho,\tau,t}\right) + \alpha \cdot \left(\ell_{\rho,\tau} - \left(\max_t \mathcal{F}_{\rho,\tau,t}\right)\right), \tag{6.31}$$

in which the parameter $0 \le \alpha \le 1$ defines the actual reduction of the limit $\ell_{\rho,\tau}$. We use $\max_t \mathcal{F}_{\rho,\tau,t}$ rather than $\mathcal{F}_{\rho,\tau,t}$ when defining the reduced limit, because otherwise the limit reductions become unnecessarily conservative. It is only important that the reduced limits do not become lower than the expected flows, and therefore the maximum over these expectations suffices. With the current formulation the limit $\ell'_{\rho,\tau}$ cannot become lower than $\max_t \mathcal{F}_{\rho,\tau,t}$, and therefore it is guaranteed that the linear program in Equation 6.15 has a feasible solution for all possible α.

Our new parameter α can be used to control whether the agents are allowed to use the capacity of the lines or not, but without a priori information it is unclear how to set this parameter prior to solving the planning problem. We propose an

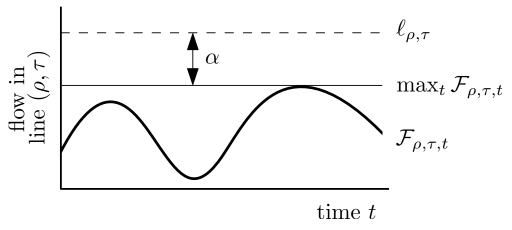

$\ell_{\rho,\tau}$
α
$\max_t \mathcal{F}_{\rho,\tau,t}$
$\mathcal{F}_{\rho,\tau,t}$
flow in line (ρ,τ)
time t

Figure 6.3: Expected uncontrollable flow $\mathcal{F}_{\rho,\tau,t}$ in line $(\rho,\tau) \in \mathcal{L}$ as a function of time t

algorithm which gradually increases α as much as possible based on two criteria. First, the algorithm increases α until it reaches a solution with expected reward $\mathcal{E}$, as defined in Section 6.4.3, because in that case the expected reward is optimal. Second, the algorithm determines empirically whether constraints are violated, and it increases α until it encounters more violations than tolerated. Grid constraints are more strict than the reachability of the agent goals, and therefore the second case is always considered more important than the first when deciding whether the algorithm should proceed.

A more detailed description of our approach is provided in Algorithm 15. The algorithm first determines the expected uncontrollable flows $\mathcal{F}_{\rho,\tau,t}$, after which it starts executing iterations with an increasing α from line 7 to 23. Within an iteration, it solves the linear program in Equation 6.15 with the reduced limits $\ell'_{\rho,\tau}$, and the resulting solution induced by the variables $x^i_{t,s,a}$ is denoted by x' (line 9). We simulate this solution on line 10 to measure the probability p that at least one grid constraint is violated during execution. If this estimate p exceeds the given tolerance $\bar{p}$, then the algorithm stops and it returns the solution x it has found before. If the estimate is lower than the tolerance, then the algorithm uses $\mathcal{E}$ to check whether all the agents are guaranteed to reach their goal. If this is the case, then the algorithm stops and it returns the current solution. Otherwise, it increments α and it proceeds with the next iteration.

Our algorithm uses simulation to obtain an empirical measure of the probability that a grid constraint is violated. This means that the actual violation probability may be lower or higher than the measure p on line 10. However, it is important to note that an exact computation of the violation probability becomes intractable in the multi-agent setting since it requires enumeration of all possible combinations of agent states. Computing an empirical estimate of the violation probability is relatively cheap, and it has shown to perform well in the context of planning for Constrained MDPs (De Nijs et al., 2017).

Finally, we want to highlight a few important properties which demonstrate the benefits of our constraint reduction technique. The planning problems within an iteration are relatively easy to solve because the optimization problem is a linear program and it does not include binary variables. In contrast, the preallocation

Algorithm 15: Solve planning problem with constraint violation tolerance $\bar{p}$

```
input  : violation tolerance $0 < \bar{p} \leq 1$, increment factor $0 < \alpha' \leq 1$
output : solution $x$

1  solve planning problem without constraints using Equation 6.15
2  determine expected uncontrollable flows $\mathcal{F}_{\rho,\tau,t}$ using Equation 6.30
3  $\alpha \leftarrow 0$
4  $x \leftarrow$ NULL
5  $\omega \leftarrow$ number of agents with goals
6  $\phi \leftarrow 1$
7  while $\phi = 1 \wedge \alpha \leq 1$ do
8      define reduced limits $\ell'_{\rho,\tau}$ based on Equation 6.31
9      solve planning problem using $\ell'_{\rho,\tau}$ to obtain $x'$
10     simulate solution $x'$ to measure constraint violation probability $p$
11     if $p > \bar{p}$ then
12         $\phi \leftarrow 0$
13     else
14         compute $\mathcal{E}$ using Equation 6.17
15         if $\mathcal{E} = \omega$ then
16             $\phi \leftarrow 0$
17             $x \leftarrow x'$
18         else
19             $x \leftarrow x'$
20         end
21     end
22     $\alpha \leftarrow \alpha + \alpha'$
23 end
24 return $x$
```

technique requires a large number of binary variables, and solving the problem is intractable for larger instances. Furthermore, since we are not enforcing hard constraints, it can be expected that the algorithm computes solutions with significantly higher expected reward than the solutions computed based on preallocation of grid capacity. We will test this observation empirically in our experimental evaluation in the next section.

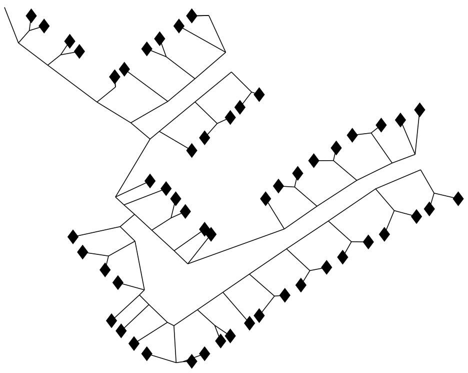


Figure 6.4: IEEE low voltage test feeder represented by 112 nodes and 155 lines, to which 55 households (marked in black) are connected

6.5. Experiments

In this section we present the results of our experimental evaluation, in which we demonstrate how our proposed techniques can be applied for congestion management in distribution grids. We consider a distribution grid with conventional households whose uncontrollable power consumption behaves stochastically over time. In addition, we consider renewable generators at the distribution level and electric vehicles which require power from the grid to charge their batteries. Since these vehicles may introduce congestion, our algorithms are used to decide when the vehicles should charge while preventing congestion.

6.5.1. Experimental setup

Our experiments are executed in a simulation environment in which we simulate households and renewable generators that are connected to the distribution grid. Furthermore, electric vehicles use an MDP policy to decide online when they charge their batteries. In the remainder of this subsection we provide descriptions of the models that we use for planning and simulation.

Distribution grid

We use the IEEE European low voltage test feeder[4] for our experiments (Schneider et al., 2018). This test feeder depicts a low voltage distribution system that is radial, but can be meshed. In Figure 6.4 the test feeder is represented by 112 nodes and 155 lines, to which 55 households are connected in total. The parameters and line capacities of the lines are provided by IEEE. Additionally, 100 representative load profiles, spanning a full day, are provided for the households.

[4]The test feeder data and additional information can be found at the power and energy society of IEEE: `http://sites.ieee.org/pes-testfeeders/resources/`

The test feeder is connected to a transmission grid which ensures that the power demand and supply in the test feeder are always balanced. We use the details of the grid to model the grid according to the models presented in Section 6.3. Additionally, we use the resources and constraints described in Section 6.4.4 to integrate the grid constraints in our planning algorithm. The constraints have been set in such a way that the connection to the transmission grid has a capacity of 75 kW.

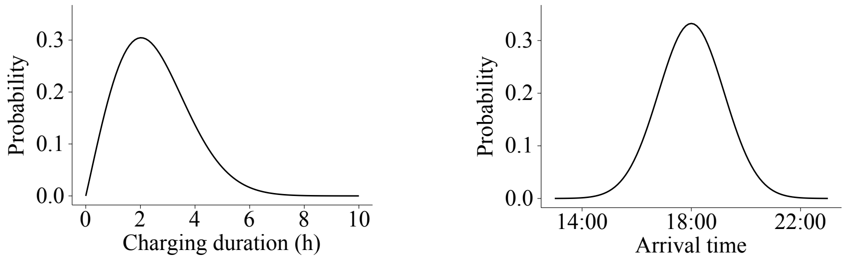
It is expected that the connection to the transmission grid is the heaviest loaded line, because the IEEE grid is radial, the contribution of distributed energy resources (e.g., batteries and solar panels) is marginal, and there is only a single connection to the transmission grid. To improve readability, we will only display the power flow of the line connecting the feeder to the transmission grid in our experiments, however congestion is examined for all the lines in the system during planning and evaluation. An additional experiment with multiple congested lines is described in Section 6.5.8.

Conventional uncontrollable load

Each household connected to the grid requires power throughout the day to supply loads such as the washing machine, dishwasher, fridge and television. This consumption behavior is stochastic and it is assumed that households keep full control over this consumption, which means that it cannot be controlled by external entities such as aggregators. Typically, households have a consumption peak late in the afternoon when people arrive at home, and a consumption peak early in the morning when people wake up. We model each household as an individual Markov Decision Process in which the power consumption behaves stochastically over time, and the single action does not influence the state transitions. Our model has been constructed in such a way that the total consumption of the households follows the typical power consumption pattern with two peaks. This is further studied in Section 6.5.2. More details about the modeling of households can be found in Appendix 6B.

Electric vehicles

We model electric vehicles which include two sources of uncertainty. First, there is uncertainty about the arrival time of the vehicle, because typically it is not known precisely when someone arrives at home after, e.g., a day at work. Second, there is uncertainty about the amount of time required to charge the battery, because the state of charge of the battery only becomes known upon arrival. Once connected to the charging station at home, it is assumed that the vehicle remains connected until the next morning, which means that the electric vehicles naturally provide flexibility if the amount of time required for charging is less than the total time connected.



(a) Distribution for charging duration (b) Distribution for arrival time

Figure 6.5: Distributions used for modeling the uncertainty of electric vehicles

In the literature various studies have been conducted to analyze charging behavior based on the data collected from a large number of charging stations. In particular, Khoo et al. (2014) present distributions for the duration of charging sessions, the energy usage during charging and starting times of charging sessions in a corporate and household setting. Since these distributions have been constructed and validated based on data collected in the real world, we decided to use these distributions to model the uncertainty of our electric vehicles. It turns out that the total duration of charging sessions follows a Weibull distribution for small fully electric vehicles charged at households. The corresponding probability density function is defined as:

$$\frac{k}{\lambda} \times \left(\frac{x}{\lambda}\right)^{k-1} \times e^{-(x/\lambda)^k}, \tag{6.32}$$

with $k = 2.022$ and $\lambda = 2.837$, in which x represents the charging duration in hours. The distribution we derived is visualized in Figure 6.5a, and the MDPs of our electric vehicles are constructed in such a way that the total amount of time required for charging is sampled from this distribution. Due to the discrete number of timesteps considered in our planning model, we discretize the distribution accordingly when constructing the MDP models. For the arrival time of vehicles we define a distribution whose mean is centered around 18:00, which is consistent with the statistics for household charging presented in multiple studies (Khoo et al., 2014; Sadeghianpourhamami et al., 2018). In particular, we assume a Gaussian distribution with the mean at 18:00 and a standard deviation of 1.2 hours. The distribution we use for the vehicle arrival time is visualized in Figure 6.5b. It should be noted that our model supports any continuous distribution over arrival times, and therefore our work can also be used with other distributions besides the Gaussian distribution. Finally, it is assumed that the charging rate of the vehicles is equal to 3 kW, which corresponds to a typical slow charging rate at home, and this charging rate is consistent with the statistics found by Khoo et al. (2014) for small fully electric vehicles.

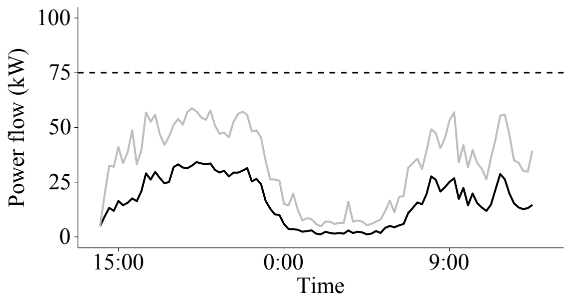


Figure 6.6: Mean uncontrollable flow (black), maximum uncontrollable flow (gray), and the capacity limit of the power line (dashed)

Renewable power generation using solar panels

Similar to the power consumption of households, the power generation of renewables can be considered uncontrollable. The generated power is only dependent on exogenous weather conditions and it cannot be influenced by the households. Therefore, renewables can be be treated similarly to an uncontrollable load from a modeling point of view, but with power generation instead of consumption.

Fortunately, the generated power of solar panels can be forecasted with high accuracy (Bizzarri and Mucci, 2018). Therefore, we model solar panels as MDPs which only include limited uncertainty ($\pm 10\%$). A description of these MDP models is provided in Appendix 6B. In our experiments we include 10 households with solar panels.

6.5.2. Uncontrollable flows

In our first experiment we consider the stochastic behavior of the households connected to the grid. The power consumption of these households cannot be controlled explicitly, but it needs to be taken into account while performing planning for congestion management. We consider all the households and their solar panels, for which we run 10000 simulation runs representing a full day starting at 14:00. This time window was selected because it allows us to study overnight charging of electric vehicles in subsequent experiments. During the simulation runs we measure the power flows created due to the power consumption of the households. In Figure 6.6 we consider the power flows through the line that connects the distribution grid to the transmission grid, which is the line in which congestion typically arises first since it serves all loads, as we discussed in Section 6.5.1. The black line represents the mean power flow as a function of time, which shows us a pattern with two demand peaks. There is a demand peak during the late afternoon and early evening, which is a time window during which people typically arrive at home. There is another demand peak during the early morning when people

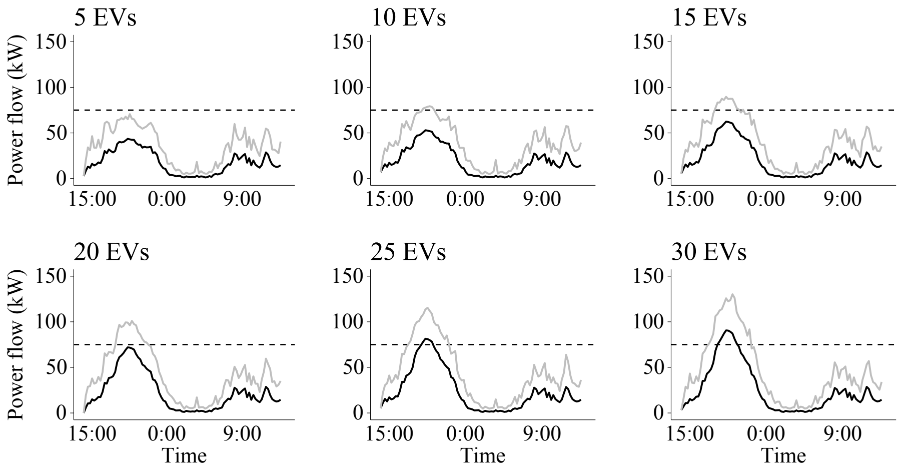


Figure 6.7: Mean and maximum flow with multiple EVs without coordination

wake up and go to work. The gray line represents the maximum flow measured during the simulation runs, which is not allowed to exceed the capacity limit that is represented by the dashed line at 75 kW. As can be seen, the maximum flows observed during simulation are always below the capacity limit, and during several parts of the day a significant part of the line capacity remains unused. This unused capacity can be used by electric vehicles for which the flexible consumption can be shifted in time.

6.5.3. Uncoordinated electric vehicle charging

In this experiment we study whether electric vehicles introduce grid congestion in case they charge their batteries in an uncoordinated manner. This means that the vehicles have uncertain arrival time and demand, but upon arrival they start charging immediately without any coordination or planning. This potentially creates a significant peak in power demand if a large number of vehicles starts charging at the same time. This is confirmed by the graphs in Figure 6.7, which shows the mean and maximum flows for an increasing number of EVs. For these graphs it is important to note that we consider total flows due to households, solar panels and electric vehicles. If the number of EVs is relatively low then there is no congestion. However, when increasing the number of EVs further then the demand peak grows, which confirms our expectations.

We also measure congestion by counting the number of simulation runs during which grid congestion was encountered in at least one of the lines. In Figure 6.8 we depict this as a function of the number of electric vehicles. As expected, when increasing the number of electric vehicles we gradually encounter congestion

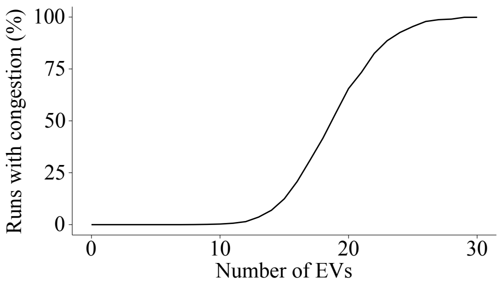


Figure 6.8: Number of runs with grid congestion, as a function of the number of EVs

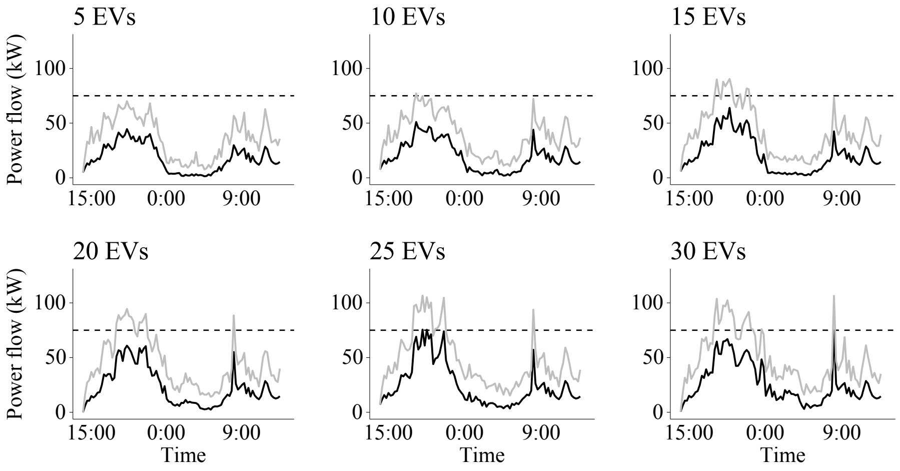


Figure 6.9: Mean and maximum flow with multiple EVs and constraints on expectations

due to significant power demand. The results of this experiment show us that grid congestion becomes more severe if there is a large number of EVs. From the experiment we can conclude that uncoordinated charging potentially leads to grid congestion, and therefore it becomes infeasible to charge a large number of vehicles simultaneously.

6.5.4. Constraints on expected power flows

The standard Constrained MMDP model, as defined in Equation 6.15, imposes constraints on expectations. This means that the resulting solution only prevents violations of grid constraints in expectation, and constraints may still be violated during execution. In this experiment we empirically test whether this is indeed the case. Compared to the experiment with uncoordinated charging we make two changes. First, we include grid constraints in the model, based on Equation 6.19. Second, we add flexibility to the electric vehicles, such that charging load can be

Number of EVs	5	10	15	20	25	30
Time (s)	2.31	3.15	4.67	7.52	9.02	31.88

Table 6.1: Runtime of solving the linear program with constraints on expectations

Number of EVs	5	10	15	20	25	30
Time (s)	72.81	76.83	85.14	96.07	116.88	128.81

Table 6.2: Runtime of solving the planning problem with empirical bounding

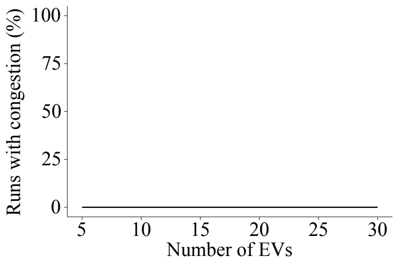
temporally shifted while solving the planning problem if the constraints require this. The vehicles are allowed to charge their batteries until 8AM, which represents the charging deadline of the electric vehicles.

Similar to the previous experiment, we visualize the flows in Figure 6.9. The black line representing the mean flow is below the limit, which is what we intuitively expected because the computed solution ensures that the flows are below the limit in expectation. Unfortunately, the maximum flows can still exceed the capacity limit in some cases, which is confirmed by the gray line that exceeds the dashed limit in most of the graphs. The spike around 8:00 is caused by the fact that vehicles are allowed to charge until 8:00, and in some cases the planner has decided that vehicles still require charging in that hour in order to collect reward. The running times required for solving the individual linear program are reported in Table 6.1, which shows that the running time gradually increases when the size of the linear program increases. To summarize, our experiment has shown that the planning problem can be solved quickly due to the linearity of the optimization problem, but at the same time it confirms that constraints on expectations are indeed not sufficient to ensure that power flows do not exceed the capacity limits.

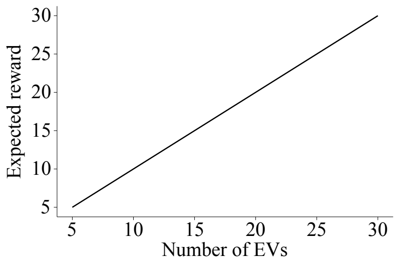
6.5.5. Reducing violations using empirical bounding

Now we evaluate the planning algorithm which reduces the number of constraint violations by testing empirically whether violations may occur, as presented in Algorithm 15. We execute this algorithm with tolerance $\bar{p} = 0.001$ and increment factor $\alpha' = 0.02$. Similar to the previous experiments, we run the algorithm on instances with an increasing number of electric vehicles ranging from 5 to 30 vehicles. We study algorithm performance based on two criteria. Most importantly, we study whether the algorithm provides solutions which respect the grid constraints. Furthermore, we study whether the solutions ensure that the vehicles reach a fully-charged battery before departure.

In Figure 6.10a we show the number of simulation runs with violations. As



(a) Number of runs with congestion



(b) Expected reward

Figure 6.10: Results for Constrained MMDP solutions computed with empirical bounding

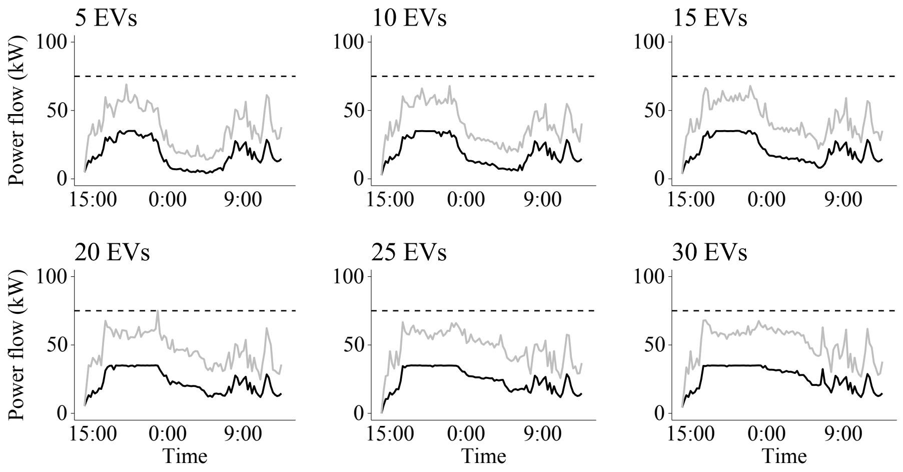


Figure 6.11: Mean and maximum flow for multiple EVs with empirical bounding

can be seen, the solutions computed by our algorithm ensure that no violations of grid constraints occur during execution of the policies. This is an important result, because it confirms that our algorithm is able to take realistic grid constraints into account while solving the planning problems. As discussed in Section 6.4.3 and Theorem 9, it is guaranteed that the electric vehicles reach a fully-charged battery in case the total expected reward equals the number of electric vehicles. Therefore, we expect that the expected reward of the solutions increases linearly when increasing the number of electric vehicles. Figure 6.10b confirms this observation, and together with the results in Figure 6.10a it implies that our algorithm ensures that vehicles reach a fully-charged battery while respecting the constraints imposed by the grid.

The flows during policy execution are visualized in Figure 6.11. The maximum flows are always below the capacity limit indicated by the dashed line, which means that no constraint violations were encountered. It is interesting to compare the results with Figure 6.7 and Figure 6.9. In the first figure the constraints were violated due to the lack of coordination, while constraints were violated in the second figure due to the type of constraints added to the linear program. Our algorithm based on empirical bounding, on the other hand, computes solutions for which both the mean and maximum flows are below the limits.

The runtimes are reported in Table 6.2. Compared to the previous experiment it takes more time to solve the problem, mainly because the algorithm solves the linear program multiple times and it executes simulation runs to estimate the violation probability. However, from a practical point of view the runtimes are still very low and in practice it does not need to be faster. As a final remark, we want to emphasize that the vehicles execute their charging policies in a decentralized way. This means that grid constraints are respected without any form of communication during execution.

6.5.6. Flexibility of electric vehicles

In our next experiment we study to what extent the amount of flexibility influences the ability to reach a fully-charged battery. Intuitively, if there is limited flexibility available, then it becomes more difficult to shift consumption to other periods of the day, which potentially means that not all the vehicles can be charged. We test this hypothesis by increasing the amount of flexibility available, and we keep track of the mean reward of the solution. It can be expected that this increases when we increase the amount of flexibility available. Throughout the experiment we only consider the scenario with 10 electric vehicles, which means that we only vary the flexibility.

In Figure 6.12 we report the expected reward as a function of flexibility, in which flexibility is characterized by the deadline after which charging cannot be performed anymore. For example, the deadline 0:00 means that vehicles are not able to charge after midnight. As expected, we can see that increasing the flexibility leads to an increase of the expected reward. If a sufficient amount of flexibility is available then all vehicles are guaranteed to reach a fully charged battery. Under these circumstances increasing the flexibility even more does not lead to an increase in expected reward, because then there is more flexibility available than necessary. In our scenario with 10 electric vehicles we can see that all vehicles can be fully charged if charging can be performed until midnight, and additional flexibility does not increase expected reward. With limited flexibility it is not guaranteed that all vehicles get fully charged. For example, if charging can be performed until 22:00 then approximately 9 vehicles reach their maximum battery level in expectation.

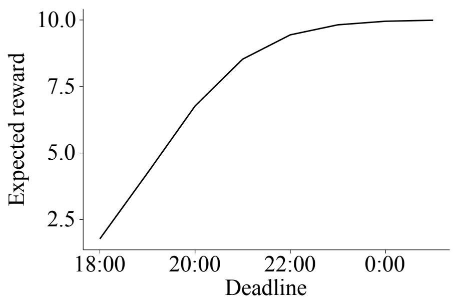


Figure 6.12: Influence of deadline on expected reward

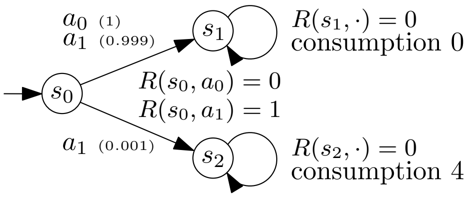
6.5.7. Preallocation of grid capacity

An alternative to empirical bounding of constraint violation probabilities is preallocation of grid capacity, as discussed in Section 6.4.5. This approach ensures that the probability of constraint violations is always 0, but due to this strict guarantee the solutions may become extremely conservative. This means that the solutions do not fully utilize the grid capacity that is available, which subsequently gives lower expected reward. In this section we empirically test whether this is indeed the case, and we show that our empirical bounding technique provides solutions with higher expected reward.

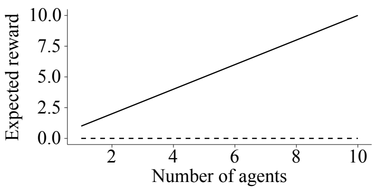
We perform our experiment using a synthetic MDP which has been constructed in such a way that it is easy to understand why the preallocation technique gives low reward. The MDP consists of 3 states and 2 actions, and it is visualized in Figure 6.13a. The main idea is that it is required to execute action a_1 in the first step in order to maximize the reward, but at the same time there is a small probability $\beta > 0$ that this leads to constraint violations in all subsequent steps. The preallocation technique enforces that the probability of constraint violations is 0, and therefore it never executes a_1 in the first step, which means that the expected reward of the solutions is always 0.

To be more specific, action a_0 always transitions to state s_1, while action a_1 may also transition to state s_2 with low probability $\beta > 0$. The states s_1 and s_2 are absorbing and hence they cannot be left. The capacity of all lines of the grid is set to 3, such that consumption 4 immediately leads to a constraint violation. The preallocation technique should execute a_0 in the first step, such that the power consumption is guaranteed to be zero in the remaining steps. The empirical bounding technique may choose action a_1 in the first step, which gives immediate reward, but there is a negligible risk that this leads to power consumption 4 in all remaining steps.

The comparison of preallocation and empirical bounding is shown in Figure 6.13b. The expected reward of the solutions computed using preallocation



(a) Synthetic MDP with $\beta = 0.001$



(b) Expected reward of preallocation (dashed) and empirical bounding (solid) solutions

Figure 6.13: Comparison preallocation and empirical bounding

is indeed always 0, as indicated by the dashed line, which means that it never executes action a_0 in the first step. For our empirical bounding technique we set the tolerance $\bar{p}$ in such a way that the probability to reach constraint violations is lower. Note that we can define the probability $\beta > 0$ of this state transition arbitrarily close to 0. The solutions computed by our empirical bounding technique do actually execute action a_1 in the first step and the solutions give positive expected reward, as indicated by the solid line. This synthetic example confirms our initial expectation that solutions computed using preallocation may be unnecessarily conservative. Under these circumstances our empirical bounding technique can be used to obtain solutions with higher expected reward while the probability of constraint violations remains empirically bounded.

6.5.8. Multiple binding grid constraints

In a distribution grid with a radial topology congestion typically arises in the lines close to the point where the grid is connected to the transmission grid. This happens because the grid has a tree structure, and for all the loads the power is supplied through one of these lines, which become congested first. There are multiple scenarios, however, in which multiple constraints at different locations in the grid become congested, and under these circumstances it is important that planning algorithms consider all constraints. For example, if the grid has a mesh structure in which there are cycles in the grid topology, then all the flows and the directions of these flows can change dramatically due to a change in demand in one individual node. Another scenario is a distribution grid with storage devices which discharge at specific times (e.g., vehicle to grid), which means that there are multiple nodes in which a significant amount of power is injected, in addition to the supply from the transmission grid. In this section we describe an experiment in which we show that our planning techniques support such settings. Furthermore, the experiment confirms that congestion management methods for distribution grids have to consider the constraints imposed by all the lines.

We consider the distribution grid shown in Figure 6.14, which consists of 9 nodes. The specific setting has been constructed in such a way that it enables us to test empirically whether our algorithms can deal with multiple congestion points. It is assumed that there are 7 households with a constant consumption of 1 kW, represented by the small boxes. There is a storage device connected to node 8, which provides a constant power output that is equal to 7 kW, which means that all the consumption of the households is supplied using the battery, and there is no additional power from the transmission grid in node 0. The line $(0,1)$ has capacity 12, and the line $(4,8)$ has capacity 4.9. The capacities of the other lines are irrelevant because they have been set much higher such that no congestion arises there. The grid is a direct current distribution grid in which the lines have resistance 0.0097511.

The households marked with a dot have an electric vehicle with a charging rate that is equal to 3 kW. During charging this additional power needs to be supplied through node 0 because the battery only provides 7 kW. At most 4 vehicles can be charged simultaneously due to the capacity of line $(0,1)$. In case there are fewer than 5 charging vehicles then the limit of line $(0,1)$ is not violated, but the capacity limit of line $(4,8)$ may be exceeded, depending on the specific combination of vehicles that charges at the same time. In other words, besides the number of charging vehicles it is also relevant to consider specific vehicle combinations that charge simultaneously.

We illustrate the interplay of electric vehicles and constraints using a few numerical examples in a DC distribution grid. For convenience we use the term vehicle j to refer to the vehicle connected to node j. Charging vehicle 2 and 3 at the same time creates a 4.63 kW flow from node 8 to 4, which is feasible. However, if vehicle 2 and 4 charge at the same time then this flow becomes equal to 5 kW, which is infeasible. In both cases there are only 2 vehicles that charge simultaneously, but depending on their location they can create congestion. It is interesting to observe that we can actually charge vehicle 2 and 4 simultaneously if vehicle 6 also charges, because in that case the flow decreases to 4.25 kW. In other words, the feasibility of simultaneous charging of vehicle 2 and 4 depends on other vehicles that charge at the same time.

The example grid is used to test whether our algorithms compute solutions which respect the constraints in this difficult scenario. Figure 6.15 visualizes the flows in the lines $(0,1)$ and $(4,8)$ without planning (unconstrained) and with our congestion management techniques (constrained). As can be seen, without any planning both constraints get violated, while the solutions computed by our algorithms ensure that multiple constraints are respected. This confirms that our congestion management algorithms are able to deal with multiple congestion points and difficult interactions between nodes.

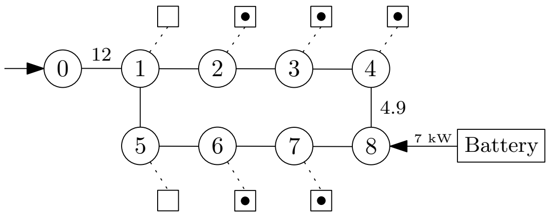


Figure 6.14: Distribution grid with storage and multiple congestion points

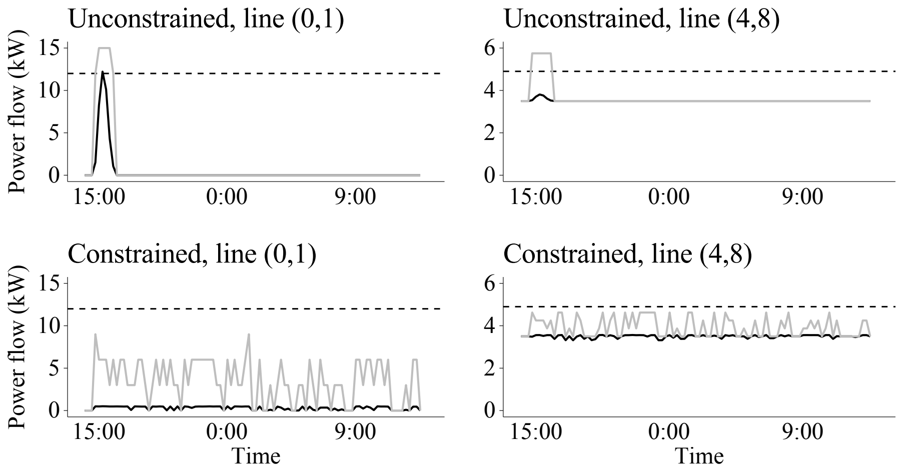


Figure 6.15: Power flows for the experiment involving multiple binding constraints

6.6. Conclusions

The power grid that serves our society faces major changes due to the increased number of electric vehicles and the integration of renewables. Distribution grids in neighborhoods and cities may become congested, which means that the grid is unable to accommodate the total power demand due to the limited capacities of power lines. Increasing the capacity of the grid is a straightforward solution to prevent congestion, but it is needless to say that it represents significant infrastructural work and it can be very expensive. In this chapter we investigated how artificial intelligence techniques can provide an alternative solution to congestion management problems, such that grid reinforcements are not required.

We provided an overview of ongoing work in the artificial intelligence community related to the development of smart grids. This literature study has shown that artificial intelligence is used for control of flexible loads, the participation in markets, grid control, load forecasting and smart metering. Furthermore, we found that existing artificial intelligence techniques are currently not suitable for congestion management because power grid constraints are typically not considered. We close this knowledge gap by presenting a planning framework for congestion management based on Constrained Multi-agent Markov Decision Processes (CMMDPs). We described a novel mapping of power grid constraints to CMMDP constraints, and we introduced two techniques which ensure that loads in a distribution grid respect these grid constraints while executing their decision making policy. In a series of experiments based on a realistic IEEE distribution grid we have shown that our techniques effectively prevent congestion in a setting where charging of multiple electric vehicles can be shifted in time. This demonstrates that our planning framework effectively deals with constraints. It also shows that techniques developed in the artificial intelligence community can play a key role in solutions to problems with significant societal impact in the context of smart grids.

We envision two main directions of future work which are directly relevant for further development of smart grids in our society. First, rather than performing online control of flexible loads, our framework can be used to build decision support systems which allow organizations to study where congestion may occur in the future. For example, it can be studied how the share of electric vehicles in certain neighborhoods affects congestion, which is relevant to know when deciding about deployment of public charging stations in the residential area. Second, there is increased interest in deployment of large-scale batteries in distribution grids, which contribute to congestion management by charging and discharging at the right time. Such batteries can be integrated in our current model, and the resulting model can also be used to decide where such batteries should be located in order to be effective. Finally, from a mathematical point of view it is interesting to investigate whether the probability distributions of the individual MDPs can be used to reason about changes in the constraint violation probabilities due to changes in the limits used in the constraints. This information can be used to make our empirical bounding technique more efficient.

Appendix 6A. AI literature on smart grids

In this appendix we provide a full overview of the AI literature that we have identified in the literature study in Chapter 6. Table 6.3 gives a complete overview of AI literature on flexible loads. Literature on grid control, integration of renewables, load forecasting and smart metering is presented in Table 6.4 and Table 6.5. Table 6.6 presents literature focusing on markets, tariffs and energy trading. In all tables the column **F** indicates whether the work considers realistic grid constraints.

Authors	**Year**	**Venue**	**Description**	**F**
De Nijs et al.	2018	AAAI	Planning for constraints that change stochastically	
De Weerdt et al.	2018	IJCAI	Complexity analysis of EV charging problems	
Fioretto et al.	2017	AAMAS	Scheduling method for loads based on DCOP	
Walraven et al.	2016	ECAI	MDP-based approach for scheduling EV charging	
Kuppannagari et al.	2016	IJCAI	System demo of a demand response system on campus	
Styler et al.	2015	AAAI	Control algorithm for an EV based on forecasts	
De Nijs et al.	2015	AAAI	Best-response strategy for resource-constrained agents	
Valogianni et al.	2015	AAMAS	Pricing mechanism for control of flexible EVs	
Marinescu et al.	2015	AAMAS	Multi-agent reinforcement learning method for smart grids	
Angelidakis et al.	2015	AAMAS	MDP formulation for prosumer decision making	
Valogianni et al.	2014	AAMAS	Decentralized charging strategy for multiple EVs	
Shann et al.	2014	AAMAS	Computes a policy for a heating system using MDPs	
Van Den Briel et al.	2013	IJCAI	Distributed load control method for smart appliances	
Shann et al.	2013	IJCAI	Learning strategy for heating homes	
Reddy et al.	2012	AAAI	Factored representation for a smart grid decision problem	
Ramchurn et al.	2011	AAMAS	Multi-agent system for demand-side management	
Vandael et al.	2011	AAMAS	Scheduling method to prevent imbalance	

Table 6.3: AI literature focusing on control of flexible loads

Authors	**Year**	**Venue**	**Description**	**F**
Coninx et al.	2016	AAMAS	Coordination of wind turbines to prevent violations	✓
Andoni et al.	2016	AAMAS	Studies curtailment and line reinforcements	
Bandyopadhyay et al.	2016	ICAPS	Studies a curtailment problem and a bandit approach	
Agrawal et al.	2015	AAMAS	Studies a power grid restoration after line failures	✓
Chau et al.	2014	AAMAS	Power allocation strategy based on a knapsack problem	
Piacentini et al.	2013	ICAPS	Connects classical planner with power flow software	✓
Yu et al.	2013	AAMAS	Presents a knapsack problem for power allocation	
Miller et al.	2012	AAMAS	Applies DCOP to a generator dispatch problem	
Fox et al.	2011	ICAPS	Planning approach for usage of multiple batteries	

Table 6.4: AI literature focusing on grid control and integration of renewables

Authors	Year	Venue	Description	F
Valovage et al.	2018	AAAI	Identify appliances from aggregated metering data	
Chen et al.	2018	IJCAI	Load forecasting method based on time series	
Pang et al.	2018	IJCAI	Load forecasting method based on time series	
Wang et al.	2018	AAMAS	Load forecasting method based on random fields	
Hao et al.	2015	IJCAI	Studies meter accuracy and meter deployment	
Chen at al.	2013	IJCAI	Wind forecasting method based on Gaussian processes	
Parson et al.	2012	AAAI	Identify appliances from aggregated metering data	

Table 6.5: AI literature focusing on forecasting and smart meters

Appendix 6B. Details on experimental setup

In this appendix we provide additional information about the MDP models that we use for modeling households, electric vehicles and renewable generators in the experiments in Chapter 6. When describing an MDP model we always refer to an individual agent i, and therefore we omit the index i in the description.

MDP for households

We use an MDP model in which the state represents the total consumption of a household. The transition model of the households was constructed based on IEEE load profiles which define the time-dependent consumption for a period of 24 hours[5].

We consider a time horizon of 24 hours, discretized into 15 minute intervals, and we use 10 discrete states that characterize the power consumption of the household. For each household we assign an initial state by using the first state of an IEEE load profile that we select uniformly at random. We construct a state transition model by iterating over all observed state transitions (s, a, s') in the IEEE data set at time t. Based on these transitions we construct the transition function $T(s, a, s', t)$.

The power consumption for a given state is formalized using the power consumption function $c : S \times A \to \mathbb{R}$. Assuming that we have 10 states $s \in \{0, 1, \ldots, 9\}$, we define the power consumption as $c(s, a) = 535 \times s$. The constant 535 has been set in such a way that the states span the full range of consumptions observed in the IEEE data.

There is only one dummy action in the model, which does not influence the state transitions. The number of states in our MDP model has shown to be sufficient to get the aggregate load pattern with two demand peaks for 55 households. If necessary the accuracy can be adjusted by modifying the number of states.

[5]See `http://sites.ieee.org/pes-testfeeders/resources/` for more details.

Authors	**Year**	**Venue**	**Description**	**F**
Perez-Diaz et al.	2018	AAMAS	Model of multiple self-interested EV aggregators	
Mrkos et al.	2018	AAMAS	Dynamic pricing method based on MDPs	
Burgess et al.	2018	AAMAS	Trading for multiple agents subject to grid constraints	✓
Yang et al.	2018	IJCAI	Broker strategy based on reinforcement learning	
Methenitis et al.	2017	AAMAS	Mechanisms for electricity trading	
Ma et al.	2017	AAMAS	Mechanisms for bidding in a demand response setting	
Stein et al.	2016	AAMAS	User interface for EVs participating in a market	
Gerding et al.	2016	IJCAI	Market mechanisms for EV parking lots	
Bandyopadhyay et al.	2016	AAAI	Mechanisms for dynamic pricing	
Chowdhury	2016	AAAI	Describes automated agents in wholesale markets	
Urieli et al.	2016	AAMAS	Description of an MDP-based agent in Power TAC	
Methenitis et al.	2016	IJCAI	Tariff scheme to incentivize intelligent agent behavior	
Ma et al.	2016	IJCAI	Mechanisms for incentivizing truthfulness	
Kahlen et al.	2015	AAAI	Algorithms for EVs participating in a reserve market	
Cerquides et al.	2015	AAMAS	Market for trading subject to constraints	✓
Hernandez-Leal et al.	2015	AAMAS	Algorithm to learn a model of an opponent in trading	
Strawser et al.	2015	AAMAS	Proposal for a reliability market in smart grid settings	
Hayakawa et al.	2015	IJCAI	Mechanisms for charging of electric vehicles	
Berlink et al.	2015	IJCAI	Reinforcement learning for buying and selling energy	
Perrault et al.	2015	IJCAI	Market for matching producers and consumers	
Bandyopadhyay et al.	2015	IJCAI	Game theoretic perspective on real-time pricing	
Kahlen et al.	2014	AAMAS	Study of a trading strategy for multiple EVs	
Urieli et al.	2014	AAMAS	Agent for the Power Trading Agent Competition	
Alan et al.	2014	AAMAS	Considers a tariff switching problem	
Jain et al.	2014	AAAI	Presents a demand response mechanism	
Ketter et al.	2013	AAAI	Description of the Power Trading Agent Competition	
Reddy et al.	2013	AAAI	Learning method to select tariffs	
Rose et al.	2012	AAMAS	Mechanism for aggregate demand prediction in smart grids	
Reddy et al.	2011	IJCAI	Q-learning approach for developing pricing strategies	
Voice et al.	2011	AAAI	Presents pricing scheme for decentralized control of storage	
Reddy et al.	2011	AAAI	Presents strategies and simulation models for trading agents	
Vasirani et al.	2011	AAMAS	Considers coalitions of generators to participate in markets	
Gerding et al.	2011	AAMAS	Mechanisms for charging of electric vehicles	
Kamboj et al.	2011	AAMAS	Multi-agent systems of EVs participating in a market	
Chalkiadakis et al.	2011	AAMAS	Mechanism to incentivize creation of energy cooperatives	
Vytelingum et al.	2010	AAMAS	A market strategy for control of storage in smart grids	
Vytelingum et al.	2010	AAMAS	A proposal for an electricity market	

Table 6.6: AI literature focusing on energy markets, tariffs and trading

MDP for electric vehicles

We use an MDP model in which the states represent whether a vehicle is connected, and whether the vehicle requires charging. The action a_0 corresponds to being idle, and the action a_1 corresponds to charging during the next time step. We provide a high-level description of the state transitions, the reward function and the power consumption function.

The state s_0 represents that the vehicle is not connected. In state s_1 the vehicle is connected, but it has no demand and it does not require charging. Regardless of the executed action, at time t there can be a state transition from state s_0 to a state s_d with $d > 1$, representing the event that the vehicle arrives. The probability of such a state transition is determined based on the distribution in Figure 6.5b. In particular, the vehicle arrives at time t with probability $P_A(t)$ defined by:

$$P_A(T) = \begin{cases} P(X=t) & t=0 \\ P(X=t) \;/\; \prod_{t'=1}^{t-1}(1-P_A(t')) & \text{otherwise} \end{cases}, \tag{6.33}$$

in which X is the underlying random variable of the Gaussian distribution. In that case the state transitions to a state s_d with $d > 1$. The probability distribution over these states s_d is determined by the Weibull distribution shown in Figure 6.5a and Equation 6.32. In other words, the transition to a state s_d representing charging demand is determined by both the arrival time distribution and the charging duration distribution. The state remains s_0 with probability $1 - P(X \leq t)$. State s_1 always transitions to state s_1 with probability 1. The state s_d transitions to s_{d-1} when executing a_1, and it transitions to s_d when executing action a_0.

The reward function has been designed in such a way that the agent gets reward 1 when reaching a fully-charged battery, consistent with Section 6.4.3. This can be achieved by defining $R(s_2, a_1) = 1$, and the reward of all other state-action pairs is equal to 0.

The agent requires power in case it charges the battery, and it is assumed that the charging rate is 3 kW, as discussed in Section 6.5.1. We formalize this using the power consumption function $c : S \times A \to \mathbb{R}$. We define $c(s_d, a_1) = 3$ for $d > 1$, and 0 otherwise. It is important that the unit in the power consumption function corresponds to the unit that is used in the constraints (i.e., when defining the consumption in kW then the constraints should also be defined in kW).

MDP for solar panels

As discussed in Section 6.5.1, renewables bring limited uncertainty in the system. Therefore, we model a time-dependent power production profile as shown in Table 6.7. It represents the production of a small solar panel that injects at most 100 W into the grid. We use a discrete state space that characterizes the power production within this range, discretized into 10 W intervals. The time-dependent state transitions are defined in such a way that the state transitions to the power production listed in Table 6.7 with probability 0.9. With probability 0.05 the state transitions to a state one production interval higher or lower, which represents the limited uncertainty. Finally, we want to note that there is just one action which

Hour of day	7	8	9	10	11	12	13	14	15	16	17	18
Production (W)	10	20	30	50	80	90	70	60	50	40	20	10

Table 6.7: Time-dependent power production of a small renewable generator

$\mathcal{P}(0)$	$\mathcal{P}(1)$	$\mathcal{P}(2)$	$\mathcal{P}(3)$	$\mathcal{P}(4)$	$\mathcal{P}(5)$	$\mathcal{P}(6)$	$\mathcal{P}(7)$	$\mathcal{P}(8)$	$\mathcal{P}_{4,8}$
6000	-1000	-4000	-4000	-1000	-1000	-1000	-1000	7000	-4625
6000	-1000	-4000	-1000	-4000	-1000	-1000	-1000	7000	-5000
9000	-1000	-4000	-1000	-4000	-1000	-4000	-1000	7000	-4250

Table 6.8: Numbers used in flow calculations in Section 6.5.8

does not influence the state transitions, and the rewards are always zero. The resulting MDP models an exogenous process that is not influenced by the decision maker.

Meshed grid in experiments

In Section 6.5.8 we consider a meshed distribution grid, and for specific combinations of power injections we provide the power flows in the line $(4,8)$. These numbers can be reproduced by modeling a DC distribution grid using the information in the manuscript corresponding to the chapter. It is assumed that each line in the power grid has resistance 0.0097511. The power flow in the line $(4,8)$ is computed based on the power injections $\mathcal{P}(j)$ in nodes j. These injections as well as the resulting flows are provided in Table 6.8.

Appendix 6C. Distribution grid model for AC grids

This appendix provides the derivation of the power flow approximation for AC distribution grids. The power flow equation for the real power of bus i is as follows:

$$P_i = \sum_{j=0}^{n-1} |V_i||V_j|(G_{i,j}\cos(\theta_i - \theta_j) + B_{i,j}\sin(\theta_i - \theta_j)) \tag{6.34}$$

where P_i is the power injected at bus i, $G_{i,j}$ is the real part of the Y-bus admittance matrix element in row i and column j, and $B_{i,j}$ is the imaginary part of the Y-bus admittance matrix element in row i and column j. The Y-bus admittance matrix is defined as:

$$Y_{i,j} = \begin{cases} y_{i,i} + \sum_{k\neq i} y_{i,k} & \text{if } j = i \\ -y_{i,j} & \text{if } j \neq i \end{cases} \tag{6.35}$$

where $y_{i,k} = g_{i,k} + \mathrm{j}\, b_{i,k}$ is the admittance of the line from i to j and j is the imaginary unit.

We will simplify Equation 6.34 by approximating the values of $G_{i,j}$ and $B_{i,j}$. In order to do this, we first compute the admittance $y_{i,j}$ of an arbitrary line using the impedance $r_{i,j} + \mathrm{j}\, x_{i,j}$, which is denoted $z_{i,j}$:

$$\begin{aligned} y_{i,j} &= \frac{1}{z_{i,j}} = \frac{1}{r_{i,j} + \mathrm{j}\, x_{i,j}} = \frac{1}{r_{i,j} + \mathrm{j}\, x_{i,j}} \frac{r_{i,j} - \mathrm{j}\, x_{i,j}}{r_{i,j} - \mathrm{j}\, x_{i,j}} = \frac{r_{i,j} - \mathrm{j}\, x_{i,j}}{r_{i,j}^2 + x_{i,j}^2} \\ &= \frac{r_{i,j}}{r_{i,j}^2 + x_{i,j}^2} + \mathrm{j}\, \frac{-x_{i,j}}{r_{i,j}^2 + x_{i,j}^2} \end{aligned} \tag{6.36}$$

Based on this derivation we observe that $g_{i,j} = \frac{r_{i,j}}{r_{i,j}^2 + x_{i,j}^2}$ and $b_{i,j} = \frac{-x_{i,j}}{r_{i,j}^2 + x_{i,j}^2}$. For AC distribution grids it is assumed that the resistance $r_{i,j}$ is very small compared to the reactance $x_{i,j}$, we can then approximate them as $g_{i,j} \approx 0$ and $b_{i,j} \approx \frac{-1}{x_{i,j}}$. Since the real part $G_{i,j}$ of the Y-bus matrix is zero, and the imaginary part $B_{i,j}$ equals $-1 \cdot b_{i,j} = -1 \cdot \frac{-1}{x_{i,j}} = \frac{1}{x_{i,j}}$, we derive the following approximation of the power flow equation:

$$P_i = \sum_{j=0}^{n-1} |V_i||V_j| \left(\frac{1}{x_{i,j}} \sin(\theta_i - \theta_j) \right) \tag{6.37}$$

The equation can be further simplified by observing that $\sin(\theta_i - \theta_j) \approx \theta_i - \theta_j$ if the angle differences are small, which is the case under stable conditions. Moreover, the voltage magnitudes are close to the nominal voltage $\overline{U}$. Now we obtain the simplified power flow equation:

$$P_i = \sum_{j=0}^{n-1} \frac{\overline{U}^2}{x_{i,j}} (\theta_i - \theta_j) \tag{6.38}$$

This equation is the same as Equation 6.2. If j equals i, then angle difference is zero and therefore this term can be discarded from the summation.

7 Conclusions

Developing algorithms for intelligent decision making systems is a key research area in the field of artificial intelligence. In this dissertation we focused on a specific class of decision making problems which include partial observability and constraints. For such problems we have presented a collection of techniques to enhance the scalability of solution algorithms, and we have applied our algorithms to address a real-world planning problem in the context of smart distribution grids. In this final chapter we summarize our contributions, and we explain how these contributions affect current research in the field. Finally, we identify multiple directions for additional research which show how our work can be extended or applied in the future.

7.1. Contributions and implications

In this section we summarize our contributions and we explain how these contributions potentially affect research on planning under uncertainty.

7.1.1. Exact value iteration for POMDPs

In Chapter 3 we presented two algorithmic improvements which create the fastest pruning-based exact value iteration algorithm for solving Partially Observable Markov Decision Processes. We specifically focused on reducing the running time that is required to solve the linear programs in the vector pruning subroutine, which represent a major part of the running time of exact value iteration. We first presented a constraint generation procedure which adds constraints incrementally, in such a way that only a few constraints need to be considered explicitly by the linear programming solver. We further improved this procedure by bootstrapping binding constraints from similar linear programs that were solved in previous iterations of value iteration. Both algorithmic improvements significantly reduce the running time required to solve linear programs, and the resulting variant of exact value iteration outperforms the existing state of the art. Our techniques enhance the efficiency of exact value iteration, but computing optimal solutions remains problematic in practice due to the exponential growth of the number of vectors, and in general computing optimal POMDP solutions does not become feasible. We want to emphasize, however, that our contributions affect the research field in several different ways. For example, our research has shown that linear programs should not be treated as black-box optimization problems. For other researchers it can be beneficial to investigate how existing linear programs in planning algorithms are solved given the structure of the formulation, rather than focusing on new formulations. Furthermore, it may incentivize others to accelerate other exact or approximate algorithms which rely on linear programming. Finally, our algorithm is currently the fastest variant of exact value iteration, which means that we have introduced a new baseline algorithm to compare against when developing new exact POMDP algorithms in the future.

7.1.2. Finite-horizon planning for POMDPs

In Chapter 4 we focused on approximate solution methods for solving finite-horizon POMDPs. A significant body of research in the artificial intelligence community has focused on approximate algorithms for solving infinite-horizon problems, but we provided an extensive discussion which has shown that these algorithms are not suitable for solving finite-horizon problems while providing guarantees on solution quality. We addressed this by introducing FiVI, which is a finite-horizon POMDP algorithm which unifies multiple ideas from existing algorithms for infinite-horizon problems. We also presented two additional strategies which enhance the efficiency of the resulting algorithm. FiVI is the first approximate algorithm that converges to an optimal finite-horizon POMDP solution in the limit while providing value upper bounds during execution. FiVI does not include discounting and it does not require casting finite-horizon problems as infinite-horizon problems, which makes FiVI an attractive algorithm for solving finite-horizon problems. From a more general point of view, we hope that the planning community starts to focus more on tailored algorithms such as FiVI when solving finite-horizon problems.

7.1.3. Approximate algorithm for Constrained POMDPs

In Chapter 5 we presented a new class of solution algorithms for solving Constrained POMDPs. In contrast to existing Constrained POMDP literature, we have shown how Constrained POMDPs can be solved as a sequence of unconstrained

POMDPs. For this purpose we have introduced a column generation procedure, which incrementally generates new columns by solving a series of unconstrained POMDPs using approximate methods. The resulting algorithm can be seen as a new approximate Constrained POMDP algorithm for single-agent problems, but also for multi-agent problems in which multiple independent agents share a global constraint. The algorithm has shown to outperform the existing state of the art, which means that it is currently the most attractive approach for solving Constrained POMDPs. From a more general perspective we believe that the underlying principles of our algorithm can be useful for other constrained planning problems in the research field. For example, we derived the algorithm from an initial linear program that provides a high-level formalization of the optimization problem, and it may be possible to apply the same line of reasoning in other settings as well. Finally, our column generation approach can be seen as the start of a new research line in the general area of Constrained MDPs and Constrained POMDPs, in which various algorithm and model extensions can be developed.

7.1.4. Constrained planning in smart distribution grids

In Chapter 6 we focused on an application in which constrained multi-agent planning under uncertainty can be used to solve planning problems with significant societal impact. In particular, we established a connection between constrained planning under uncertainty and the development of smart distribution grids. In such distribution grids power consumption of flexible loads can be shifted temporally in order to prevent grid congestion. We presented a planning-based approach for congestion management, in which we formalized the congestion management problem as a Constrained Multi-agent Markov Decision Process. Furthermore, we described additional techniques to ensure that the agents respect power grid constraints during policy execution. From an artificial intelligence perspective these techniques can be seen as a new approach to compute conditional preallocations of finite resources to multiple agents, in which the actual usage of resources depends on the realization of the uncertainty. This is an attractive alternative to existing deterministic preallocation methods, which tend to be more conservative since the uncertainty is not explicitly considered during allocation. Besides the implications for the field of artificial intelligence, our techniques also affect future research and development in the area of smart distribution grids. The connection between smart distribution grids and constrained planning for Markov Decision Processes has not been made before, and therefore our work may accelerate the integration of artificial intelligence methods in smart congestion management methods for distribution grids in the real world.

7.2. Directions for future work

There are multiple directions of future research that can be pursued in the future, building upon the algorithms presented in this dissertation. Furthermore, there are several opportunities to apply our ideas while developing intelligent decision making systems in the real world. In the remainder of this section we describe these directions and ideas in more detail.

Incremental constraint generation and bootstrapping has shown to be an effective technique to improve the performance of exact value iteration algorithms for POMDPs. Our techniques are general in the sense that no assumptions are made about the actual vectors that are part of the value function, which means that domain-specific characteristics are ignored. In the future it can be investigated whether information about the POMDP domain under consideration can be used to further improve pruning methods. This does not only apply to our pruning techniques, because other pruning methods may also benefit from such domain-specific details. Although our techniques have shown to be effective for exact POMDP planning, we believe that the most promising future work directions can be found in the area of approximate planning and other types of decision making algorithms based on vectors. For example, our ideas may be used to accelerate approximate POMDP algorithms (Varakantham et al., 2007), and our preliminary results for Multi-Objective MDPs (Roijers, Walraven, and Spaan, 2018) indicate that our techniques are applicable beyond POMDP planning as well. Both directions can be further investigated and expanded in the future.

In this dissertation we considered two types of constrained planning problems which involve multiple agents. The first type focuses on planning with partial observability of states, and the second type has a more applied nature and focuses on enforcing strict resource constraints in the context of Constrained MDP planning. Since both types have several interesting connections, we provide a combined discussion which indicates how research in this area can proceed and how both areas can be connected.

Our approximate algorithm for Constrained POMDPs relies on a master linear program which imposes constraints directly in the policy space. The size of this linear program does not introduce a bottleneck at the moment, but it may be interesting to investigate whether alternatives exist to replace this linear program. For example, the master linear program is currently the only part which requires a centralized optimization step. Since all agents operate in a distributed fashion, it may be worth investigating whether a distributed optimization scheme can be used to create a fully-distributed algorithm that enables coordination between the agents while computing a series of policies. Although not reported in this dissertation, we did preliminary experiments with a distributed version of the simplex algorithm, which splits the simplex tableau into multiple parts and the agents store and update these parts of the tableau separately. This creates a fully-distributed algorithm, but currently it has limited practical value because it is still more efficient if there is just one agent that solves the entire master linear program and broadcasts the solution to the agents. In future work it can be studied whether more efficient solutions exist which exploit the specific structure of the master linear program.

A second research direction focusing on the linear programming step involves kernelization of the constrained optimization problem based on heuristic information about the domain and potential congestion. Such an approach would reduce the size of the optimization problem in a preprocessing stage, such that the solution to the smaller problem can be used for the larger problem. For example, the linear program used for Constrained MDPs may grow large due to grid constraints and the number of states and actions of the agents, while it may only be relevant to consider congestion for a few lines and specific time steps. Smart reduction of the model size prior to solving would improve the scalability of our approach.

Our constrained planning algorithms have been presented and evaluated in isolation, but it is interesting to investigate whether the ideas from Chapter 5 and Chapter 6 can be combined, such that our planning algorithm for congestion management also considers partial observability. Grid constraints can be integrated directly in the master linear program used for Constrained POMDPs, but since POMDPs are more computationally more demanding we expect that the existing approach combined with a large number of constraints does not provide sufficient scalability for real-world instances. Addressing these scalability problems would be an open problem for future work. It also needs to be investigated under which circumstances modeling of partial observability is necessary while managing congestion in distribution grids.

Another important next step is additional evaluation of our congestion management methods in a more realistic setting. Our current evaluation is based on a realistic mathematical model describing power flows in grids, which is sufficient to study power flows and congestion. However, additional evaluation in simulators such as GridLAB-D would provide more insight into the potential benefits and impact of our algorithm in the context of development of smart grid technology.

The research directions that we have described can be seen as direct extensions of the work presented in this dissertation. We think that these directions are important for extending and improving our algorithms, but from a more general point of view there are more significant obstacles which need to be addressed in order to bring algorithms for sequential decision making under uncertainty to the real world. As an example we consider our algorithms for congestion management in smart grids. The current state of affairs is that theoretical algorithms have shown to be able to solve congestion management problems based on a given model in synthetic simulations. However, as discussed in Chapter 1, there is an inherent mismatch between mathematical models used for planning and the characteristics of the real world, which subsequently leads to model uncertainty. Theoretical guarantees and computational results obtained in planning research do not necessarily apply in the real world due to this additional source of uncertainty, which raises the question whether and how new planning techniques can be used in the real world.

We think that there are two key visionary research directions which would finally lead to large-scale application of planning under uncertainty. First, it is necessary to establish significant understanding of model uncertainty in the context of MDPs and POMDPs, such that it becomes known how it affects algorithm characteristics. This knowledge is required for plain MDP and POMDP algorithms, but also for more sophisticated techniques such as the algorithms presented in this dissertation. This is particularly relevant when grid constraints need to be respected at all times, because model uncertainty may cause situations in which constraints are still violated. Besides studying model uncertainty it is required to use this understanding to develop robust planning systems that are aware of this type of uncertainty while computing plans and during execution of plans. Future work may build upon initial results for POMDPs with imprecise parameters (Itoh and Nakamura, 2007), as well as reinforcement learning techniques which respect constraints while learning in an unknown environment (García and Fernández, 2015). As a result, in the context of smart distribution grids it would lead to truly AI-based congestion management in the real world. From a more general perspective it makes it easier to build real-world applications based on MDPs and POMDPs.

Bibliography

P. Agrawal, A. Kumar, and P. Varakantham (2015). “Near-optimal decentralized power supply restoration in smart grids”. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 1275–1283.

P. Agrawal, P. Varakantham, and W. Yeoh (2016). “Scalable Greedy Algorithms For Task / Resource Constrained Multi-Agent Stochastic Planning”. In: *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 10–16.

A. Alan, E. Costanza, J. Fischer, S. D. Ramchurn, T. Rodden, and N. R. Jennings (2014). “A field study of human-agent interaction for electricity tariff switching”. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 965–972.

E. Altman (1999). *Constrained Markov Decision Processes*. CRC Press.

C. Amato, D. S. Bernstein, and S. Zilberstein (2010). “Optimizing fixed-size stochastic controllers for POMDPs and decentralized POMDPs”. In: *Journal of Autonomous Agents and Multi-Agent Systems* 21.3, pp. 293–320.

D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané (2016). “Concrete Problems in AI Safety”. In: *arXiv preprint arXiv:1606.06565*.

M. Andoni and V. Robu (2016). “Game-theoretic Modeling of Transmission Line Reinforcements with Distributed Generation”. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 1291–1292.

A. Angelidakis and G. Chalkiadakis (2015). “Factored MDPs for optimal prosumer decision-making”. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 503–511.

S. Bandyopadhyay, P. Kumar, and V. Arya (2016). “Planning Curtailment of Renewable Generation in Power Grids”. In: *Proceedings of the International Conference on Automated Planning and Scheduling*, pp. 353–357.

S. Bandyopadhyay, R. Narayanam, R. Kota, M. I. Petra, and Z. Charbiwala (2015). “Aggregate Demand-Based Real-Time Pricing Mechanism for the Smart Grid: A Game-Theoretic Analysis”. In: *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 2554–2560.

S. Bandyopadhyay, R. Narayanam, P. Kumar, S. D. Ramchurn, V. Arya, and M. I. Petra (2016). “An Axiomatic Framework for Ex-Ante Dynamic Pricing Mechanisms in Smart Grid”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 3800–3806.

L. Barrett and S. Narayanan (2008). “Learning all optimal policies with multiple criteria”. In: *Proceedings of the International Conference on Machine Learning*, pp. 41–47.

R. Bellman (1957). *Dynamic programming*. Princeton University Press.

J. F. Benders (1962). “Partitioning procedures for solving mixed-variables programming problems”. In: *Numerische Mathematik* 4.1, pp. 238–252.

H. Berlink and A. H. Costa (2015). “Batch Reinforcement Learning for Smart Home Energy Management”. In: *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 2561–2567.

F. Bizzarri and D. Mucci (2018). “Day-Ahead Hourly Forecasting of Power Generation From Photovoltaic Plants”. In: *IEEE Transactions on Sustainable Energy* 9.2, pp. 831–842.

B. Bonet and H. Geffner (2009). “Solving POMDPs: RTDP-Bel vs. Point-based Algorithms”. In: *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 1641–1646.

C. Boutilier (1996). “Planning, Learning and Coordination in Multiagent Decision Processes”. In: *Proceedings of the Conference on Theoretical Aspects of Rationality and Knowledge*, pp. 195–210.

C. Boutilier and T. Lu (2016). “Budget Allocation using Weakly Coupled, Constrained Markov Decision Processes”. In: *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pp. 52–61.

S. Bradley, A. Hax, and T. Magnanti (1977). *Applied Mathematical Programming*. Addison Wesley.

D. Braziunas and C. Boutilier (2004). “Stochastic Local Search for POMDP Controllers”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 690–696.

C. Breyer, D. Bogdanov, A. Gulagi, A. Aghahosseini, L. S. Barbosa, O. Koskinen, M. Barasa, U. Caldera, S. Afanasyeva, M. Child, J. Farfan, and P. Vainikka (2017). “On the role of solar photovoltaics in global energy transition scenarios”. In: *Progress in Photovoltaics: Research and Applications* 25.8, pp. 727–745.

M. A. Burgess, A. C. Chapman, and P. Scott (2018). “The Generalized N&K Value: An Axiomatic Mechanism for Electricity Trading”. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 1883–1885.

E. Byon and Y. Ding (2010). “Season-Dependent Condition-Based Maintenance for a Wind Turbine Using a Partially Observed Markov Decision Process”. In: *IEEE Transactions on Power Systems* 25.4, pp. 1823–1834.

A. R. Cassandra, M. L. Littman, and N. L. Zhang (1997). “Incremental Pruning: A Simple, Fast, Exact Method for Partially Observable Markov Decision Processes”. In: *Proceedings of the Conference on Uncertainty in Artificial Intelligence*.

A. R. Cassandra (1998). “Exact and approximate algorithms for partially observable Markov decision processes”. PhD thesis. Brown University.

J. Cerquides, G. Picard, and J. A. Rodríguez-Aguilar (2015). “Designing a Marketplace for the Trading and Distribution of Energy in the Smart Grid”. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 1285–1293.

G. Chalkiadakis, V. Robu, R. Kota, A. Rogers, and N. R. Jennings (2011). “Cooperatives of distributed energy resources for efficient virtual power plants”. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 787–794.

C. Chau, K. Elbassioni, and M. Khonji (2014). “Truthful mechanisms for combinatorial AC electric power allocation”. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 1005–1012.

N. Chen, Z. Qian, I. T. Nabney, and X. Meng (2013). “Short-Term Wind Power Forecasting Using Gaussian Processes”. In: *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 2790–2796.

P. Chen, S. Liu, C. Shi, B. Hooi, B. Wang, and X. Cheng (2018). “NeuCast: Seasonal Neural Forecast of Power Grid Time Series”. In: *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 3315–3321.

H. Cheng (1988). “Algorithms for Partially Observable Markov Decision Processes”. PhD thesis. University of British Columbia.

M. M. P. Chowdhury (2016). “Predicting Prices in the Power TAC Wholesale Energy Market”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 4204–4205.

C. Coffrin, H. L. Hijazi, and P. Van Hentenryck (2016). “The QC Relaxation: A Theoretical and Computational Study on Optimal Power Flow”. In: *IEEE Transactions on Power Systems* 31.4, pp. 3008–3018.

C. Coffrin and P. Van Hentenryck (2014). “A Linear-Programming Approximation of AC Power Flows”. In: *INFORMS Journal on Computing* 26.4, pp. 718–734.

K. Coninx and T. Holvoet (2016). “Coordinating wind turbines and flexible consumers with cooperative and competitive agents”. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 1407–1408.

R. H. Crites (1996). “Large-scale dynamic optimization using teams of reinforcement learning agents”. PhD thesis. University of Massachusetts Amherst.

G. Dantzig (1963). *Linear programming and extensions*. Princeton University Press.

B. Defourny, D. Ernst, and L. Wehenkel (2012). “Multistage Stochastic Programming: A Scenario Tree Based Approach to Planning under Uncertainty”. In: *Decision Theory Models for Applications in Artificial Intelligence: Concepts and Solutions*. Ed. by L. E. Sucar, E. F. Morales, and J. Hoey. Information Science Publishing, pp. 97–143.

J. Desrosiers and M. E. Lübbecke (2005). *A primer in column generation*. Kluwer.

D. Dolgov and E. H. Durfee (2003). “Approximating Optimal Policies for Agents with Limited Execution Resources”. In: *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 1107–1112.

Y. Dujardin, T. Dietterich, and I. Chadès (2015). “α-min: A Compact Approximate Solver For Finite-Horizon POMDPs”. In: *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 2582–2588.

Y. Dujardin, T. Dietterich, and I. Chadès (2017). “Three New Algorithms to Solve N-POMDPs”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 4495–4501.

H. Ellis, M. Jiang, and R. B. Corotis (1995). “Inspection, maintenance, and repair with partial observability”. In: *Journal of Infrastructure Systems* 1.2, pp. 92–99.

S. Fararooy and J. Allan (1995). “Condition-based maintenance of railway signalling equipment”. In: *Proceedings of the International Conference on Electric Railways in a United Europe*, pp. 33–37.

Z. Feng and S. Zilberstein (2004). “Region-Based Incremental Pruning for POMDPs”. In: *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pp. 146–153.

F. Fioretto, W. Yeoh, and E. Pontelli (2017). "A Multiagent System Approach to Scheduling Devices in Smart Homes". In: *Proceedings of the Conference on Autonomous Agents and Multiagent Systems*, pp. 981–989.

M. Fox, D. Long, and D. Magazzeni (2011). "Automatic Construction of Efficient Multiple Battery Usage Policies". In: *Proceedings of the International Conference on Automated Planning and Scheduling*, pp. 74–81.

J. García and F. Fernández (2015). "A comprehensive survey on safe reinforcement learning". In: *Journal of Machine Learning Research* 16.1, pp. 1437–1480.

E. Gerding, S. Stein, S. Ceppi, and V. Robu (2016). "Online Mechanism Design for Vehicle-to-Grid Car Parks". In: *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 286–293.

E. H. Gerding, V. Robu, S. Stein, D. C. Parkes, A. Rogers, and N. R. Jennings (2011). "Online mechanism design for electric vehicle charging". In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 811–818.

P. C. Gilmore and R. E. Gomory (1961). "A Linear Programming Approach to the Cutting-Stock Problem". In: *Operations Research* 9.6, pp. 849–859.

J. Grau-Moya, F. Leibfried, T. Genewein, and D. A. Braun (2016). "Planning with Information Processing Constraints and Model Uncertainty in Markov Decision Processes". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, pp. 475–491.

M. Grześ, P. Poupart, and J. Hoey (2013). "Isomorph-Free Branch and Bound Search for Finite State Controllers". In: *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 2282–2290.

M. Grześ, P. Poupart, X. Yang, and J. Hoey (2015). "Energy Efficient Execution of POMDP Policies". In: *IEEE Transactions on Cybernetics* 45.11, pp. 2484–2497.

C. Guestrin and G. Gordon (2002). "Distributed Planning in Hierarchical Factored MDPs". In: *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pp. 197–206.

C. Guestrin, D. Koller, R. Parr, and S. Venkataraman (2003). "Efficient Solution Algorithms for Factored MDPs". In: *Journal of Artificial Intelligence Research* 19.c, pp. 399–468.

E. A. Hansen (1998). "Solving POMDPs by Searching in Policy Space". In: *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pp. 211–219.

E. A. Hansen (2007). “Indefinite-Horizon POMDPs with Action-Based Termination”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 291–296.

X. Hao, B. Tang, and Y. Wang (2015). “On the Balance of Meter Deployment Cost and NILM Accuracy”. In: *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 2603–2609.

R. M. Harman (2002). “Wireless solutions for aircraft condition based maintenance systems”. In: *Proceedings of the IEEE Aerospace Conference*, pp. 6–2877 –6–2886.

M. Hauskrecht (2000). “Value-Function Approximations for Partially Observable Markov Decision Processes”. In: *Journal of Artificial Intelligence Research* 13, pp. 33–94.

K. Hayakawa, E. Gerding, S. Stein, and T. Shiga (2015). “Online mechanisms for charging electric vehicles in settings with varying marginal electricity costs”. In: *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 2610–2616.

P. Hernandez-Leal, M. E. Taylor, E. Munoz de Cote, and L. E. Sucar (2015). “Bidding in Non-Stationary Energy Markets”. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 1709–1710.

H. Hijazi, C. Coffrin, and P. Van Hentenryck (2016). “Polynomial SDP Cuts for Optimal Power Flow”. In: *Power Systems Computation Conference*. IEEE.

K. Hiraoka, M. Yoshida, and T. Mishima (2009). “Parallel reinforcement learning for weighted multi-criteria model with adaptive margin”. In: *Cognitive Neurodynamics* 3, pp. 17–24.

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

J. D. Isom, S. P. Meyn, and R. D. Braatz (2008). “Piecewise Linear Dynamic Programming for Constrained POMDPs”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 291–296.

H. Itoh and K. Nakamura (2007). “Partially observable Markov decision processes with imprecise parameters”. In: *Artificial Intelligence* 171.8, pp. 453 –490.

M. Jain, E. Kardes, C. Kiekintveld, F. Ordónez, and M. Tambe (2010). “Security Games with Arbitrary Schedules: A Branch and Price Approach”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 792–797.

S. Jain, B. Narayanaswamy, and Y. Narahari (2014). “A Multiarmed Bandit Incentive Mechanism for Crowdsourcing Demand Response in Smart Grids”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 721–727.

A. K. Jardine, D. Lin, and D. Banjevic (2006). “A review on machinery diagnostics and prognostics implementing condition-based maintenance”. In: *Mechanical Systems and Signal Processing* 20.7, pp. 1483–1510.

L. P. Kaelbling (1993). *Learning in Embedded Systems*. MIT press.

L. P. Kaelbling, M. L. Littman, and A. R. Cassandra (1998). “Planning and acting in partially observable stochastic domains”. In: *Artificial intelligence* 101.1, pp. 99–134.

M. Kahlen and W. Ketter (2015). “Aggregating Electric Cars to Sustainable Virtual Power Plants: The Value of Flexibility in Future Electricity Markets”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 665–671.

M. Kahlen, W. Ketter, and J. Van Dalen (2014). “Agent-coordinated virtual power plants of electric vehicles”. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 1547–1548.

S. Kamboj, W. Kempton, and K. S. Decker (2011). “Deploying power grid-integrated electric vehicles as a multi-agent system”. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 13–20.

N. Karmarkar (1984). “A new polynomial-time algorithm for linear programming”. In: *Proceedings of the Annual ACM symposium on Theory of Computing*. ACM, pp. 302–311.

W. Ketter, M. Peters, and J. Collins (2013). “Autonomous Agents in Future Energy Markets: The 2012 Power Trading Agent Competition”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 1298–1304.

L. Khachiyan (1980). “Polynomial algorithms in linear programming”. In: *USSR Computational Mathematics and Mathematical Physics* 20.1, pp. 53 –72.

Y. B. Khoo, C. Wang, P. Paevere, and A. Higgins (2014). “Statistical modeling of Electric Vehicle electricity consumption in the Victorian EV Trial, Australia”. In: *Transportation Research Part D: Transport and Environment* 32, pp. 263–277.

D. Kim, J. Lee, K. Kim, and P. Poupart (2011). “Point-Based Value Iteration for Constrained POMDPs”. In: *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 1968–1974.

J. W. Kim, G. B. Choi, and J. M. Lee (2018). “A POMDP framework for integrated scheduling of infrastructure maintenance and inspection”. In: *Computers & Chemical Engineering* 112, pp. 239–252.

J. Koehler and D. Ottiger (2002). “An AI-based approach to destination control in elevators”. In: *AI magazine* 23.3, p. 59.

A. Kumar and S. Zilberstein (2015). “History-Based Controller Design and Optimization for Partially Observable MDPs”. In: *Proceedings of the International Conference on Automated Planning and Scheduling*, pp. 156–164.

S. R. Kuppannagari, R. Kannan, C. Chelmis, and V. K. Prasanna (2016). “Implementation of Learning-Based Dynamic Demand Response on a Campus Micro-Grid”. In: *Proceedings of the International Joint Conference on Artificial Intelligence*,
pp. 4250–4251.

H. Kurniawati, D. Hsu, and W. S. Lee (2008). “SARSOP : Efficient Point-Based POMDP Planning by Approximating Optimally Reachable Belief Spaces”. In: *Proceedings of Robotics: Science and Systems IV*.

J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt, M. Sokolsky, G. Stanek, D. Stavens, A. Teichman, M. Werling, and S. Thrun (2011). “Towards Fully Autonomous Driving: Systems and Algorithms”. In: *IEEE Intelligent Vehicles Symposium (IV)*. IEEE, pp. 163–168.

M. L. Littman (1994). “Markov games as a framework for multi-agent reinforcement learning”. In: *Proceedings of the International Conference on Machine Learning*, pp. 157–163.

M. L. Littman (1996). “Algorithms for Sequential Decision Making”. PhD thesis. Brown University.

M. Lunde, I. Røpke, and E. Heiskanen (2016). “Smart grid: hope or hype?” In: *Energy Efficiency* 9.2, pp. 545–562.

H. Ma, D. C. Parkes, and V. Robu (2017). “Generalizing demand response through reward bidding”. In: *Proceedings of the Conference on Autonomous Agents and Multiagent Systems*, pp. 60–68.

H. Ma, V. Robu, N. Li, and D. C. Parkes (2016). “Incentivizing Reliability in Demand-Side Response”. In: *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 352–358.

A. Marinescu, I. Dusparic, A. Taylor, V. Cahill, and S. Clarke (2015). “P-MARL: Prediction based Multi-Agent Reinforcement Learning for non-stationary environments”. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 1897–1898.

T. H. Mattheiss (1973). “An Algorithm for Determining Irrelevant Constraints and all Vertices in Systems of Linear Inequalities”. In: *Operations Research* 21.1, pp. 247–260.

D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins (1998). *PDDL – The Planning Domain Definition Language*.

G. Methenitis, M. Kaisers, and H. La Poutré (2016). "Incentivizing Intelligent Customer Behavior in Smart-Grids: A Risk-Sharing Tariff & Optimal Strategies". In: *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 380–386.

G. Methenitis, M. Kaisers, and H. La Poutré (2017). "SLA-Mechanisms for Electricity Trading under Volatile Supply and Varying Criticality of Demand". In: *Proceedings of the Conference on Autonomous Agents and Multiagent Systems*, pp. 1640–1642.

S. Miller, S. D. Ramchurn, and A. Rogers (2012). "Optimal decentralised dispatch of embedded generation in the smart grid". In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 281–288.

G. E. Monahan (1982). "A survey of partially observable Markov decision processes: theory, models, and algorithms". In: *Management Science* 28.1.

G. Morales-España, A. Lorca, and M. M. de Weerdt (2018). "Robust unit commitment with dispatchable wind power". In: *Electric Power Systems Research* 155, pp. 58–66.

G. Morales-España, R. Baldick, J. García-González, and A. Ramos (2016). "Power-Capacity and Ramp-Capability Reserves for Wind Integration in Power-Based UC". In: *IEEE Transactions on Sustainable Energy* 7.2, pp. 614–624.

J. Mrkos, A. Komenda, and M. Jakob (2018). "Revenue Maximization for Electric Vehicle Charging Service Providers Using Sequential Dynamic Pricing". In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 832–840.

L. C. Neves and D. M. Frangopol (2005). "Condition, safety and cost profiles for deteriorating structures with emphasis on bridges". In: *Reliability Engineering & System Safety* 89.2, pp. 185 –198.

F. de Nijs, M. T. J. Spaan, and M. M. de Weerdt (2015). "Best-Response Planning of Thermostatically Controlled Loads under Power Constraints". In: *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 615–621.

F. de Nijs, M. T. J. Spaan, and M. M. de Weerdt (2018). "Preallocation and Planning under Stochastic Resource Constraints". In: *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 4662–4669.

F. de Nijs, E. Walraven, M. M. de Weerdt, and M. T. J. Spaan (2017). “Bounding the Probability of Resource Constraint Violations in Multi-Agent MDPs”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 3562–3568.

F. de Nijs, G. Theocharous, N. Vlassis, M. M. de Weerdt, and M. T. J. Spaan (2018). “Capacity-aware Sequential Recommendations”. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*.

F. A. Oliehoek and C. Amato (2016). *A Concise Introduction to Decentralized POMDPs*. Springer.

N. P. Padhy (2004). “Unit commitment-a bibliographical survey”. In: *IEEE Transactions on Power Systems* 19.2, pp. 1196–1205.

Y. Pang, B. Yao, X. Zhou, Y. Zhang, Y. Xu, and Z. Tan (2018). “Hierarchical Electricity Time Series Forecasting for Integrating Consumption Patterns Analysis and Aggregation Consistency”. In: *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 3506–3512.

C. H. Papadimitriou and K. Steiglitz (1982). *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall.

O. Parson, S. Ghosh, M. J. Weal, and A. Rogers (2012). “Non-Intrusive Load Monitoring Using Prior Models of General Appliance Types”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 356–362.

A. Perez-Diaz, E. Gerding, and F. McGroarty (2018). “Coordination of Electric Vehicle Aggregators: A Coalitional Approach”. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 676–684.

A. Perrault and C. Boutilier (2015). “Approximately Stable Pricing for Coordinated Purchasing of Electricity”. In: *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 2624–2631.

C. Piacentini, V. Alimisis, M. Fox, and D. Long (2013). “Combining a Temporal Planner with an External Solver for the Power Balancing Problem in an Electricity Network”. In: *Proceedings of the International Conference on Automated Planning and Scheduling*, pp. 398–406.

J. Pineau, G. Gordon, and S. Thrun (2003). “Point-based value iteration: An anytime algorithm for POMDPs”. In: *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 1025–1030.

P. Poupart and C. Boutilier (2003). “Bounded Finite State Controllers”. In: *Advances in Neural Information Processing Systems*, pp. 823–830.

P. Poupart, K. Kim, and D. Kim (2011). “Closing the Gap: Improved Bounds on Optimal POMDP Solutions”. In: *Proceedings of the International Conference on Automated Planning and Scheduling*, pp. 194–201.

P. Poupart, A. Malhotra, P. Pei, K. Kim, B. Goh, and M. Bowling (2015). “Approximate Linear Programming for Constrained Partially Observable Markov Decision Processes”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 3342–3348.

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

S. D. Ramchurn, P. Vytelingum, A. Rogers, and N. R. Jennings (2011). “Agent-Based Control for Decentralised Demand Side Management in the Smart Grid”. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 5–12.

S. D. Ramchurn, P. Vytelingum, A. Rogers, and N. R. Jennings (2012). “Putting the ’Smarts’ into the Smart Grid: A Grand Challenge for Artificial Intelligence”. In: *Communications of the ACM* 55.4, pp. 86–97.

C. Raphael and G. Shani (2012). “The Skyline algorithm for POMDP value function pruning”. In: *Annals of Mathematics and Artificial Intelligence* 65.1, pp. 61–77.

P. P. Reddy and M. M. Veloso (2011a). “Learned Behaviors of Multiple Autonomous Agents in Smart Grid Markets”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 1396–1401.

P. P. Reddy and M. M. Veloso (2011b). “Strategy learning for autonomous agents in smart grid markets”. In: *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 1446–1451.

P. P. Reddy and M. M. Veloso (2012). “Factored Models for Multiscale Decision-Making in Smart Grid Customers”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 363–369.

P. P. Reddy and M. M. Veloso (2013). “Negotiated Learning for Smart Grid Agents: Entity Selection based on Dynamic Partially Observable Features”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 1313–1319.

D. M. Roijers, E. Walraven, and M. T. J. Spaan (2018). “Bootstrapping LPs in Value Iteration for Multi-Objective and Partially Observable MDPs”. In: *Proceedings of the International Conference on Automated Planning and Scheduling*, pp. 218–226.

D. M. Roijers, P. Vamplew, S. Whiteson, and R. Dazeley (2013). “A Survey of Multi-Objective Sequential Decision-Making”. In: *Journal of Artificial Intelligence Research* 48, pp. 67–113.

H. Rose, A. Rogers, and E. H. Gerding (2012). “A Scoring Rule-Based Mechanism for Aggregate Demand Prediction in the Smart Grid”. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 661–668.

S. Ross, J. Pineau, and B. Chaib-Draa (2008). “Theoretical Analysis of Heuristic Search Methods for Online POMDPs”. In: *Advances in Neural Information Processing Systems*. Vol. 20, pp. 1216–1225.

N. Sadeghianpourhamami, N. Refa, M. Strobbe, and C. Develder (2018). “Quantitive analysis of electric vehicle flexibility: A data-driven approach”. In: *International Journal of Electrical Power & Energy Systems* 95, pp. 451–462.

P. Schavemaker and L. van der Sluis (2008). *Electrical power system essentials*. John Wiley & Sons.

K. Schneider, B. Mather, B. C. Pal, C. Ten, G. Shirek, H. Zhu, J. Fuller, J. L. R. Pereira, L. Ochoa, L. De Araujo, R. Dugan, S Matthias, S Paudyal, T. McDermott, and W. Kersting (2018). “Analytic Considerations and Design Basis for the IEEE Distribution Test Feeders”. In: *IEEE Transactions on Power Systems* 33.3, pp. 3181–3188.

G. Shani, J. Pineau, and R. Kaplow (2013). “A survey of point-based POMDP solvers”. In: *Journal of Autonomous Agents and Multi-Agent Systems* 27.1, pp. 1–51.

M. Shann and S. Seuken (2013). “An Active Learning Approach to Home Heating in the Smart Grid”. In: *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 2892–2899.

M. Shann and S. Seuken (2014). “Adaptive home heating under weather and price uncertainty using GPS and MDPs”. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 821–828.

T. Smith (2007). “Probabilistic Planning for Robotic Exploration”. PhD thesis. Pittsburgh, PA: The Robotics Institute, Carnegie Mellon University.

T. Smith and R. Simmons (2005). “Point-Based POMDP Algorithms: Improved Analysis and Implementation”. In: *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pp. 542–549.

T. Smith and R. Simmons (2006). “Focused Real-Time Dynamic Programming for MDPs: Squeezing More Out of a Heuristic”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 1227–1232.

E. J. Sondik (1971). “The optimal control of partially observable Markov processes.” PhD thesis. Stanford University.

M. T. J. Spaan (2012). “Partially Observable Markov Decision Processes”. In: *Reinforcement Learning – State-of-the-Art*. Ed. by M. Wiering and M. van Otterlo. Springer, pp. 387–414.

M. T. J. Spaan and N. Vlassis (2005). “Perseus: Randomized Point-based Value Iteration for POMDPs”. In: *Journal of Artificial Intelligence Research* 24, pp. 195–220.

S. Stein, E. H. Gerding, A. Nedea, A. Rosenfeld, and N. R. Jennings (2016). “Bid2Charge: Market user interface design for electric vehicle charging”. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 882–890.

B. Stott, J. Jardim, and O. Alsac (2009). “DC Power Flow Revisited”. In: *IEEE Transactions on Power Systems* 24.3, pp. 1290–1300.

D. Strawser, B. Williams, and W. Inam (2015). “A Market for Reliability for Electricity Scheduling in Developing World Microgrids”. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 1833–1834.

A. D. Styler and I. R. Nourbakhsh (2015). “Real-Time Predictive Optimization for Energy Management in a Hybrid Electric Vehicle”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 737–744.

O. Sundstrom and C. Binding (2012). “Flexible Charging Optimization for Electric Vehicles Considering Distribution Grid Constraints”. In: *IEEE Transactions on Smart Grid* 3.1, pp. 26–37.

F. W. Trevizan, S. Thiébaux, P. H. Santana, and B. C. Williams (2016). “Heuristic Search in Dual Space for Constrained Stochastic Shortest Path Problems”. In: *Proceedings of the International Conference on Automated Planning and Scheduling*,
pp. 326–334.

D. Urieli and P. Stone (2014). “TacTex’13: a champion adaptive power trading agent”. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 1447–1448.

D. Urieli and P. Stone (2016). “An MDP-based winning approach to autonomous power trading: formalization and empirical analysis”. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 827–835.

K. Valogianni, W. Ketter, and J. Collins (2014). “Learning to schedule electric vehicle charging given individual customer preferences”. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent systems*, pp. 1591–1592.

K. Valogianni, W. Ketter, and J. Collins (2015). “A Multiagent Approach to Variable-Rate Electric Vehicle Charging Coordination”. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 1131–1139.

M. Valovage, A. Shekhawat, and M. L. Gini (2018). “Model-Free Iterative Temporal Appliance Discovery for Unsupervised Electricity Disaggregation”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 2484–2491.

M. Van Den Briel, P. Scott, and S. Thiébaux (2013). “Randomized Load Control: A Simple Distributed Approach for Scheduling Smart Appliances”. In: *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 2915–2922.

S. Vandael, N. Boucké, T. Holvoet, K. De Craemer, and G. Deconinck (2011). “Decentralized coordination of plug-in hybrid vehicles for imbalance reduction in a smart grid”. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 803–810.

P. R. Varakantham, R. Maheswaran, T. Gupta, and M. Tambe (2007). “Towards efficient computation of quality bounded solutions in POMDPs: Expected Value Approximation and Dynamic Disjunctive Beliefs”. In: *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 2638–2643.

M. Vasirani, S. Ossowski, R. Kota, R. L. Cavalcante, and N. R. Jennings (2011). “Using coalitions of wind generators and electric vehicles for effective energy market participation”. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 1099–1100.

E. Veldman and R. A. Verzijlbergh (2015). “Distribution Grid Impacts of Smart Electric Vehicle Charging From Different Perspectives”. In: *IEEE Transactions on Smart Grid* 6.1, pp. 333–342.

T. Voice, P. Vytelingum, S. D. Ramchurn, A. Rogers, and N. R. Jennings (2011). “Decentralised Control of Micro-Storage in the Smart Grid”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 1421–1427.

P. Vytelingum, T. D. Voice, S. D. Ramchurn, A. Rogers, and N. R. Jennings (2010a). “Agent-based micro-storage management for the smart grid”. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 39–46.

P. Vytelingum, S. D. Ramchurn, T. D. Voice, A. Rogers, and N. R. Jennings (2010b). “Trading agents for the smart electricity grid”. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 897–904.

E. Walraven and M. T. J. Spaan (2015). “Planning under Uncertainty with Weighted State Scenarios”. In: *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pp. 912–921.

E. Walraven and M. T. J. Spaan (2016). “Planning Under Uncertainty for Aggregated Electric Vehicle Charging with Renewable Energy Supply”. In: *Proceedings of the European Conference on Artificial Intelligence*, pp. 904–912.

E. Walraven and M. T. J. Spaan (2017). “Accelerated Vector Pruning for Optimal POMDP Solvers”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 3672–3678.

E. Walraven and M. T. J. Spaan (2018). “Column Generation Algorithms for Constrained POMDPs”. In: *Journal of Artificial Intelligence Research* 62, pp. 489–533.

E. Walraven and M. T. J. Spaan (2019). “Point-Based Value Iteration for Finite-Horizon POMDPs”. In: *Journal of Artificial Intelligence Research*.

X. Wang, M. Zhang, and F. Ren (2016). “Load Forecasting in a Smart Grid through Customer Behaviour Learning Using L1-Regularized Continuous Conditional Random Fields”. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 817–826.

M. M. de Weerdt, M. Albert, V. Conitzer, and K. van der Linden (2018). “Complexity of Scheduling Charging in the Smart Grid”. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 1924–1926.

S. Weitemeyer, D. Kleinhans, T. Vogt, and C. Agert (2015). “Integration of Renewable Energy Sources in future power systems: The role of storage”. In: *Renewable Energy* 75, pp. 14–20.

C. C. White (1991). “A survey of solution techniques for the partially observed Markov decision process”. In: *Annals of Operations Research* 32.1, pp. 215–230.

M. A. Wiering, M. Withagen, and M. M. Drugan (2014). “Model-based multi-objective reinforcement learning”. In: *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*.

J. Wu and E. H. Durfee (2010). “Resource-Driven Mission-Phasing Techniques for Constrained Agents in Stochastic Environments”. In: *Journal of Artificial Intelligence Research* 38, pp. 415–473.

Y. Yang, J. Hao, M. Sun, Z. Wang, C. Fan, and G. Strbac (2018). “Recurrent Deep Multiagent Q-Learning for Autonomous Brokers in Smart Grid”. In: *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 569–575.

K. A. Yost and A. R. Washburn (2000). “The LP/POMDP Marriage: Optimization with Imperfect Information”. In: *Naval Research Logistics* 47.8, pp. 607–619.

K. M. Yousef, M. N. Al-Karaki, and A. M. Shatnawi (2010). “Intelligent traffic light flow control system using wireless sensors networks”. In: *Journal of Information Science and Engineering* 26.3, pp. 753–768.

L. Yu and C. Chau (2013). “Complex-demand knapsack problems and incentives in AC power systems”. In: *Proceedings of the International Conference on Autonomous agents and Multiagent Systems*, pp. 973–980.

Z. Zhang, D. Hsu, W. Lee, Z. Lim, and A. Bai (2015). “PLEASE: Palm Leaf Search for POMDPs with Large Observation Spaces”. In: *Proceedings of the 25th International Conference on Automated Planning and Scheduling*, pp. 249–257.

Summary

Developing intelligent decision making systems in the real world requires planning algorithms which are able to deal with sources of uncertainty and constraints. An example can be found in smart distribution grids, in which planning can be used to decide when electric vehicles charge their batteries, such that capacity limits of lines are respected at all times. In this particular example there can be uncertainty in the arrival time and charging demand of vehicles, and constraints follow directly from the capacity limits of the distribution grid to which vehicles are connected. Existing algorithms for planning under uncertainty subject to constraints are currently not suitable for these types of applications, and therefore this dissertation aims improve the applicability of these algorithms by advancing the state of the art in constrained multi-agent planning under uncertainty. To be more specific, the dissertation focuses on 4 open research challenges, which we describe next.

In Chapter 3 we start with exact value iteration algorithms for solving POMDPs. These algorithms are computationally demanding to execute and computing optimal POMDP solutions in a tractable way can be seen as an open problem. In order to improve the performance of exact value iteration algorithms we present a constraint generation procedure which accelerates solving linear programs within iterations of value iteration. Additionally, we describe a bootstrapping procedure which further improves the performance of solving linear programs by identifying similar linear programs solved previously. The resulting algorithm improves the state of the art in exact POMDP planning and it is currently the fastest pruning-based value iteration algorithm that is available.

In Chapter 4 we change the focus to approximate POMDP algorithms for finite-horizon problems. Existing approximate algorithms are typically suitable for solving infinite-horizon POMDPs, and we discuss why these algorithms do not generalize easily to finite-horizon settings. We present the algorithm FiVI, which is a tailored approximate POMDP algorithm for solving finite-horizon problems. This algorithm unifies several ideas from recent infinite-horizon algorithms and it is the first point-based POMDP algorithm for problems with a finite horizon that converges to an optimal solution in the limit. We also present two heuristics which further improve the efficiency of the algorithm.

In the second part of the dissertation we focus on multi-agent planning problems which include both uncertainty and constraints. Planning with constraints for problems with partial observability received only limited attention in the past, and existing algorithms provide limited scalability. In Chapter 5 we present a novel approximate algorithm which is based on the insight that we can solve a constrained planning problem as a sequence of unconstrained planning problems. This principle allows us to derive a scalable algorithm for solving Constrained POMDPs in both single-agent as well as multi-agent settings. Our algorithm outperforms the state of the art and it represents a new class of algorithms for planning under uncertainty in domains with partial observability and constraints.

The final research challenge addressed in Chapter 6 focuses on constrained planning under uncertainty in smart distribution grids and therefore it has a more applied nature. It connects our theoretical contributions to an application domain with significant societal relevance. In particular, we explain that distribution grids become congested in the future, and we show how contrained planning under uncertainty can be used for congestion management. To this end we discuss how realistic power grid constraints can be combined with planning, and we present multiple techniques which ensure that constraints imposed by a distribution grid are respected during plan execution.

The technical contributions presented in this dissertation advance the state of the art in multiple areas of the planning under uncertainty research field. Additionally, our work improves the applicability of these algorithms in real-world problems in our society. As a result, we obtained improved algorithms which bring us closer to truly constrained planning under uncertainty in the real world.

Samenvatting

Het ontwikkelen van intelligente beslissingssystemen in de echte wereld vereist planningsalgoritmen die rekening kunnen houden met bronnen van onzekerheid en beperkingen. Een voorbeeld is de ontwikkeling van slimme energienetwerken, waarin planning kan worden gebruikt om te beslissen wanneer elektrische voertuigen hun batterij opladen, zodanig dat capaciteitslimieten van kabels in het netwerk nooit overschreden worden. In dit specifieke voorbeeld kan er onzekerheid zijn over de aankomsttijd en energievraag van voertuigen, en de beperkingen volgen uit de capaciteitslimieten van het distributienetwerk waar de voertuigen aan verbonden zijn. Bestaande algoritmen voor planning met onzekerheid en beperkingen zijn momenteel niet geschikt voor dergelijke toepassingen. Daarom heeft dit proefschrift als doel om de toepasbaarheid van deze algoritmen te verbeteren door het ontwikkelen van geavanceerde methodes voor planning met onzekerheid en beperkingen in situaties met meerdere agenten. Dit proefschrift richt zich op 4 open uitdagingen, die we hieronder nader zullen beschrijven.

In Hoofdstuk 3 wordt gekeken naar een exacte versie van het value iteration algoritme voor het oplossen van POMDP modellen. Dergelijke algoritmen zijn rekenintensief en het optimaal oplossen van POMDP modellen op een schaalbare manier kan daarom worden gezien als een open probleem. Om de prestaties van exacte value iteration te verbeteren presenteert dit proefschrift een procedure die beperkingen genereert, zodanig dat het oplossen van lineaire programma's binnen value iteration wordt versneld. Daarnaast presenteert dit proefschrift een 'bootstrapping' procedure die de prestaties verder verbetert door soortgelijke lineare programma's te identificeren die in een eerder stadium tijdens de uitvoering van het algoritme reeds zijn opgelost. Het resulterende algoritme verbetert de huidige stand van de techniek op het gebied van POMDP modellen, en momenteel is het de snelste variant van het exacte value iteration algoritme.

In Hoofdstuk 4 bekijken we benaderingsalgoritmen voor POMDP modellen met een eindige horizon. Bestaande benaderingsalgoritmen zijn doorgaans geschikt voor het oplossen van POMDP modellen met een oneindige horizon, en we bespreken waarom deze algoritmen niet makkelijk generaliseren naar situaties met een eindige horizon. We presenteren een algoritme genaamd FiVI, welke gezien kan worden als een op maat gemaakt benaderingsalgoritme voor POMDP modellen met een eindige horizon. Dit algoritme verenigt meerdere ideeën uit recente algoritmen voor problemen met een oneindige horizon, en het kan worden gezien als het eerste punt-gebaseerde benaderingsalgoritme voor problemen met een eindige horizon dat in de limiet convergeert naar optimaliteit. Daarnaast presenteren we twee heuristieken die de efficiëntie van het algoritme verder verbeteren.

In het tweede deel van het proefschrift wordt er gekeken naar planningsproblemen met meerdere agenten, onzekerheid en beperkingen. Planning met beperkingen voor problemen met gedeeltelijke waarneembaarheid heeft tot op heden weinig aandacht gekregen in de literatuur, en bestaande algoritmen bieden momenteel beperkte schaalbaarheid. In Hoofdstuk 5 presenteren we een nieuw benaderingsalgoritme, gebaseerd op het inzicht dat een planningsprobleem met beperkingen kan worden gezien als een reeks van meerdere planningsproblemen zonder beperkingen. Dit principe stelt ons in staat om een schaalbaar algoritme af te leiden voor POMDP modellen met beperkingen in situaties met een of meerdere agenten. Het algoritme presteert beter dan bestaande algoritmen, en daarnaast representeert het een geheel nieuwe klasse van algoritmen voor planning met onzekerheid in domeinen met gedeeltelijke waarneembaarheid en beperkingen.

Hoofdstuk 6 van dit proefschrift richt zich op planning met beperkingen en onzekerheid in slimme energienetwerken. Het hoofdstuk heeft een meer praktische invalshoek, en daarnaast verbindt het de theoretische contributies met een maatschappelijk relevant toepassingsdomein. Het hoofdstuk legt uit hoe er in de toekomst congestie kan ontstaan in distributienetwerken, en hoe planning met onzekerheid en beperkingen kan worden gebruikt om congestie te voorkomen. Om dit te realiseren wordt er beschreven hoe informatie over netwerkbeperkingen kan worden geïntegreerd in planningsalgoritmen, en daarnaast presenteren we meerdere technieken die ervoor zorgen dat netwerkbeperkingen worden gerespecteerd tijdens het uitvoeren van een plan.

De technische bijdragen van dit proefschrift verbeteren de stand van de techniek in meerdere gebieden van het onderzoeksveld dat zich richt op planning met onzekerheid en beperkingen. Daarnaast verbetert het werk in dit proefschrift de toepasbaarheid van deze algoritmen voor planningsproblemen met maatschappelijke relevantie die optreden in de hedendaagse samenleving. Het resultaat is dat er een stap wordt gezet in de richting van planning met onzekerheid en beperkingen voor beslissingssystemen in de echte wereld.

List of publications

The research results presented in this dissertation are based on the publications listed below. For some papers the author of this dissertation is not listed as first author, but in that case the first author and second author of the paper contributed equally to the technical development of algorithms and writing the paper.

E. Walraven and M. T. J. Spaan (2019). "Point-Based Value Iteration for Finite-Horizon POMDPs". In: *Journal of Artificial Intelligence Research*.

E. Walraven and M. T. J. Spaan (2018). "Column Generation Algorithms for Constrained POMDPs". In: *Journal of Artificial Intelligence Research* 62, pp. 489–533.

D. M. Roijers, E. Walraven, and M. T. J. Spaan (2018). "Bootstrapping LPs in Value Iteration for Multi-Objective and Partially Observable MDPs". In: *Proceedings of the International Conference on Automated Planning and Scheduling*, pp. 218–226.

E. Walraven and M. T. J. Spaan (2017). "Accelerated Vector Pruning for Optimal POMDP Solvers". In: *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 3672–3678.

F. de Nijs, E. Walraven, M. M. de Weerdt, and M. T. J. Spaan (2017). "Bounding the Probability of Resource Constraint Violations in Multi-Agent MDPs". In: *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 3562–3568.

Other peer-reviewed publications

E. Walraven and M. T. J. Spaan (2016). "Planning Under Uncertainty for Aggregated Electric Vehicle Charging with Renewable Energy Supply". In: *Proceedings of the European Conference on Artificial Intelligence*, pp. 904–912.

E. Walraven and M. T. J. Spaan (2015). "Planning under Uncertainty with Weighted State Scenarios". In: *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pp. 912–921.

Acknowledgments

The completion of my dissertation marks the end of a long journey as a student and PhD candidate at TU Delft. I want to start by thanking Tomas, Hans, Mathijs and Cees. By teaching the basic concepts of theoretical computer science and algorithms you contributed to the first steps of my journey. Moreover, I learned many things while helping with your courses as a teaching assistant, and eventually it made my choice for the Algorithmics specialization in the master program really easy. This is where my collaboration with Matthijs started, during multiple courses and the final thesis project, which introduced me to planning research. Thank you for giving me the opportunity to join the group as a PhD candidate. Our collaboration has been very pleasant, and I think we learned a lot of things together!

During my PhD years a significant amount of time was spent in the office, but I have to admit that the office was not only a place to do research. I want to thank Peter Novák and Rens for many office conversations in the 'old' building. I especially want to thank Peter for the elaborate discussions about a wide range of topics such as robots for milking cows, colored soap bubbles and aviation. It is also important to mention Gleb here, because his numerous visits and his interesting questions were a constant factor in our office environment. I really enjoyed it and we had a lot of fun together! The move to the new office building created a fresh and productive environment to write the dissertation. I am very happy that I could work in another 'gezellige' office with Natalia and Canmanie, and I also want to thank you both for the activities that we organized with 'the Delft people' after work. During this year I also enjoyed the discussions with Koos, and I want to thank Anna for reminding me about the required coffee consumption!

The lunch discussions together with other (former) Algorithmics colleagues were also very important during my days at the faculty. Although we shifted from a strict 'lunch at noon' tradition to a more flexible lunch approach, the lunch discussions have always been interesting. I want to thank Thiago, Lei, Greg, Frits, Neil, Longjian, Pim, Laurens, Qisong, Bob, Peter Bosman, Simon, Joris, Michel, Shruti, Léon, Germán, Stefan, Jeroen, Bruno, Ivan, Jaume and Tuomas for joining us! I also want to thank Shemara, Kim and Stephen for helping me with administrative and technical questions.

Doing research on your own is important but not always a fun experience, and therefore it was nice to get the opportunity to work together with a few colleagues. I want to thank Frits for many in-depth discussions about constrained planning, and for the feedback on my work. I am very happy that we found the 'famous' Yost

& Washburn paper, which led to our new algorithms and our AAAI conference trip together. More importantly, this collaboration influenced our individual work and our research ideas significantly. I also want to thank Diederik, Germán and Nils for working together, which contributed greatly to this dissertation.

Finally, I want to express my gratitude to my family for supporting me throughout this journey, and for asking many questions about my work in Delft.

SIKS dissertation series

2011

01 Botond Cseke (RUN), *Variational Algorithms for Bayesian Inference in Latent Gaussian Models*

02 Nick Tinnemeier (UU), *Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language*

03 Jan Martijn van der Werf (TU/e), *Compositional Design and Verification of Component-Based Information Systems*

04 Hado van Hasselt (UU), *Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference*

05 Bas van der Raadt (VU), *Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline.*

06 Yiwen Wang (TU/e), *Semantically-Enhanced Recommendations in Cultural Heritage*

07 Yujia Cao (UT), *Multimodal Information Presentation for High Load Human Computer Interaction*

08 Nieske Vergunst (UU), *BDI-based Generation of Robust Task-Oriented Dialogues*

09 Tim de Jong (OU), *Contextualised Mobile Media for Learning*

10 Bart Bogaert (UvT), *Cloud Content Contention*

11 Dhaval Vyas (UT), *Designing for Awareness: An Experience-focused HCI Perspective*

12 Carmen Bratosin (TU/e), *Grid Architecture for Distributed Process Mining*

13 Xiaoyu Mao (UvT), *Airport under Control. Multiagent Scheduling for Airport Ground Handling*

14 Milan Lovric (EUR), *Behavioral Finance and Agent-Based Artificial Markets*

15 Marijn Koolen (UvA), *The Meaning of Structure: the Value of Link Evidence for Information Retrieval*

16 Maarten Schadd (UM), *Selective Search in Games of Different Complexity*

17 Jiyin He (UvA), *Exploring Topic Structure: Coherence, Diversity and Relatedness*

18 Mark Ponsen (UM), *Strategic Decision-Making in complex games*

19 Ellen Rusman (OU), *The Mind's Eye on Personal Profiles*

20 Qing Gu (VU), *Guiding service-oriented software engineering - A view-based approach*

21 Linda Terlouw (TUD), *Modularization and Specification of Service-Oriented Systems*

22 Junte Zhang (UvA), *System Evaluation of Archival Description and Access*

23 Wouter Weerkamp (UvA), *Finding People and their Utterances in Social Media*

24 Herwin van Welbergen (UT), *Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior*

25 Syed Waqar ul Qounain Jaffry (VU), *Analysis and Validation of Models for Trust Dynamics*

26 Matthijs Aart Pontier (VU), *Virtual Agents for Human Communication - Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots*

27 Aniel Bhulai (VU), *Dynamic website optimization through autonomous management of design patterns*

28 Rianne Kaptein (UvA), *Effective Focused Retrieval by Exploiting Query Context and Document Structure*

29 Faisal Kamiran (TU/e), *Discrimination-aware Classification*

30 Egon van den Broek (UT), *Affective Signal Processing (ASP): Unraveling the mystery of emotions*

31 Ludo Waltman (EUR), *Computational and Game-Theoretic Approaches for Modeling Bounded Rationality*

32 Nees-Jan van Eck (EUR), *Methodological Advances in Bibliometric Mapping of Science*

33 Tom van der Weide (UU), *Arguing to Motivate Decisions*

34 Paolo Turrini (UU), *Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations*

35 Maaike Harbers (UU), *Explaining Agent Behavior in Virtual Training*

36 Erik van der Spek (UU), *Experiments in serious game design: a cognitive approach*

37 Adriana Burlutiu (RUN), *Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference*

38 Nyree Lemmens (UM), *Bee-inspired Distributed Optimization*

39 Joost Westra (UU), *Organizing Adaptation using Agents in Serious Games*

40 Viktor Clerc (VU), *Architectural Knowledge Management in Global Software Development*

41 Luan Ibraimi (UT), *Cryptographically Enforced Distributed Data Access Control*

42 Michal Sindlar (UU), *Explaining Behavior through Mental State Attribution*

43 Henk van der Schuur (UU), *Process Improvement through Software Operation Knowledge*

44 Boris Reuderink (UT), *Robust Brain-Computer Interfaces*

45 Herman Stehouwer (UvT), *Statistical Language Models for Alternative Sequence Selection*

46 Beibei Hu (TUD), *Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work*

47 Azizi Bin Ab Aziz (VU), *Exploring Computational Models for Intelligent Support of Persons with Depression*

48 Mark Ter Maat (UT), *Response Selection and Turn-taking for a Sensitive Artificial Listening Agent*

49 Andreea Niculescu (UT), *Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality*

2012

01 Terry Kakeeto (UvT), *Relationship Marketing for SMEs in Uganda*

02 Muhammad Umair (VU), *Adaptivity, emotion, and Rationality in Human and Ambient Agent Models*

03 Adam Vanya (VU), *Supporting Architecture Evolution by Mining Software Repositories*

04 Jurriaan Souer (UU), *Development of Content Management System-based Web Applications*

05 Marijn Plomp (UU), *Maturing Interorganisational Information Systems*

06 Wolfgang Reinhardt (OU), *Awareness Support for Knowledge Workers in Research Networks*

07 Rianne van Lambalgen (VU), *When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions*

08 Gerben de Vries (UvA), *Kernel Methods for Vessel Trajectories*

09 Ricardo Neisse (UT), *Trust and Privacy Management Support for Context-Aware Service Platforms*

10 David Smits (TU/e), *Towards a Generic Distributed Adaptive Hypermedia Environment*

11 J.C.B. Rantham Prabhakara (TU/e), *Process Mining in the Large: Preprocessing, Discovery, and Diagnostics*

12 Kees van der Sluijs (TU/e), *Model Driven Design and Data Integration in Semantic Web Information Systems*

13 Suleman Shahid (UvT), *Fun and Face: Exploring non-verbal expressions of emotion during playful interactions*

14 Evgeny Knutov (TU/e), *Generic Adaptation Framework for Unifying Adaptive Web-based Systems*

15 Natalie van der Wal (VU), *Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes.*

16 Fiemke Both (VU), *Helping people by understanding them - Ambient Agents supporting task execution and depression treatment*

17 Amal Elgammal (UvT), *Towards a Comprehensive Framework for Business Process Compliance*

18 Eltjo Poort (VU), *Improving Solution Architecting Practices*

19 Helen Schonenberg (TU/e), *What's Next? Operational Support for Business Process Execution*

20 Ali Bahramisharif (RUN), *Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing*

21 Roberto Cornacchia (TUD), *Querying Sparse Matrices for Information Retrieval*

22 Thijs Vis (UvT), *Intelligence, politie en veiligheidsdienst: verenigbare grootheden?*

23 Christian Muehl (UT), *Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction*

24 Laurens van der Werff (UT), *Evaluation of Noisy Transcripts for Spoken Document Retrieval*

25 Silja Eckartz (UT), *Managing the Business Case Development in Inter-Organizational IT Projects: A Methodology and its Application*

26 Emile de Maat (UvA), *Making Sense of Legal Text*

27 Hayrettin Gurkok (UT), *Mind the Sheep! User Experience Evaluation & Brain-Computer Interface Games*

28 Nancy Pascall (UvT), *Engendering Technology Empowering Women*

29 Almer Tigelaar (UT), *Peer-to-Peer Information Retrieval*

30 Alina Pommeranz (TUD), *Designing Human-Centered Systems for Reflective Decision Making*

31 Emily Bagarukayo (RUN), *A Learning by Construction Approach for Higher Order Cognitive Skills Improvement, Building Capacity and Infrastructure*

32 Wietske Visser (TUD), *Qualitative multi-criteria preference representation and reasoning*

33 Rory Sie (OU), *Coalitions in Cooperation Networks (COCOON)*

34 Pavol Jancura (RUN), *Evolutionary analysis in PPI networks and applications*

35 Evert Haasdijk (VU), *Never Too Old To Learn – On-line Evolution of Controllers in Swarm- and Modular Robotics*

36 Denis Ssebugwawo (RUN), *Analysis and Evaluation of Collaborative Modeling Processes*

37 Agnes Nakakawa (RUN), *A Collaboration Process for Enterprise Architecture Creation*

38 Selmar Smit (VU), *Parameter Tuning and Scientific Testing in Evolutionary Algorithms*

39 Hassan Fatemi (UT), *Risk-aware design of value and coordination networks*

40 Agus Gunawan (UvT), *Information Access for SMEs in Indonesia*

41 Sebastian Kelle (OU), *Game Design Patterns for Learning*

42 Dominique Verpoorten (OU), *Reflection Amplifiers in self-regulated Learning*

43 *Withdrawn*

44 Anna Tordai (VU), *On Combining Alignment Techniques*

45 Benedikt Kratz (UvT), *A Model and Language for Business-aware Transactions*

46 Simon Carter (UvA), *Exploration and Exploitation of Multilingual Data for Statistical Machine Translation*

47 Manos Tsagkias (UvA), *Mining Social Media: Tracking Content and Predicting Behavior*

48 Jorn Bakker (TU/e), *Handling Abrupt Changes in Evolving Time-series Data*

49 Michael Kaisers (UM), *Learning against Learning - Evolutionary dynamics of reinforcement learning algorithms in strategic interactions*

50 Steven van Kervel (TUD), *Ontologogy driven Enterprise Information Systems Engineering*

51 Jeroen de Jong (TUD), *Heuristics in Dynamic Sceduling; a practical framework with a case study in elevator dispatching*

2013

01 Viorel Milea (EUR), *News Analytics for Financial Decision Support*

02 Erietta Liarou (CWI), *MonetDB/DataCell: Leveraging the Column-store Database Technology for Efficient and Scalable Stream Processing*

03 Szymon Klarman (VU), *Reasoning with Contexts in Description Logics*

04 Chetan Yadati (TUD), *Coordinating autonomous planning and scheduling*

05 Dulce Pumareja (UT), *Groupware Requirements Evolutions Patterns*

06 Romulo Goncalves (CWI), *The Data Cyclotron: Juggling Data and Queries for a Data Warehouse Audience*

07 Giel van Lankveld (UvT), *Quantifying Individual Player Differences*

08 Robbert-Jan Merk (VU), *Making enemies: cognitive modeling for opponent agents in fighter pilot simulators*

09 Fabio Gori (RUN), *Metagenomic Data Analysis: Computational Methods and Applications*

10 Jeewanie Jayasinghe Arachchige (UvT), *A Unified Modeling Framework for Service Design.*

11 Evangelos Pournaras (TUD), *Multi-level Reconfigurable Self-organization in Overlay Services*

12 Marian Razavian (VU), *Knowledge-driven Migration to Services*

13 Mohammad Safiri (UT), *Service Tailoring: User-centric creation of integrated IT-based homecare services to support independent living of elderly*

14 Jafar Tanha (UvA), *Ensemble Approaches to Semi-Supervised Learning Learning*

15 Daniel Hennes (UM), *Multiagent Learning - Dynamic Games and Applications*

16 Eric Kok (UU), *Exploring the practical benefits of argumentation in multi-agent deliberation*

17 Koen Kok (VU), *The PowerMatcher: Smart Coordination for the Smart Electricity Grid*

18 Jeroen Janssens (UvT), *Outlier Selection and One-Class Classification*

19 Renze Steenhuizen (TUD), *Coordinated Multi-Agent Planning and Scheduling*

20 Katja Hofmann (UvA), *Fast and Reliable Online Learning to Rank for Information Retrieval*

21 Sander Wubben (UvT), *Text-to-text generation by monolingual machine translation*

22 Tom Claassen (RUN), *Causal Discovery and Logic*

23 Patricio de Alencar Silva (UvT), *Value Activity Monitoring*

24 Haitham Bou Ammar (UM), *Automated Transfer in Reinforcement Learning*

25 Agnieszka Anna Latoszek-Berendsen (UM), *Intention-based Decision Support. A new way of representing and implementing clinical guidelines in a Decision Support System*

26 Alireza Zarghami (UT), *Architectural Support for Dynamic Homecare Service Provisioning*

27 Mohammad Huq (UT), *Inference-based Framework Managing Data Provenance*

28 Frans van der Sluis (UT), *When Complexity becomes Interesting: An Inquiry into the Information eXperience*

29 Iwan de Kok (UT), *Listening Heads*

30 Joyce Nakatumba (TU/e), *Resource-Aware Business Process Management: Analysis and Support*

31 Dinh Khoa Nguyen (UvT), *Blueprint Model and Language for Engineering Cloud Applications*

32 Kamakshi Rajagopal (OU), *Networking For Learning; The role of Networking in a Lifelong Learner's Professional Development*

33 Qi Gao (TUD), *User Modeling and Personalization in the Microblogging Sphere*

34 Kien Tjin-Kam-Jet (UT), *Distributed Deep Web Search*

35 Abdallah El Ali (UvA), *Minimal Mobile Human Computer Interaction*

36 Than Lam Hoang (TU/e), *Pattern Mining in Data Streams*

37 Dirk Börner (OU), *Ambient Learning Displays*

38 Eelco den Heijer (VU), *Autonomous Evolutionary Art*

39 Joop de Jong (TUD), *A Method for Enterprise Ontology based Design of Enterprise Information Systems*

40 Pim Nijssen (UM), *Monte-Carlo Tree Search for Multi-Player Games*

41 Jochem Liem (UvA), *Supporting the Conceptual Modelling of Dynamic Systems: A Knowledge Engineering Perspective on Qualitative Reasoning*

42 Léon Planken (TUD), *Algorithms for Simple Temporal Reasoning*

43 Marc Bron (UvA), *Exploration and Contextualization through Interaction and Concepts*

2014

01 Nicola Barile (UU), *Studies in Learning Monotone Models from Data*

02 Fiona Tuliyano (RUN), *Combining System Dynamics with a Domain Modeling Method*

03 Sergio Raul Duarte Torres (UT), *Information Retrieval for Children: Search Behavior and Solutions*

04 Hanna Jochmann-Mannak (UT), *Websites for children: search strategies and interface design - Three studies on children's search performance and evaluation*

05 Jurriaan van Reijsen (UU), *Knowledge Perspectives on Advancing Dynamic Capability*

06 Damian Tamburri (VU), *Supporting Networked Software Development*

07 Arya Adriansyah (TU/e), *Aligning Observed and Modeled Behavior*

08 Samur Araujo (TUD), *Data Integration over Distributed and Heterogeneous Data Endpoints*

09 Philip Jackson (UvT), *Toward Human-Level Artificial Intelligence: Representation and Computation of Meaning in Natural Language*

10 Ivan Salvador Razo Zapata (VU), *Service Value Networks*

11 Janneke van der Zwaan (TUD), *An Empathic Virtual Buddy for Social Support*

12 Willem van Willigen (VU), *Look Ma, No Hands: Aspects of Autonomous Vehicle Control*

13 Arlette van Wissen (VU), *Agent-Based Support for Behavior Change: Models and Applications in Health and Safety Domains*

14 Yangyang Shi (TUD), *Language Models With Meta-information*

15 Natalya Mogles (VU), *Agent-Based Analysis and Support of Human Functioning in Complex Socio-Technical Systems: Applications in Safety and Healthcare*

16 Krystyna Milian (VU), *Supporting trial recruitment and design by automatically interpreting eligibility criteria*

17 Kathrin Dentler (VU), *Computing healthcare quality indicators automatically: Secondary Use of Patient Data and Semantic Interoperability*

18 Mattijs Ghijsen (UvA), *Methods and Models for the Design and Study of Dynamic Agent Organizations*

19 Vinicius Ramos (TU/e), *Adaptive Hypermedia Courses: Qualitative and Quantitative Evaluation and Tool Support*

20 Mena Habib (UT), *Named Entity Extraction and Disambiguation for Informal Text: The Missing Link*

21 Kassidy Clark (TUD), *Negotiation and Monitoring in Open Environments*

22 Marieke Peeters (UU), *Personalized Educational Games - Developing agent-supported scenario-based training*

23 Eleftherios Sidirourgos (UvA/CWI), *Space Efficient Indexes for the Big Data Era*

24 Davide Ceolin (VU), *Trusting Semi-structured Web Data*

25 Martijn Lappenschaar (RUN), *New network models for the analysis of disease interaction*

26 Tim Baarslag (TUD), *What to Bid and When to Stop*

27 Rui Jorge Almeida (EUR), *Conditional Density Models Integrating Fuzzy and Probabilistic Representations of Uncertainty*

28 Anna Chmielowiec (VU), *Decentralized k-Clique Matching*

29 Jaap Kabbedijk (UU), *Variability in Multi-Tenant Enterprise Software*

30 Peter de Cock (UvT), *Anticipating Criminal Behaviour*

31 Leo van Moergestel (UU), *Agent Technology in Agile Multiparallel Manufacturing and Product Support*

32 Naser Ayat (UvA), *On Entity Resolution in Probabilistic Data*

33 Tesfa Tegegne (RUN), *Service Discovery in eHealth*

34 Christina Manteli (VU), *The Effect of Governance in Global Software Development: Analyzing Transactive Memory Systems.*

35 Joost van Ooijen (UU), *Cognitive Agents in Virtual Worlds: A Middleware Design Approach*

36 Joos Buijs (TU/e), *Flexible Evolutionary Algorithms for Mining Structured Process Models*

37 Maral Dadvar (UT), *Experts and Machines United Against Cyberbullying*

38 Danny Plass-Oude Bos (UT), *Making brain-computer interfaces better: improving usability through post-processing.*

39 Jasmina Maric (UvT), *Web Communities, Immigration, and Social Capital*

40 Walter Omona (RUN), *A Framework for Knowledge Management Using ICT in Higher Education*

41 Frederic Hogenboom (EUR), *Automated Detection of Financial Events in News Text*

42 Carsten Eijckhof (CWI/TUD), *Contextual Multidimensional Relevance Models*

43 Kevin Vlaanderen (UU), *Supporting Process Improvement using Method Increments*

44 Paulien Meesters (UvT), *Intelligent Blauw. Met als ondertitel: Intelligence-gestuurde politiezorg in gebiedsgebonden eenheden.*

45 Birgit Schmitz (OU), *Mobile Games for Learning: A Pattern-Based Approach*

46 Ke Tao (TUD), *Social Web Data Analytics: Relevance, Redundancy, Diversity*

47 Shangsong Liang (UvA), *Fusion and Diversification in Information Retrieval*

2015

01 Niels Netten (UvA), *Machine Learning for Relevance of Information in Crisis Response*

02 Faiza Bukhsh (UvT), *Smart auditing: Innovative Compliance Checking in Customs Controls*

03 Twan van Laarhoven (RUN), *Machine learning for network data*

04 Howard Spoelstra (OU), *Collaborations in Open Learning Environments*

05 Christoph Bösch (UT), *Cryptographically Enforced Search Pattern Hiding*

06 Farideh Heidari (TUD), *Business Process Quality Computation - Computing Non-Functional Requirements to Improve Business Processes*

07 Maria-Hendrike Peetz (UvA), *Time-Aware Online Reputation Analysis*

08 Jie Jiang (TUD), *Organizational Compliance: An agent-based model for designing and evaluating organizational interactions*

09 Randy Klaassen (UT), *HCI Perspectives on Behavior Change Support Systems*

10 Henry Hermans (OU), *OpenU: design of an integrated system to support lifelong learning*

11 Yongming Luo (TU/e), *Designing algorithms for big graph datasets: A study of computing bisimulation and joins*

12 Julie M. Birkholz (VU), *Modi Operandi of Social Network Dynamics: The Effect of Context on Scientific Collaboration Networks*

13 Giuseppe Procaccianti (VU), *Energy-Efficient Software*

14 Bart van Straalen (UT), *A cognitive approach to modeling bad news conversations*

15 Klaas Andries de Graaf (VU), *Ontology-based Software Architecture Documentation*

16 Changyun Wei (UT), *Cognitive Coordination for Cooperative Multi-Robot Teamwork*

17 André van Cleeff (UT), *Physical and Digital Security Mechanisms: Properties, Combinations and Trade-offs*

18 Holger Pirk (CWI), *Waste Not, Want Not! - Managing Relational Data in Asymmetric Memories*

19 Bernardo Tabuenca (OU), *Ubiquitous Technology for Lifelong Learners*

20 Lois Vanhée (UU), *Using Culture and Values to Support Flexible Coordination*

21 Sibren Fetter (OU), *Using Peer-Support to Expand and Stabilize Online Learning*

22 Zhemin Zhu (UT), *Co-occurrence Rate Networks*

23 Luit Gazendam (VU), *Cataloguer Support in Cultural Heritage*

24 Richard Berendsen (UvA), *Finding People, Papers, and Posts: Vertical Search Algorithms and Evaluation*

25 Steven Woudenberg (UU), *Bayesian Tools for Early Disease Detection*

26 Alexander Hogenboom (EUR), *Sentiment Analysis of Text Guided by Semantics and Structure*

27 Sándor Héman (CWI), *Updating compressed colomn stores*

28 Janet Bagorogoza (TiU), *Knowledge Management and High Performance; The Uganda Financial Institutions Model for HPO*

29 Hendrik Baier (UM), *Monte-Carlo Tree Search Enhancements for One-Player and Two-Player Domains*

30 Kiavash Bahreini (OU), *Real-time Multimodal Emotion Recognition in E-Learning*

31 Yakup Koç (TUD), *On the robustness of Power Grids*

32 Jerome Gard (UL), *Corporate Venture Management in SMEs*

33 Frederik Schadd (TUD), *Ontology Mapping with Auxiliary Resources*

34 Victor de Graaf (UT), *Gesocial Recommender Systems*

35 Jungxao Xu (TUD), *Affective Body Language of Humanoid Robots: Perception and Effects in Human Robot Interaction*

2016

01 Syed Saiden Abbas (RUN), *Recognition of Shapes by Humans and Machines*

02 Michiel Christiaan Meulendijk (UU), *Optimizing medication reviews through decision support: prescribing a better pill to swallow*

03 Maya Sappelli (RUN), *Knowledge Work in Context: User Centered Knowledge Worker Support*

04 Laurens Rietveld (VU), *Publishing and Consuming Linked Data*

05 Evgeny Sherkhonov (UvA), *Expanded Acyclic Queries: Containment and an Application in Explaining Missing Answers*

06 Michel Wilson (TUD), *Robust scheduling in an uncertain environment*

07 Jeroen de Man (VU), *Measuring and modeling negative emotions for virtual training*

08 Matje van de Camp (TiU), *A Link to the Past: Constructing Historical Social Networks from Unstructured Data*

09 Archana Nottamkandath (VU), *Trusting Crowdsourced Information on Cultural Artefacts*

10 George Karafotias (VU), *Parameter Control for Evolutionary Algorithms*

11 Anne Schuth (UvA), *Search Engines that Learn from Their Users*

12 Max Knobbout (UU), *Logics for Modelling and Verifying Normative Multi-Agent Systems*

13 Nana Baah Gyan (VU), *The Web, Speech Technologies and Rural Development in West Africa - An ICT4D Approach*

14 Ravi Khadka (UU), *Revisiting Legacy Software System Modernization*

15 Steffen Michels (RUN), *Hybrid Probabilistic Logics - Theoretical Aspects, Algorithms and Experiments*

16 Guangliang Li (UvA), *Socially Intelligent Autonomous Agents that Learn from Human Reward*

17 Berend Weel (VU), *Towards Embodied Evolution of Robot Organisms*

18 Albert Meroño Peñuela (VU), *Refining Statistical Data on the Web*

19 Julia Efremova (Tu/e), *Mining Social Structures from Genealogical Data*

20 Daan Odijk (UvA), *Context & Semantics in News & Web Search*

21 Alejandro Moreno Célleri (UT), *From Traditional to Interactive Playspaces: Automatic Analysis of Player Behavior in the Interactive Tag Playground*

22 Grace Lewis (VU), *Software Architecture Strategies for Cyber-Foraging Systems*

23 Fei Cai (UvA), *Query Auto Completion in Information Retrieval*

24 Brend Wanders (UT), *Repurposing and Probabilistic Integration of Data; An Iterative and data model independent approach*

25 Julia Kiseleva (TU/e), *Using Contextual Information to Understand Searching and Browsing Behavior*

26 Dilhan Thilakarathne (VU), *In or Out of Control: Exploring Computational Models to Study the Role of Human Awareness and Control in Behavioural Choices, with Applications in Aviation and Energy Management Domains*

27 Wen Li (TUD), *Understanding Geo-spatial Information on Social Media*

28 Mingxin Zhang (TUD), *Large-scale Agent-based Social Simulation - A study on epidemic prediction and control*

29 Nicolas Höning (TUD), *Peak reduction in decentralised electricity systems - Markets and prices for flexible planning*

30 Ruud Mattheij (UvT), *The Eyes Have It*

31 Mohammad Khelghati (UT), *Deep web content monitoring*

32 Eelco Vriezekolk (UT), *Assessing Telecommunication Service Availability Risks for Crisis Organisations*

33 Peter Bloem (UvA), *Single Sample Statistics, exercises in learning from just one example*

34 Dennis Schunselaar (TU/e), *Configurable Process Trees: Elicitation, Analysis, and Enactment*

35 Zhaochun Ren (UvA), *Monitoring Social Media: Summarization, Classification and Recommendation*

36 Daphne Karreman (UT), *Beyond R2D2: The design of nonverbal interaction behavior optimized for robot-specific morphologies*

37 Giovanni Sileno (UvA), *Aligning Law and Action - a conceptual and computational inquiry*

38 Andrea Minuto (UT), *Materials that Matter - Smart Materials meet Art & Interaction Design*

39 Merijn Bruijnes (UT), *Believable Suspect Agents; Response and Interpersonal Style Selection for an Artificial Suspect*

40 Christian Detweiler (TUD), *Accounting for Values in Design*

41 Thomas King (TUD), *Governing Governance: A Formal Framework for Analysing Institutional Design and Enactment Governance*

42 Spyros Martzoukos (UvA), *Combinatorial and Compositional Aspects of Bilingual Aligned Corpora*

43 Saskia Koldijk (RUN), *Context-Aware Support for Stress Self-Management: From Theory to Practice*

44 Thibault Sellam (UvA), *Automatic Assistants for Database Exploration*

45 Bram van de Laar (UT), *Experiencing Brain-Computer Interface Control*

46 Jorge Gallego Perez (UT), *Robots to Make you Happy*

47 Christina Weber (UL), *Real-time foresight - Preparedness for dynamic innovation networks*

48 Tanja Buttler (TUD), *Collecting Lessons Learned*

49 Gleb Polevoy (TUD), *Participation and Interaction in Projects. A Game-Theoretic Analysis*

50 Yan Wang (UvT), *The Bridge of Dreams: Towards a Method for Operational Performance Alignment in IT-enabled Service Supply Chains*

2017

01 Jan-Jaap Oerlemans (UL), *Investigating Cybercrime*

02 Sjoerd Timmer (UU), *Designing and Understanding Forensic Bayesian Networks using Argumentation*

03 Daniël Harold Telgen (UU), *Grid Manufacturing; A Cyber-Physical Approach with Autonomous Products and Reconfigurable Manufacturing Machines*

04 Mrunal Gawade (CWI), *Multi-core Parallelism in a Column-store*

05 Mahdieh Shadi (UvA), *Collaboration Behavior*

06 Damir Vandic (EUR), *Intelligent Information Systems for Web Product Search*

07 Roel Bertens (UU), *Insight in Information: from Abstract to Anomaly*

08 Rob Konijn (VU), *Detecting Interesting Differences:Data Mining in Health Insurance Data using Outlier Detection and Subgroup Discovery*

09 Dong Nguyen (UT), *Text as Social and Cultural Data: A Computational Perspective on Variation in Text*

10 Robby van Delden (UT), *(Steering) Interactive Play Behavior*

11 Florian Kunneman (RUN), *Modelling patterns of time and emotion in Twitter #anticipointment*

12 Sander Leemans (TU/e), *Robust Process Mining with Guarantees*

13 Gijs Huisman (UT), *Social Touch Technology - Extending the reach of social touch through haptic technology*

14 Shoshannah Tekofsky (UvT), *You Are Who You Play You Are: Modelling Player Traits from Video Game Behavior*

15 Peter Berck (RUN), *Memory-Based Text Correction*

16 Aleksandr Chuklin (UvA), *Understanding and Modeling Users of Modern Search Engines*

17 Daniel Dimov (UL), *Crowdsourced Online Dispute Resolution*

18 Ridho Reinanda (UvA), *Entity Associations for Search*

19 Jeroen Vuurens (UT), *Proximity of Terms, Texts and Semantic Vectors in Information Retrieval*

20 Mohammadbashir Sedighi (TUD), *Fostering Engagement in Knowledge Sharing: The Role of Perceived Benefits, Costs and Visibility*

21 Jeroen Linssen (UT), *Meta Matters in Interactive Storytelling and Serious Gaming (A Play on Worlds)*

22 Sara Magliacane (VU), *Logics for causal inference under uncertainty*

23 David Graus (UvA), *Entities of Interest — Discovery in Digital Traces*

24 Chang Wang (TUD), *Use of Affordances for Efficient Robot Learning*

25 Veruska Zamborlini (VU), *Knowledge Representation for Clinical Guidelines, with applications to Multimorbidity Analysis and Literature Search*

26 Merel Jung (UT), *Socially intelligent robots that understand and respond to human touch*

27 Michiel Joosse (UT), *Investigating Positioning and Gaze Behaviors of Social Robots: People's Preferences, Perceptions and Behaviors*

28 John Klein (VU), *Architecture Practices for Complex Contexts*

29 Adel Alhuraibi (UvT), *From IT-Business Strategic Alignment to Performance: A Moderated Mediation Model of Social Innovation, and Enterprise Governance of IT*

30 Wilma Latuny (UvT), *The Power of Facial Expressions*

31 Ben Ruijl (UL), *Advances in computational methods for QFT calculations*

32 Thaer Samar (RUN), *Access to and Retrievability of Content in Web Archives*

33 Brigit van Loggem (OU), *Towards a Design Rationale for Software Documentation: A Model of Computer-Mediated Activity*

34 Maren Scheffel (OU), *The Evaluation Framework for Learning Analytics*

35 Martine de Vos (VU), *Interpreting natural science spreadsheets*

36 Yuanhao Guo (UL), *Shape Analysis for Phenotype Characterisation from High-throughput Imaging*

37 Alejandro Montes Garcia (TU/e), *WiBAF: A Within Browser Adaptation Framework that Enables Control over Privacy*

38 Alex Kayal (TUD), *Normative Social Applications*

39 Sara Ahmadi (RUN), *Exploiting properties of the human auditory system and compressive sensing methods to increase noise robustness in ASR*

40 Altaf Hussain Abro (VU), *Steer your Mind: Computational Exploration of Human Control in Relation to Emotions, Desires and Social Support For applications in human-aware support systems*

41 Adnan Manzoor (VU), *Minding a Healthy Lifestyle: An Exploration of Mental Processes and a Smart Environment to Provide Support for a Healthy Lifestyle*

42 Elena Sokolova (RUN), *Causal discovery from mixed and missing data with applications on ADHD datasets*

43 Maaike de Boer (RUN), *Semantic Mapping in Video Retrieval*

44 Garm Lucassen (UU), *Understanding User Stories - Computational Linguistics in Agile Requirements Engineering*

45 Bas Testerink (UU), *Decentralized Runtime Norm Enforcement*

46 Jan Schneider (OU), *Sensor-based Learning Support*

47 Jie Yang (TUD), *Crowd Knowledge Creation Acceleration*

48 Angel Suarez (OU), *Collaborative inquiry-based learning*

2018

01 Han van der Aa (VU), *Comparing and Aligning Process Representations*

02 Felix Mannhardt (TU/e), *Multi-perspective Process Mining*

03 Steven Bosems (UT), *Causal Models For Well-Being: Knowledge Modeling, Model-Driven Development of Context-Aware Applications, and Behavior Prediction*

04 Jordan Janeiro (TUD), *Flexible Coordination Support for Diagnosis Teams in Data-Centric Engineering Tasks*

05 Hugo Huurdeman (UvA), *Supporting the Complex Dynamics of the Information Seeking Process*

06 Dan Ionita (UT), *Model-Driven Information Security Risk Assessment of Socio-Technical Systems*

07 Jieting Luo (UU), *A formal account of opportunism in multi-agent systems*

08 Rick Smetsers (RUN), *Advances in Model Learning for Software Systems*

09 Xu Xie (TUD), *Data Assimilation in Discrete Event Simulations*

10 Julienka Mollee (VU), *Moving forward: supporting physical activity behavior change through intelligent technology*

11 Mahdi Sargolzaei (UvA), *Enabling Framework for Service-oriented Collaborative Networks*

12 Xixi Lu (TU/e), *Using behavioral context in process mining*

13 Seyed Amin Tabatabaei (VU), *Computing a Sustainable Future*

14 Bart Joosten (UvT), *Detecting Social Signals with Spatiotemporal Gabor Filters*

15 Naser Davarzani (UM), *Biomarker discovery in heart failure*

16 Jaebok Kim (UT), *Automatic recognition of engagement and emotion in a group of children*

17 Jianpeng Zhang (TU/e), *On Graph Sample Clustering*

18 Henriette Nakad (UL), *De Notaris en Private Rechtspraak*

19 Minh Duc Pham (VU), *Emergent relational schemas for RDF*

20 Manxia Liu (RUN), *Time and Bayesian Networks*

21 Aad Slootmaker (OU), *EMERGO: a generic platform for authoring and playing scenario-based serious games*

22 Eric Fernandes de Mello Araujo (VU), *Contagious: Modeling the Spread of Behaviours, Perceptions and Emotions in Social Networks*

23 Kim Schouten (EUR), *Semantics-driven Aspect-Based Sentiment Analysis*

24 Jered Vroon (UT), *Responsive Social Positioning Behaviour for Semi-Autonomous Telepresence Robots*

25 Riste Gligorov (VU), *Serious Games in Audio-Visual Collections*

26 Roelof Anne Jelle de Vries (UT), *Theory-Based and Tailor-Made: Motivational Messages for Behavior Change Technology*

27 Maikel Leemans (TU/e), *Hierarchical Process Mining for Scalable Software Analysis*

28 Christian Willemse (UT), *Social Touch Technologies: How they feel and how they make you feel*

29 Yu Gu (UvT), *Emotion Recognition from Mandarin Speech*

30 Wouter Beek (VU), *The 'K' in 'semantic web' stands for 'knowledge': scaling semantics to the web*

2019

01 Rob van Eijk (UL), *Comparing and Aligning Process Representations*

02 Emmanuelle Beauxis Aussalet (CWI, UU), *Statistics and Visualizations for Assessing Class Size Uncertainty*

03 Eduardo Gonzalez Lopez de Murillas (TU/e), *Process Mining on Databases: Extracting Event Data from Real Life Data Sources*

04 Ridho Rahmadi (RUN), *Finding stable causal structures from clinical data*

05 Sebastiaan van Zelst (TU/e), *Process Mining with Streaming Data*

06 Chris Dijkshoorn (VU), *Niche sourcing for Improving Access to Linked Cultural Heritage Datasets*

07 Soude Fazeli (OU), *Recommender Systems in Social Learning Platforms*

08 Frits de Nijs (TUD), *Resource-constrained Multi-agent Markov Decision Processes*

09 Fahimeh Alizadeh Moghaddam (UvA), *Self-adaptation for energy efficiency in software systems*

10 Qing Chuan Ye (EUR), *Multi-objective Optimization Methods for Allocation and Prediction*

11 Yue Zhao (TUD), *Learning Analytics Technology to Understand Learner Behavioral Engagement in MOOCs*

12 Jacqueline Heinerman (VU), *Better Together*

13 Guanliang Chen (TUD), *MOOC Analytics: Learner Modeling and Content*

14 Daniel Davis (TUD), *Large-Scale Learning Analytics: Modeling Learner Behavior & Improving Learning Outcomes in Massive Open Online Courses*

15 Erwin Walraven (TUD), *Planning under Uncertainty in Constrained and Partially Observable Environments*