Journal of Artificial Intelligence Research 62 (2018) 489-533 Submitted 11/17; published 07/18

Column Generation Algorithms for Constrained POMDPs

Erwin Walraven E.M.P.WALRAVEN@TUDELFT.NL
Matthijs T. J. Spaan M.T.J.SPAAN@TUDELFT.NL
Delft University of Technology, The Netherlands

Abstract

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. First, we use a tailored point-based POMDP algorithm 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. Experiments based on several domains show that our method outperforms the current state of the art.

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, & 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 & 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, & 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, & 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, Vamplew, Whiteson, & Dazeley, 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 can conclude that decision making under uncertainty subject to additional 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, & 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 & Durfee, 2003; Wu & Durfee, 2010; Agrawal, Varakantham, & 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, & Thrun, 2003) has been generalized to Constrained POMDPs (Kim, Lee, Kim, & Poupart, 2011). In addition, a method has been proposed to optimize finite-state controllers using approximate linear programming (Poupart, Malhotra, Pei, Kim, Goh, & Bowling, 2015), which is also based on the linear program used for Constrained MDPs. The aforementioned approaches have two common 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 & 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, & 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.

1.1 Contributions

In this paper 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 of the paper 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 this paper. 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 a tailored point-based POMDP algorithm for solving subproblems, which first computes a vector-based value function and then translates this 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.

1.2 Overview

This paper is structured as follows. In Section 2 we introduce Constrained POMDPs and an extension suitable for multi-agent planning. In Section 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 4 we describe techniques to solve column generation subproblems using an approximate POMDP algorithm, which significantly improve the performance of column generation. In Section 5 we provide the results of our experimental evaluation. In Section 6 and Section 7 we describe related work, our conclusions and future work.

2. Constrained POMDPs

In this section we provide a formal introduction to Partially Observable Markov Decision Processes (POMDPs). We also describe Constrained POMDPs, which extend the POMDP model with an additional constraint on a secondary objective.

2.1 Partially Observable Markov Decision Processes

A Partially Observable Markov Decision Process (Kaelbling et al., 1998) provides a mathematical framework for sequential decision making under uncertainty in partially observable domains. It is an extension of a Markov Decision Processes (Puterman, 1994) which allows for modeling of partial observability. A POMDP models an agent that interacts with an environment by executing actions, in such a way that a notion of reward is optimized (e.g., in order to achieve a particular goal). Decision making is difficult for the agent because of two reasons. First, the agent does not precisely know how the executed actions affect the environment, because the environment may behave stochastically. Second, the agent may not be able to observe all aspects of the environment when making a decision, which means that some information about the environment remains hidden from the viewpoint of the agent. The interaction between the agent and the environment modeled by a POMDP takes into account both types of uncertainty.

We consider finite-horizon planning problems in this paper, because most applications of Constrained POMDPs naturally require a finite planning horizon and typically constraints on resource consumption do not include discounting. Formally, a finite-horizon POMDP is defined by a tuple $M = \langle S, A, O, T, Z, R, b_1, h \rangle$. The set S contains all possible states $s \in S$, which encode the current characteristics of the environment, and the set A contains all possible actions $a \in A$ that can be executed. The function $T : S \times A \times S \rightarrow [0, 1]$ defines the state transition probabilities, such that $T(s, a, s') = P(s'|s, a)$ defines the probability that the state changes from state $s \in S$ to state $s' \in S$ after executing action $a \in A$. The function T is also known as the transition function. If action a is executed in state s, then the agent receives reward $R(s, a)$ from the environment. Rather than observing the current state of the environment directly, as would occur in an MDP, the agent gets observations $o \in O$ which provide some information about the current state of the environment. This is formalized using an observation function $Z : A \times S \times O \rightarrow [0,1]$, such that $Z(a,s',o) = P(o|a,s')$ defines the probability to observe o after executing action a and transitioning to state s'. In this paper it is assumed that we are dealing with finite sets of states, actions and observations. The parameter h denotes a finite time horizon, such that the agent executes actions in time steps $1,\dots,h$, and execution ends at time step $h+1$.

In order to decide which action needs to be executed, the agent keeps track of a belief $b \in \Delta(S)$ based on the observations it receives. This belief is a probability distribution over the states in S. The initial belief over states is defined by $b_1 \in \Delta(S)$, which can be repeatedly updated using Bayes' rule after executing action a and observing o:

$$b_a^o(s') = \frac{P(o|a,s')}{P(o|b,a)} \sum_{s \in S} P(s'|s,a)b(s) \qquad \forall s' \in S, \tag{1}$$

in which $P(o|b,a) = \sum_{s' \in S} P(o|a,s') \sum_{s \in S} P(s'|s,a)b(s)$ is used as a normalizing constant.

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:

$$V^\pi(t,b) = E_\pi \left[\sum_{t'=t}^{h} R(b_{t'}, \pi(t', b_{t'})) \,\middle|\, b_t = b \right], \tag{2}$$

where $b_{t'}$ denotes 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. Such an optimal policy can be characterized in terms of value functions. 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:

$$V^{\pi^*}(t,b) = \begin{cases} \max_{a \in A} \left[\sum_{s \in S} R(s,a)b(s) + \sum_{o \in O} P(o|b,a) V^{\pi^*}(t+1, b_a^o) \right] & t \leq h \\ 0 & \text{otherwise} \end{cases}. \tag{3}$$

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

$$\pi^*(t,b) = \operatorname*{arg\,max}_{a \in A} \left[\sum_{s \in S} R(s,a)b(s) + \sum_{o \in O} P(o|b,a) V^{\pi^*}(t+1, b_a^o) \right], \tag{4}$$

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

It has been shown that value functions for finite-horizon POMDPs are piecewise linear and convex (Sondik, 1971). This means that the value function of each time step can be represented using a finite set of $|S|$-dimensional vectors. Value functions can be obtained using value iteration, which repeatedly executes dynamic programming iterations based on

Bellman backups defined by the first case in Equation 3. Computing optimal value functions can be done using incremental pruning (Cassandra et al., 1997), combined with accelerated vector pruning algorithms (Walraven & Spaan, 2017). It should be noted that these algorithms assume an infinite horizon with discounting, but the same dynamic programming procedure can be used for finite-horizon problems after discarding the discounting.

Solving POMDPs to optimality is PSPACE-complete (Papadimitriou & Tsitsiklis, 1987) and therefore it is typically intractable to find exact solutions to larger problems. Instead, a wide range of approximate algorithms has been proposed for POMDPs with an infinite horizon. Most notably, point-based value iteration techniques (Pineau et al., 2003; Spaan & Vlassis, 2005) execute dynamic programming backups based on a finite set of belief points. More recent algorithms such as HSVI (Smith & Simmons, 2005), SARSOP (Kurniawati, Hsu, & Lee, 2008) and GapMin (Poupart, Kim, & Kim, 2011) keep track of a lower and upper bound on the optimal value function, and they typically search for additional belief points that have the potential to improve both bounds. In this paper we use an adapted point-based method for finite-horizon POMDPs. A more elaborate introduction to this algorithm is deferred to Section 4.1.

2.2 Constrained Partially Observable Markov Decision Processes

Now we turn our attention to a setting in which an agent aims to maximize the expected value while respecting a constraint on a secondary objective, as illustrated in the introduction of the paper. 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 et al., 2008).

We define a Constrained POMDP using a tuple $M = \langle S, A, O, T, Z, R, C, L, b_1, h \rangle$. 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:

$$C^{\pi}(t, b) = E_{\pi}\left[\sum_{t'=t}^{h} C(b_{t'}, \pi(t', b_{t'})) \;\middle|\; b_t = b \right], \tag{5}$$

where $b_{t'}$ denotes 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:

$$\begin{aligned} \max_{\pi} \;\; & V^{\pi}(1, b_1) \\ \text{s.t. } & C^{\pi}(1, b_1) \leq L. \end{aligned} \tag{6}$$

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 et al. (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 3.

2.3 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, in this paper 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 $M_i = \langle S_i, A_i, O_i, T_i, Z_i, R_i, C_i, b_{1,i}, h \rangle$, 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 paper. The main idea is to find policies $\pi_1, \ldots, \pi_n$ for the agents, such that the total expected reward is maximized while the expected sum of costs is bounded:

$$\begin{aligned} \max_{\{\pi_1,\ldots,\pi_n\}} \quad & \sum_{i=1}^{n} V_i^{\pi_i}(1, b_{1,i}) \\ \text{s.t.} \quad & \sum_{i=1}^{n} C_i^{\pi_i}(1, b_{1,i}) \leq L. \end{aligned} \tag{7}$$

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

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 4 we describe how the scalability of the column generation algorithm can be improved by integrating approximate POMDP algorithms.

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:

$$\begin{aligned} \phi = \max\ & \sum_{i=1}^{n} \sum_{\pi_i \in K_i} V_i^{\pi_i} \cdot x_{i,\pi_i} & & \\ \text{s.t.}\ & \sum_{i=1}^{n} \sum_{\pi_i \in K_i} C_i^{\pi_i} \cdot x_{i,\pi_i} \leq L & & \text{(dual variable: } \lambda) \\ & \sum_{\pi_i \in K_i} x_{i,\pi_i} = 1 & \forall i \quad & \text{(dual variables: } \lambda_i) \\ & x_{i,\pi_i} \geq 0 & \forall i, \pi_i. & \end{aligned} \tag{8}$$

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 (8) with only a subset of columns. An upper bound ϕ_u

can be derived using the following Lagrangian relaxation:

$$\begin{aligned} \phi_u = \max \ & \sum_{i=1}^{n} \sum_{\pi_i \in K_i} V_i^{\pi_i} \cdot x_{i,\pi_i} + \lambda \left(L - \sum_{i=1}^{n} \sum_{\pi_i \in K_i} C_i^{\pi_i} \cdot x_{i,\pi_i} \right) \\ \text{s.t.} \ & \sum_{\pi_i \in K_i} x_{i,\pi_i} = 1 \qquad \forall i \\ & x_{i,\pi_i} \geq 0 \qquad\qquad \forall i, \pi_i, \end{aligned} \tag{9}$$

in which λ is the Lagrangian multiplier corresponding to the first constraint in (8). Since the constraints only affect individual agents, the upper bound can also be written as:

$$\phi_u = \lambda \cdot L + \sum_{i=1}^{n} \left[\max_{\pi_i \in K_i} \left(V_i^{\pi_i} - \lambda \cdot C_i^{\pi_i} \right) \right]. \tag{10}$$

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:

$$G_i(s,a) = R_i(s,a) - \lambda \cdot C_i(s,a). \tag{11}$$

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 1. On lines 2-7 the algorithm starts with initializing the LP shown in (8) 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 1 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 8 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 8. 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.

Algorithm 1: 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 $Y_i \leftarrow \{(\pi_i, x_{i,\pi_i}) \mid \pi_i \in K_i \text{ and } x_{i,\pi_i} > 0\} \quad \forall i$
22 return $\{Y_1, \dots, Y_n\}$
```

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, & Magnanti, 1977), which we explain using the following LP formulation in standard form:

$$\begin{aligned} \max\ & c^\top x \\ \text{s.t.}\ & Ax \leq b \\ & x \geq 0, \end{aligned} \tag{12}$$

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

$$\bar{c} = c - A^{\top} y, \tag{13}$$

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 et al., 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 (8) 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 (8), we derive that the reduced cost of a policy π_i is equal to:

$$\begin{aligned}
\bar{c}_{\pi_i} &= V_i^{\pi_i} - \lambda \cdot C_i^{\pi_i} - \lambda_{i,0} \cdot 1 - \lambda_{i,1} \cdot (-1) && (14)\\
&= V_i^{\pi_i} - \lambda \cdot C_i^{\pi_i} - \lambda_{i,0} + \lambda_{i,1} && (15)\\
&= V_i^{\pi_i} - \lambda \cdot C_i^{\pi_i} - (\lambda_{i,0} - \lambda_{i,1}) && (16)\\
&= V_i^{\pi_i} - \lambda \cdot C_i^{\pi_i} - \lambda_i. && (17)
\end{aligned}$$

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 1 can be interpreted as computing columns which maximize reduced cost.

Lemma 1. *In each iteration, Algorithm 1 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:

$$\begin{aligned}
G_i^{\pi_i} &= E_{\pi_i}\left[\sum_{t=1}^{h} G_i(b_t, \pi_i(t, b_t)) \;\middle|\; b_1 = b_{1,i}\right] && (18)\\
&= E_{\pi_i}\left[\sum_{t=1}^{h} R_i(b_t, \pi_i(t, b_t)) \;\middle|\; b_1 = b_{1,i}\right] - \lambda \cdot E_{\pi_i}\left[\sum_{t=1}^{h} C_i(b_t, \pi_i(t, b_t)) \;\middle|\; b_1 = b_{1,i}\right] && (19)\\
&= V_i^{\pi_i} - \lambda \cdot C_i^{\pi_i}, && (20)
\end{aligned}$$

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 17 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 & 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 2 and Lemma 3.

Lemma 2. *If Algorithm 1 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 1 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. $\square$

Lemma 3. *The master LP in Equation 8 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 and Equation 5. 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. $\square$

Algorithm 1 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 4. *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 1), 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 5. *If the master LP solution corresponds to the optimal Constrained POMDP solution, then the dual price λ becomes constant during the execution of Algorithm 1.*

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 1, as shown in Theorems 1 and 2 below.

Theorem 1. *Algorithm 1 terminates if and only if it has found an optimal Constrained POMDP solution.*

Proof. This follows immediately from Lemma 4 and Lemma 5. 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 4), 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 5), which leads to termination. □

Theorem 2. *Algorithm 1 computes an optimal Constrained POMDP solution.*

Proof. Based on Theorem 1 we know that Algorithm 1 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 1), columns with positive reduced cost are always new (Lemma 2) and the number of columns with positive reduced cost is finite (Lemma 3). 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 1 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 3. *Algorithm 1 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 (8). 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 & Steiglitz, 1982). Now it follows that there is at most one agent which has two policies with non-zero probability. $\square$

To summarize, in our analysis in this section we have shown that Algorithm 1 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.

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 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 18. 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 10 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 a tailored approximate POMDP algorithm. In particular, we present a point-based value iteration algorithm for finite-horizon POMDPs, which we can use to derive 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 the point-based algorithm to a policy graph, which allows for exact policy evaluation. Finally,

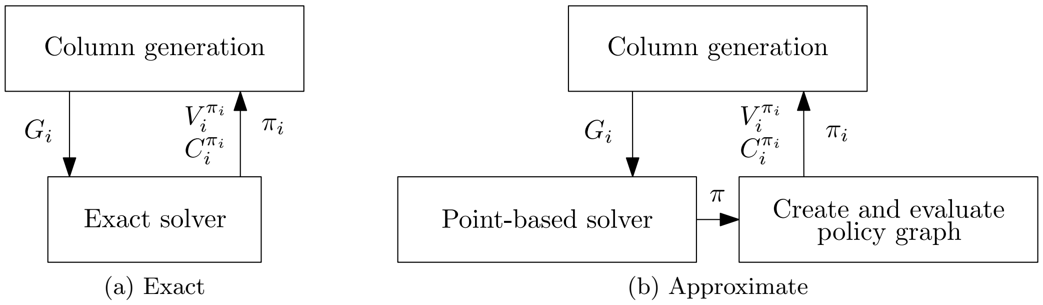


Figure 1: Overview of exact column generation and column generation with point-based methods and policy graph generation.

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 1, which indicates the differences between exact column generation and column generation based on point-based algorithms and policy graphs. In Figure 1a an exact subproblem solution is computed, which immediately gives the coefficients required in the LP. In Figure 1b a point-based solver 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 Section 4.1 we describe a tailored point-based value iteration algorithm suitable for solving finite-horizon problems. In Sections 4.2 and 4.3 we introduce policy graphs and we describe how they can be created and evaluated. In Section 4.4 we discuss our modified column generation algorithm, which is called CGCP. An additional analysis of the graph construction is provided in Section 4.5.

4.1 Point-Based Value Iteration for Finite-Horizon POMDPs

Point-based value iteration algorithms (Pineau et al., 2003; Spaan & Vlassis, 2005; Smith & Simmons, 2005; Kurniawati et al., 2008; Poupart et al., 2011) represent a class of approximate algorithms for POMDPs, which execute dynamic programming backups on a finite set of belief points. Restricting the backups to a finite set of belief points makes it computationally more efficient than optimizing over the entire continuous belief simplex. The value function of a POMDP policy π_i is represented by a finite set Γ containing alpha vectors of length $|S_i|$, which are typically denoted by α. Column generation solves POMDPs with a modified reward function $G_i(s,a) = R_i(s,a) - \lambda \cdot C_i(s,a)$, and therefore we refer to this value function as $\bar{G}_i^{\pi_i}$. A lower bound on the expected value $G_i^{\pi_i}(b)$ of belief b can be expressed as a function of the vectors in Γ:

$$\bar{G}_i^{\pi_i}(b) = \max_{\alpha \in \Gamma} b \cdot \alpha. \tag{21}$$

Note that the actual expected value $G_i^{\pi_i}(b)$ represents the exact expected value with respect to G_i while executing π_i starting from b. An appealing property of the point-based algorithms HSVI (Smith & Simmons, 2005), SARSOP (Kurniawati et al., 2008) and GapMin (Poupart et al., 2011) is that they also compute an upper bound $\hat{G}_i$ on the optimal value function. For each belief b it holds that:

$$\bar{G}_i^{\pi_i}(b) \leq G_i^{\pi_i}(b) \leq \hat{G}_i^{\pi_i}(b). \tag{22}$$

It can be shown that the lower bound and upper bound coincide in the limit, which means that HSVI, SARSOP and GapMin deliver an optimal value function in the limit.

Unfortunately, the aforementioned algorithms consider infinite-horizon POMDPs which include discounting of reward, and these algorithms cannot be used directly to solve finite-horizon subproblems without discounting during the execution of column generation. A straightforward approach for modeling a finite horizon would be augmenting the model with time-indexed states and a trap state, but this creates an excessively large POMDP model and the vectors will have many entries. Moreover, it is required to eliminate the discount factor by assuming a discount factor of 1, which leads to several problems in state-of-the-art algorithms. For example, GapMin requires a discount factor that is strictly smaller than 1 in several subroutines, and in SARSOP and HSVI the initialization of upper bounds requires the discount factor to be smaller than 1. This means that the algorithms require significant modifications before they can be used for problems without discounting. Other algorithm such as Perseus (Spaan & Vlassis, 2005) and PBVI (Pineau et al., 2003) do not provide performance guarantees, and they are generally outperformed by more recent algorithms. We conclude that computing solutions to finite-horizon problems requires tailored algorithms which do not include discounting, and they should account only for a finite number of time steps.

In Algorithm 2 we present a tailored point-based value iteration algorithm for finite-horizon POMDPs, which repeatedly computes value functions for all time steps (lines 7-28), and it incrementally expands the sets containing belief points based on heuristic search (line 6). For each time step t the algorithm keeps track of a vector set Γ_t, and B_t contains pairs $(b, \bar{v})$ which represent a belief point b and an associated value upper bound $\bar{v}$. On lines 9-13 the algorithm computes a new vector set Γ_t using backups based on the belief points in B_t. Next, it updates the value upper bounds $\bar{v}$ for each $(b, \bar{v}) \in B_t$ on lines 14-27. At the end of an iteration it computes the current value lower bound g_l and upper bound g_u, which together define the current value gap g. The algorithm terminates if the time limit τ has been exceeded, or in case the gap is at most one unit at the ρ-th significant digit.

The call to $\texttt{Backup}(b, t, r)$ computes a new alpha vector based on a given belief point b and the value function of the next time step $t+1$. Note that r represents a function defining the immediate reward vectors for each action, as defined on line 2. The backup function is formalized as follows:

$$\texttt{Backup}(b, t, r) = \underset{\{z_{b,a,t}\}_{a \in A_i}}{\arg\max}\ b \cdot z_{b,a,t} \tag{23}$$

where

$$z_{b,a,t} = \begin{cases} r(a) + \sum_{o \in O_i} \arg\max_{\{z_{a,o}^{k,t+1}\}_k} b \cdot z_{a,o}^{k,t+1} & t < h \\ r(a) & t = h \end{cases}, \tag{24}$$

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

$$z_{a,o}^{k,t}(s) = \sum_{s' \in S_i} P(o|a, s') P(s'|s, a) \alpha^{k,t}(s') \quad \forall s \in S_i. \tag{25}$$

Algorithm 2: Point-based value iteration for finite-horizon POMDPs (`PointBased`)

```
input  : POMDP $M_i$ and function $G_i$, precision $\rho$, time limit $\tau$
output: set $\Gamma_t$ for each time step $t$, upper bound $\hat{G}_i^{\pi_i}$
$\Gamma_t \leftarrow \emptyset \quad \forall t, \quad B_t \leftarrow \emptyset \quad \forall t$
$r(a) \leftarrow (G(s_1, a), G(s_2, a), \ldots, G(s_{|S_i|}, a)) \quad \forall a \in A_i$
add corner beliefs to $B_t$ with upper bound $\infty$, for each time step $t$
$\tau' \leftarrow 0, \quad g \leftarrow \infty, \quad g_a \leftarrow 0$
do
    ExpandBeliefs($M_i, \{\Gamma_1, \ldots, \Gamma_h\}, \{B_1, \ldots, B_h\}, r$)
    for $t = h, h-1, \ldots, 1$ do
        $\Gamma_t \leftarrow \emptyset$
        for $(b, \bar{v}) \in B_t$ do
            $\alpha \leftarrow$ Backup($b, t, r$)
            $\alpha_b \leftarrow b$
            $\Gamma_t \leftarrow \Gamma_t \cup \{\alpha\}$
        end
        for $(b, \bar{v}) \in B_t$ do
            $\bar{v} \leftarrow -\infty$
            for $a \in A_i$ do
                $v \leftarrow r(a) \cdot b$
                if $t < h$ then
                    for $o \in O_i$ do
                        if $P(o|b, a) > 0$ then
                            $v \leftarrow v + P(o|b, a) \cdot$ UpperBound($b_a^o, B_{t+1}$)
                        end
                    end
                end
                $\bar{v} \leftarrow \max(\bar{v}, v)$
            end
        end
    end
    $g_l \leftarrow \max_{\alpha \in \Gamma_1} \alpha \cdot b_{1,i}$
    $g_u \leftarrow$ upper bound $\bar{v}$ associated with $(b_{1,i}, \bar{v}) \in B_1$
    $g \leftarrow g_u - g_l$
    $g_a \leftarrow 10^{\lceil \log_{10}(\max(|g_l|, |g_u|)) \rceil - \rho}$
    $\tau' \leftarrow$ elapsed time after the start of the algorithm
while $\tau' < \tau \ \wedge \ g > g_a$;
return $(\{\Gamma_1, \ldots, \Gamma_h\}, g_u)$
```

Algorithm 3: Belief expansion algorithm (`ExpandBeliefs`)

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

Note that the backup operator is equivalent to the default backup operator found in point-based methods for infinite-horizon POMDPs. However, it takes into account the finite time horizon by keeping track of time-dependent vector sets.

The function `UpperBound`(b, B_t) computes an upper bound on the expected value corresponding to belief b, based on the pairs in B_t. This upper bound interpolation can be computed using a linear program, but typically a sawtooth approximation is used (Hauskrecht, 2000). This approximation is cheap to compute and it is also used by state-of-the-art algorithms SARSOP, HSVI and GapMin. A full description of the sawtooth approximation can be found in Appendix A.

The performance and convergence of the algorithm are determined by the strategy that is used to find additional belief points in the function `ExpandBeliefs`. Ideally, we would want to add belief points that are reachable during the execution of an optimal policy. We use a belief search procedure that is inspired by both HSVI and GapMin, as shown in Algorithm 3. For each time step it first selects the action a with the highest upper bound, which ensures that actions are tried until they become suboptimal. After selecting an action the algorithm proceeds with selecting an observation o which leads to a belief with the largest gap between lower bound and upper bound. Such beliefs have the potential to improve the solution quality the most within an iteration. Notice that beliefs are added to B_{t+1}, and therefore the algorithm iterates until step $h-1$.

Our tailored point-based algorithm computes vector sets for each time step. We can express the lower bound on the value of belief b in terms of the vectors in Γ_1:

$$\bar{G}_i^{\pi_i}(1, b) = \max_{\alpha \in \Gamma_1} b \cdot \alpha. \tag{26}$$

Since we are particularly interested in the value associated with the initial belief $b_{1,i}$, the algorithm computes a lower bound $g_l = \bar{G}_i^{\pi_i}$ and upper bound $g_u = \hat{G}_i^{\pi_i}$ on $G_i^{\pi_i}$, for which it holds that $\bar{G}_i^{\pi_i} \leq G_i^{\pi_i} \leq \hat{G}_i^{\pi_i}$. In Section 4.4 we use these bounds in our adapted version of the column generation algorithm.

4.2 Policy Graphs as Policy Representation

Computing value functions using point-based value iteration is relatively efficient compared to exact value iteration. However, given a policy π_i induced by vector sets $\Gamma_1, \ldots, \Gamma_h$, it is computationally difficult to obtain the expectations $V_i^{\pi_i}$ and $C_i^{\pi_i}$. It requires evaluation of

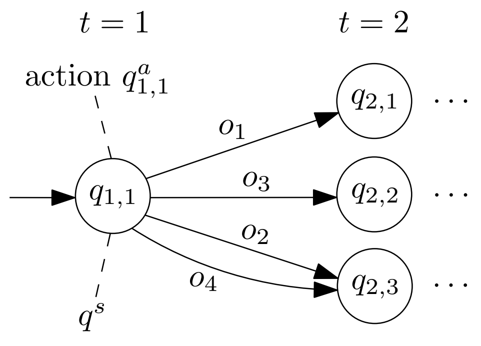


Figure 2: Policy graph example

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 et al., 1998; Hansen, 1998; Poupart & 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 in this paper 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 $q^a_{t,j} \in A_i$, and after receiving observation $o \in O_i$ the controller node transitions deterministically to node $q^o_{t,j} \in \mathcal{G}$. This means that $q^o_{t,j}$ 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 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 $q^a_{1,1} \in A_i$. 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 $q^{o_1}_{1,1} = q_{2,1}$, $q^{o_2}_{1,1} = q_{2,3}$, $q^{o_3}_{1,1} = q_{2,2}$ 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.

4.3 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 & Boutilier, 2003; Grześ, Poupart, & Hoey, 2013; Amato, Bernstein, & 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 & 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 $\Gamma_1, \ldots, \Gamma_h$ 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, Poupart, Yang, & Hoey, 2015). Algorithm 4 shows how the alpha vectors $\Gamma_1, \ldots, \Gamma_h$ 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 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 4.5.

A convenient property is that we can evaluate the quality of the policy graph using a recurrence. We let $\mathcal{V}_R(q_{t,j}, s)$ 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:

$$\mathcal{V}_R(q_{t,j}, s) = \begin{cases} R_i(s, q^a_{t,j}) + \sum_{o \in o_i, s' \in S_i} P(s' \mid s, q^a_{t,j}) P(o \mid q^a_{t,j}, s') \mathcal{V}_R(q^o_{t,j}, s') & t < h \\ R_i(s, q^a_{t,j}) & t = h \end{cases}. \quad (27)$$

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

$$V_i^{\pi_i} = \sum_{s \in S_i} \mathcal{V}_R(q^s, s) \cdot b_{1,i}(s), \quad (28)$$

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:

$$\mathcal{V}_C(q_{t,j}, s) = \begin{cases} C_i(s, q^a_{t,j}) + \sum_{o \in o_i, s' \in S_i} P(s' \mid s, q^a_{t,j}) P(o \mid q^a_{t,j}, s') \mathcal{V}_C(q^o_{t,j}, s') & t < h \\ C_i(s, q^a_{t,j}) & t = h \end{cases}. \quad (29)$$

Algorithm 4: Generating a policy graph from alpha vectors (`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          $q_{j,t}^a \leftarrow a$
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                     $k \leftarrow \arg\max_{\{\alpha^k \in \Gamma_{t+1}\}_k} \alpha^k \cdot b_a^o$
13                     $q_{j,t}^o \leftarrow q_{t+1,k}$
14                 else
15                     $q_{j,t}^o \leftarrow q_{t+1,1}$
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 $(\mathcal{G}, q^s)$
```

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

$$C_i^{\pi_i} = \sum_{s \in S_i} \mathcal{V}_C(q^s, s) \cdot b_{1,i}(s). \tag{30}$$

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 4.5. In the next section we first describe how the point-based algorithms and policy graphs are integrated in the column generation algorithm.

4.4 Adapted Column Generation Algorithm

Exact column generation in Algorithm 1 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 1 is no longer valid because computed policies do not necessarily maximize reduced cost. Lemma 2 is still valid because we are always able to determine the reduced cost $\bar{c}_{\pi_i}$ of a new policy. Lemma 3 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 4 is no longer valid. Lemma 5 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 1, 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 point-based algorithms 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, 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 the point-based algorithms 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 point-based 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 10 is no longer valid since it is not guaranteed that the point-based algorithm finds the maximizing policy $\pi_i \in K_i$. However, given the upper bound $\hat{G}_i^{\pi_i}$ computed by the point-based algorithm we derive:

$$\phi_u = \lambda \cdot L + \sum_{i=1}^{n} \left[\max_{\pi_i \in K_i} \left(V_i^{\pi_i} - \lambda \cdot C_i^{\pi_i} \right) \right] \leq \lambda \cdot L + \sum_{i=1}^{n} \hat{G}_i^{\pi_i}. \tag{31}$$

Note that the upper bound $\hat{G}_i^{\pi_i}$ is denoted by the variable g_u in our point-based algorithm. 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,

Algorithm 5: Adapted column generation (CGCP)

```
input : POMDP $M_i$ $\forall i$, time limit $\mathcal{T}$, point-based 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         $G_i(s,a) \leftarrow R_i(s,a) - \lambda \cdot C_i(s,a)\quad \forall s \in S_i,\ a \in A_i$
19         $(\Gamma_1, \dots, \Gamma_h, \hat{G}_i^{\pi_i}) \leftarrow$ PointBased($M_i, G_i, \rho, \tau$)
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 27-30
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 $x_{i,\pi_i} > 0\}\quad \forall i$
30 return $\{Y_1, \dots, Y_n\}$
```

it can be expected that the quality of the upper bound becomes better once the point-based algorithms run longer during later stages of the column generation algorithm.

In Algorithm 5 we present the modified Column Generation algorithm for Constrained POMDPs, which we call CGCP. On line 19 the algorithm invokes a point-based algorithm with 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 4. After policy evaluation using a recurrence, the policy can be added and the upper bound ϕ_u is updated according to

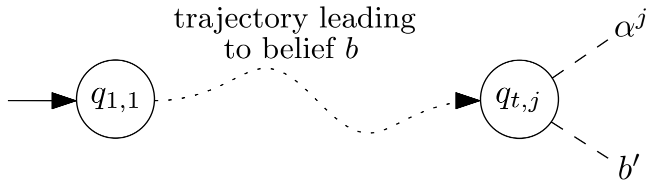


Figure 3: Graph trajectory leading to a node $q_{t,j}$

Equation 31. 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 Algorithm 2. 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.

4.5 Analysis of the Policy Graph Construction

In this section we provide an additional analysis of the translation of vectors $\Gamma_1, \ldots, \Gamma_h$ into a policy graph $\mathcal{G}$. This translation is not exact, which means that the solution quality of the policy induced by $\Gamma_1, \ldots, \Gamma_h$ is not necessarily the same as the solution quality of $\mathcal{G}$. This can be explained as follows. Algorithm 4 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 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 the point-based algorithm 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. The point-based value iteration algorithm solves a subproblem based on the modified reward function G_i, and in Lemma 1 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 1). For the corresponding policy graph $\mathcal{G}$ we can define the actual reduced cost as:

$$\left(\sum_{s \in S_i} \mathcal{V}_G(q^s, s) \cdot b_{1,i}(s) \right) - \lambda_i, \tag{32}$$

where

$$\mathcal{V}_G(q_{t,j}, s) = \begin{cases} G_i(s, q^a_{t,j}) + \sum_{o \in o_i, s' \in S_i} P(s' \mid s, q^a_{t,j}) P(o \mid q^a_{t,j}, s') \mathcal{V}_G(q^o_{t,j}, s') & t < h \\ G_i(s, q^a_{t,j}) & t = h \end{cases}. \quad (33)$$

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:

$$\left(\sum_{s \in S_i} \mathcal{V}_G(q^s, s) \cdot b_{1,i}(s) \right) - \lambda_i - \left(\bar{G}^{\pi_i}_i - \lambda_i \right) \quad (34)$$

$$= \left(\sum_{s \in S_i} \mathcal{V}_G(q^s, s) \cdot b_{1,i}(s) \right) - \bar{G}^{\pi_i}_i, \quad (35)$$

which can be positive as well as negative. The expression in Equation 35 enables us to measure the change in policy quality after translating the vectors $\Gamma_1, \ldots, \Gamma_h$ 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 $\Gamma_1, \ldots, \Gamma_h$. 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 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 6. 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 4, 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 33, 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.

Lemma 6. *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 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 $t = h, h-1, \ldots, 1$. 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 23, which yields the immediate reward vector for a value-maximizing action.

Algorithm 6: Get the probability that the action defined by $\mathcal{G}$ and $\Gamma_1, \ldots, \Gamma_h$ deviates

```
input  : POMDP model $M_i$, alpha vectors in a sets $\Gamma_1, \ldots, \Gamma_h$, graph $\mathcal{G}$
output: probability $\mathcal{P}$
1  $\mathcal{X}_t \leftarrow \emptyset \quad \forall t, \qquad \mathcal{P} \leftarrow 0$
2  $\mathcal{X}_1 \leftarrow \mathcal{X}_1 \cup \{(q^s, b_{1,i}, 1)\}$
3  for $t = 1, \ldots, h$ do
4      for $(q_{t,j}, b, p) \in X_t$ do
5          $\alpha^j \leftarrow$ vector $\alpha \in \Gamma_t$ corresponding to $q_{t,j}$
6          $b' \leftarrow$ belief using which $\alpha^j$ was generated
7          $a' \leftarrow$ action associated with $\arg\max_{\alpha \in \Gamma_t} \alpha \cdot b$
8          $a \leftarrow q^a_{t,j}$
9          if $b \neq b' \wedge a \neq a'$ then
10             $\mathcal{P} \leftarrow \mathcal{P} + p$
11         else if $t < h$ then
12             for $o \in O_i$ do
13                 if $P(o \mid b, a) > 0$ then
14                     $\mathcal{X}_{t+1} \leftarrow \mathcal{X}_{t+1} \cup \{(q^a_{t,j}, b^o_a, p \cdot P(o \mid b, a))\}$
15                 end
16             end
17         end
18     end
19 end
20 return $\mathcal{P}$
```

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 33 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 the relation $b \cdot \mathcal{V}_G(q_{t,j}, \cdot) = b \cdot \alpha_{q_{t,j}}$, where b is the belief using which both $q_{t,j}$ and $\alpha_{q_{t,j}}$ were generated. The full derivation is provided in Appendix B. Based on the principle of induction we conclude that the theorem holds for all time steps $t = 1, 2, \ldots, h$. □

Theorem 4. *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 1 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 6. The final term in the derivation is identical to the exact reduced cost of the graph-based policy, as defined in Equation 32, and therefore we can conclude that the reduced cost of both representations is identical. $\square$

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 4 provides an exact translation from vectors to a policy graph.

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 the performance of 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 in Appendix C. 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.

5.1 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, which is not desirable in our experiments. A full description of the modifications to the benchmark domains is discussed in Appendix D.

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

			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 1: Comparison of CGCP and CALP on navigation domains

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. In addition, we impose the upper bound L on the expected number of `MOVE` actions.

Table 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 in the table 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.

5.2 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 et al., 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 & Ding, 2010), railway equipment (Fararooy & Allan, 1995), bridges (Neves & 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, & 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

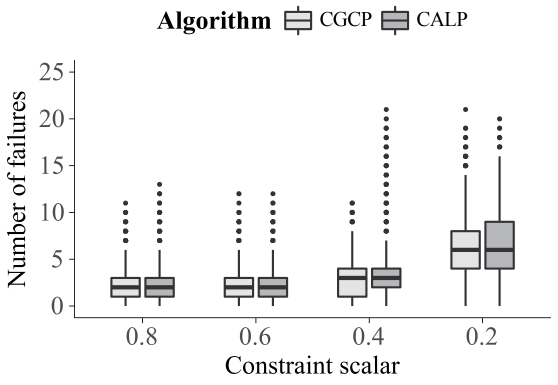
n	L_m	L	R	Gap	Std reward	C	Std cost	Time (s)
2	0.8	667.23	17001.14	6.95	797.21	667.23	118.89	484
	0.6	500.42	16861.91	13.81	810.76	500.42	68.41	362
	0.4	333.61	16437.60	39.26	870.69	333.65	67.16	485
	0.2	166.81	14920.06	77.21	1066.55	166.83	37.29	484
3	0.8	1074.75	25533.55	80.56	965.99	1074.75	146.78	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 2: Performance of CGCP on maintenance instances

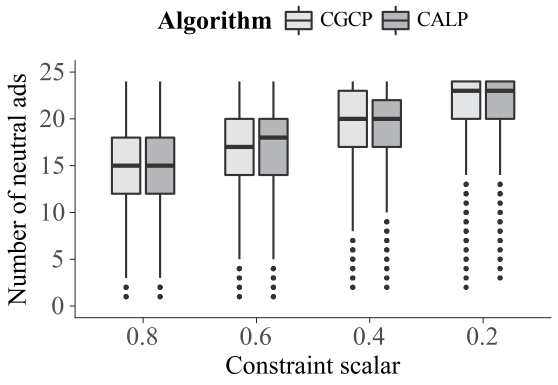
state transition characteristics defining the deterioration process. More details regarding the domain can be found in Appendix D.

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 the point-based solver 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 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



(a) Bridge failures



(b) Neutral advertising

Figure 4: Behavior of policies during simulation

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 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 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.

5.3 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 & 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

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 3: Performance of CGCP on advertising instances

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, Theocharous, Vlassis, De Weerdt, & Spaan, 2018).

We can use our Multi-agent Constrained POMDP model to formalize an online 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 the Appendix D.

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 point-based solver 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 3 for $n \in \{2, 3, \dots, 6\}$ 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 indicates 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 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.

5.4 Translation of Vectors into Policy Graph

In Section 4.5 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 35 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 Algorithm 2. 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. Each dot in the 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 some 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

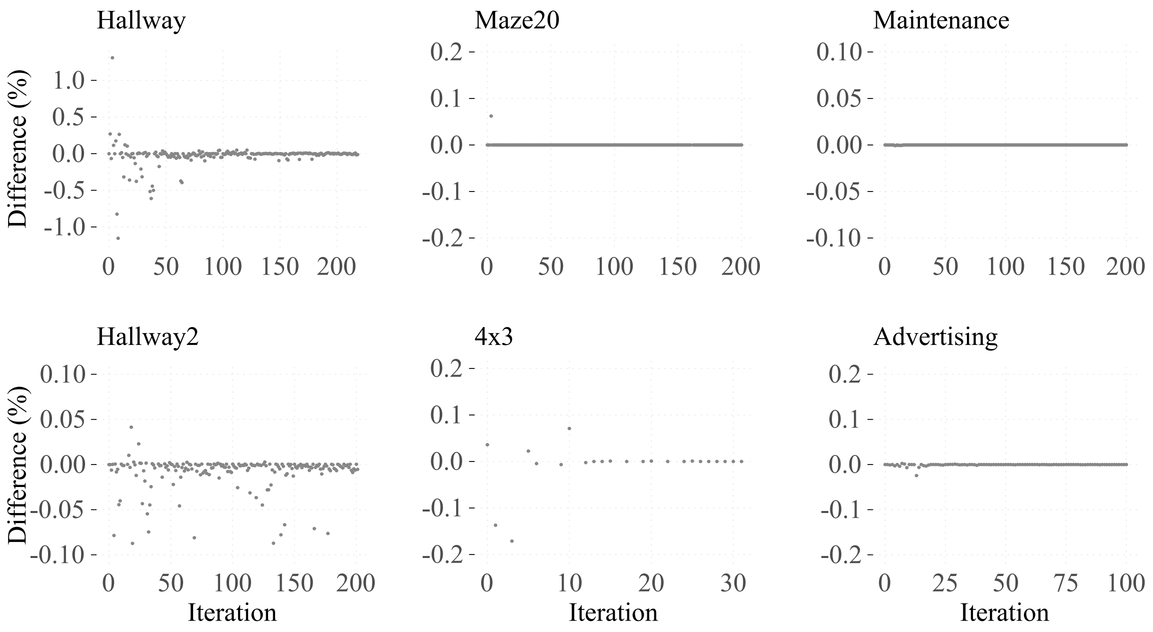


Figure 5: Lower bound difference as a function of the number of iterations

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 6, 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 6 may be used in small domains for additional assessment of the policy quality change.

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 et al., 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 et al., 1997; Walraven & 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 domains. 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 & Durfee, 2010), and its tractability can be increased by using a Lagrangian relaxation (Agrawal et al., 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, Walraven, De Weerdt, & Spaan, 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 in this paper, 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 & Boutilier, 2004), mixed-integer programming (Kumar & Zilberstein, 2015) and branch-and-bound techniques (Grześ et al., 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 et al., 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 column generation.

There are relations between Decentralized POMDPs (Oliehoek & Amato, 2016) and the multi-agent model considered in this paper. 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.

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 presented a tailored point-based algorithm for finite-horizon POMDPs, 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,

Algorithm 7: Computing upper bound using sawtooth approximation (`UpperBound`)

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

enforcing constraints on the actual cost incurred during policy execution, rather than the expected cost, remains an open challenge.

Acknowledgments

The research presented in this paper is supported by the Netherlands Organisation for Scientific Research (NWO), as part of the Uncertainty Reduction in Smart Energy Systems program. We would like to thank Frits de Nijs for several fruitful discussions.

Appendix A. Sawtooth Approximation

Our algorithms use a sawtooth approximation (Hauskrecht, 2000) to find a value upper bound for a given belief b, which is computed using an interpolation based on belief-bound pairs in a given set B. A full description of the procedure is shown in Algorithm 7, in which e_s denotes the corner belief corresponding to state $s \in S$. We use $B(e_s)$ to refer to the value upper bound $\bar{v}$ defined by the pair $(e_s, \bar{v}) \in B$. Our pseudocode closely resembles the description provided by Poupart et al. (2011). Additional details regarding the approximation are provided by Shani, Pineau, and Kaplow (2013).

Appendix B. Proof of Lemma 6

Lemma 6. *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 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 $t = h, h-1, \ldots, 1$. 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 23, 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 33 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.

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

$$= b \cdot \texttt{backup}(b) \qquad \text{backup using } G(s,a)$$
$$= b \cdot \alpha_q \qquad \alpha_q \text{ generated with } b$$

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

It is assumed that Algorithm 4 breaks ties on line 12 by selecting the value-maximizing 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 4 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 $b_{q^a}^o \cdot \mathcal{V}_G(q^o, \cdot) = b_{q^a}^o \cdot \alpha_{q^o}$, following our induction hypothesis.

All other derivation steps follow from definitions, the description in Algorithm 4, 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 $t = 1, 2, \ldots, h$. □

Appendix C. 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:

$$\begin{aligned}
\max \quad & \sum_{t=1}^{h} \sum_{s \in S} \sum_{a \in A} x_{t,s,a} \cdot R(s,a) \\
\text{s.t.} \quad & \sum_{a' \in A} x_{t+1,s',a'} = \sum_{s \in S} \sum_{a \in A} x_{t,s,a} \cdot P(s'|s,a) \quad \forall s' \in S, \quad \forall t = 1, \ldots, h \\
& \sum_{a \in A} x_{1,s,a} = P(s_0 = s) \qquad \forall s \in S \\
& \sum_{t=1}^{h} \sum_{s \in S} \sum_{a \in A} x_{t,s,a} \cdot C(s,a) \leq L.
\end{aligned} \tag{36}$$

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.

Appendix D. 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 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 the `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 online advertising problems we use the web-ad domain provided on `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 associated with it.

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

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 4: Size of the benchmark domains used in experiments

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 again such that the probabilities sum to 1. Note that this procedure ensures that the reachability of states is not affected.

References

Agrawal, P., Varakantham, P., & Yeoh, W. (2016). Scalable Greedy Algorithms For Task / Resource Constrained Multi-Agent Stochastic Planning. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence*, pp. 10–16.

Altman, E. (1999). *Constrained Markov Decision Processes.* CRC Press.

Amato, C., Bernstein, D. S., & Zilberstein, S. (2010). Optimizing fixed-size stochastic controllers for POMDPs and decentralized POMDPs. *Journal of Autonomous Agents and Multi-Agent Systems*, *21*(3), 293–320.

Boutilier, C., & Lu, T. (2016). Budget Allocation using Weakly Coupled, Constrained Markov Decision Processes. In *Proceedings of the 32nd Conference on Uncertainty in Artificial Intelligence*, pp. 52–61.

Bradley, S., Hax, A., & Magnanti, T. (1977). *Applied Mathematical Programming.* Addison-Wesley.

Braziunas, D., & Boutilier, C. (2004). Stochastic Local Search for POMDP Controllers. In *Proceedings of the 19th AAAI Conference on Artificial Intelligence*, pp. 690–696.

Byon, E., & Ding, Y. (2010). Season-dependent condition-based maintenance for a wind turbine using a partially observed markov decision process. *IEEE Transactions on Power Systems*, *25*(4), 1823–1834.

Cassandra, A., Littman, M. L., & Zhang, N. L. (1997). Incremental Pruning: A Simple, Fast, Exact Method for Partially Observable Markov Decision Processes. In *Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence.*

Cassandra, A. R. (1998). *Exact and approximate algorithms for partially observable Markov decision processes.* Ph.D. thesis, Brown University.

Dantzig, G. (1963). *Linear programming and extensions.* Princeton University Press.

De Nijs, F., Spaan, M. T. J., & De Weerdt, M. M. (2015). Best-Response Planning of Thermostatically Controlled Loads under Power Constraints. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, pp. 615–621.

De Nijs, F., Theocharous, G., Vlassis, N., De Weerdt, M., & Spaan, M. T. J. (2018). Capacity-aware Sequential Recommendations. In *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems*.

De Nijs, F., Walraven, E., De Weerdt, M. M., & Spaan, M. T. J. (2017). Bounding the Probability of Resource Constraint Violations in Multi-Agent MDPs. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, pp. 3562–3568.

Desrosiers, J., & Lübbecke, M. E. (2005). *A primer in column generation*. Kluwer.

Dolgov, D., & Durfee, E. H. (2003). Approximating Optimal Policies for Agents with Limited Execution Resources. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pp. 1107–1112.

Ellis, H., Jiang, M., & Corotis, R. B. (1995). Inspection, maintenance, and repair with partial observability. *Journal of Infrastructure Systems*, *1*(2), 92–99.

Fararooy, S., & Allan, J. (1995). Condition-based maintenance of railway signalling equipment. In *International Conference on Electric Railways in a United Europe*, pp. 33–37.

Gilmore, P. C., & Gomory, R. E. (1961). A Linear Programming Approach to the Cutting-Stock Problem. *Operations research*, *9*(6), 849–859.

Grześ, M., Poupart, P., & Hoey, J. (2013). Isomorph-Free Branch and Bound Search for Finite State Controllers. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, pp. 2282–2290.

Grześ, M., Poupart, P., Yang, X., & Hoey, J. (2015). Energy Efficient Execution of POMDP Policies. *IEEE Transactions on Cybernetics*, *45*(11), 2484–2497.

Hansen, E. A. (1998). Solving POMDPs by Searching in Policy Space. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, pp. 211–219.

Harman, R. M. (2002). Wireless solutions for aircraft condition based maintenance systems. In *Proceedings of the IEEE Aerospace Conference*, pp. 6–2877 – 6–2886.

Hauskrecht, M. (2000). Value-Function Approximations for Partially Observable Markov Decision Processes. *Journal of Artificial Intelligence Research*, *13*, 33–94.

Isom, J. D., Meyn, S. P., & Braatz, R. D. (2008). Piecewise Linear Dynamic Programming for Constrained POMDPs. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*, pp. 291–296.

Jardine, A. K., Lin, D., & Banjevic, D. (2006). A review on machinery diagnostics and prognostics implementing condition-based maintenance. *Mechanical Systems and Signal Processing*, *20*(7), 1483–1510.

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, *101*(1), 99–134.

Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming. In *Proceedings of the 16th Annual ACM symposium on Theory of Computing*, pp. 302–311. ACM.

Kim, D., Lee, J., Kim, K., & Poupart, P. (2011). Point-Based Value Iteration for Constrained POMDPs. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pp. 1968–1974.

Kim, J. W., Choi, G. B., & Lee, J. M. (2018). A POMDP framework for integrated scheduling of infrastructure maintenance and inspection. *Computers & Chemical Engineering*, *112*, 239–252.

Kumar, A., & Zilberstein, S. (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.

Kurniawati, H., Hsu, D., & Lee, W. S. (2008). SARSOP : Efficient Point-Based POMDP Planning by Approximating Optimally Reachable Belief Spaces. In *Robotics: Science and Systems IV*.

Neves, L. C., & Frangopol, D. M. (2005). Condition, safety and cost profiles for deteriorating structures with emphasis on bridges. *Reliability Engineering & System Safety*, *89*(2), 185 – 198.

Oliehoek, F. A., & Amato, C. (2016). *A Concise Introduction to Decentralized POMDPs*. Springer.

Papadimitriou, C. H., & Tsitsiklis, J. N. (1987). The complexity of Markov decision processes. *Mathematics of Operations Research*, *12*(3), 441–450.

Papadimitriou, C. H., & Steiglitz, K. (1982). *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall.

Pineau, J., Gordon, G., & Thrun, S. (2003). Point-based value iteration: An anytime algorithm for POMDPs. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pp. 1025–1030.

Poupart, P., & Boutilier, C. (2003). Bounded Finite State Controllers. In *Advances in Neural Information Processing Systems*, pp. 823–830.

Poupart, P., Kim, K., & Kim, D. (2011). Closing the Gap: Improved Bounds on Optimal POMDP Solutions. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling*, pp. 194–201.

Poupart, P., Malhotra, A., Pei, P., Kim, K., Goh, B., & Bowling, M. (2015). Approximate Linear Programming for Constrained Partially Observable Markov Decision Processes. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, pp. 3342–3348.

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

Roijers, D. M., Vamplew, P., Whiteson, S., & Dazeley, R. (2013). A Survey of Multi-Objective Sequential Decision-Making. *Journal of Artificial Intelligence Research*, *48*, 67–113.

Shani, G., Pineau, J., & Kaplow, R. (2013). A survey of point-based POMDP solvers. *Autonomous Agents and Multi-Agent Systems*, *27*(1), 1–51.

Smith, T., & Simmons, R. (2005). Point-Based POMDP Algorithms: Improved Analysis and Implementation. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pp. 542–549.

Sondik, E. J. (1971). *The optimal control of partially observable Markov processes.* Ph.D. thesis, Stanford University.

Spaan, M. T. J., & Vlassis, N. (2005). Perseus: Randomized Point-based Value Iteration for POMDPs. *Journal of Artificial Intelligence Research*, *24*, 195–220.

Walraven, E., & Spaan, M. T. J. (2017). Accelerated Vector Pruning for Optimal POMDP Solvers. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, pp. 3672–3678.

Wu, J., & Durfee, E. H. (2010). Resource-Driven Mission-Phasing Techniques for Constrained Agents in Stochastic Environments. *Journal of Artificial Intelligence Research*, *38*, 415–473.

Yost, K. A., & Washburn, A. R. (2000). The LP/POMDP Marriage: Optimization with Imperfect Information. *Naval Research Logistics*, *47*(8), 607–619.