Bounding the probability of resource constraint violations in multi-agent MDPs

De Nijs, Frits; Walraven, Erwin; De Weerdt, Mathijs M.; Spaan, Matthijs T.J.

Publication date
2017

Document Version
Accepted author manuscript

Published in
Proceedings of the 31st Conference on Artificial Intelligence, AAAI 2017

Citation (APA)
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 Conference on Artificial Intelligence, AAAI 2017* (pp. 3562-3568). American Association for Artificial Intelligence (AAAI). https://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/15009/14155

Bounding the Probability of Resource Constraint Violations in Multi-Agent MDPs

Frits de Nijs and **Erwin Walraven** and **Mathijs M. de Weerdt** and **Matthijs T. J. Spaan**
Delft University of Technology
Mekelweg 4, 2628 CD
Delft, The Netherlands

Abstract

Multi-agent planning problems with constraints on global resource consumption occur in several domains. Existing algorithms for solving Multi-agent Markov Decision Processes can compute policies that meet a resource constraint in expectation, but these policies provide no guarantees on the probability that a resource constraint violation will occur. We derive a method to bound constraint violation probabilities using Hoeffding's inequality. This method is applied to two existing approaches for computing policies satisfying constraints: the Constrained MDP framework and a Column Generation approach. We also introduce an algorithm to adaptively relax the bound up to a given maximum violation tolerance. Experiments on a hard toy problem show that the resulting policies outperform static optimal resource allocations to an arbitrary level. By testing the algorithms on more realistic planning domains from the literature, we demonstrate that the adaptive bound is able to efficiently trade off violation probability with expected value, outperforming state-of-the-art planners.

Introduction

When decision-making agents share collectively owned resources, their actions need to be coordinated subject to the availability of these resources. In many problem domains it is impractical, inefficient or costly to coordinate resource consumption during execution. For example, load balancing of energy consumption in smart energy grids has instantaneous effects on the stability of the grid, which requires real-time decisions. Other examples are rescue operations or military campaigns, during which communication options may be limited by the hostile environment.

Before execution, it is possible to merge the decision-making problems of individual agents into a joint resource-constrained optimization problem. However, solving this problem quickly becomes intractable because the size of the joint problem grows exponentially in the number of agents. Wu and Durfee (2010) show that it is more efficient to optimize an off-line deterministic allocation of resources to agents. A scalable Lagrangian relaxation of this optimization problem is presented by Agrawal, Varakantham, and Yeoh (2016). Unfortunately, when agents are operating in an uncertain environment, such an allocation may leave a significant portion of the available resources unused.

When policies of agents are allowed to satisfy resource constraints *in expectation* (such as with accidental overruns of budget, or infrastructural constraints), this weakness of deterministic resource allocations can be overcome, resulting in significantly higher expected value. Column Generation for linear programs (LPs) and Constrained Markov Decision Processes (CMDPs) can be used to compute such conditional resource allocations by demanding that the total expected resource consumption of the agents is not more than the resource availability (Yost and Washburn 2000; Altman 1999). However, the resulting policies satisfy the constraints only in expectation, and constraint violations may occur at execution time. There is no guarantee on the probability of violations occurring, which makes it unattractive when only a limited amount of violations can be tolerated.

In this paper we demonstrate that the probability of constraint violation can be bounded by using Hoeffding's inequality (1963) to derive a reduced constraint. By planning using this reduced constraint, we obtain safe conditional policies for the real constraint. Because this bound turns out to be quite conservative, we also propose to iteratively relax it, resulting in conditional resource allocations with higher expected value that still respect the constraint violation tolerance. We show that Column Generation for LPs has several attractive properties compared to CMDPs when using the constraint relaxation. We further improve the scalability of this method by introducing a column pruning technique that significantly reduces the number of columns considered per iteration, while preserving convergence characteristics.

When communication between agents is available, resource conflicts can be arbitrated at policy execution time (Meuleau et al. 1998; De Nijs, Spaan, and De Weerdt 2015). Existing deterministic preallocation methods that work when communication is not available only support binary resource consumption functions (Wu and Durfee 2010; Agrawal, Varakantham, and Yeoh 2016). Our method naturally supports multi-unit resource consumption, and requires no communication between agents during execution. It can be seen as an anytime algorithm for computing policies satisfying a given violation tolerance.

We evaluate our algorithm against two state-of-the-art deterministic resource allocation methods (Wu and Durfee

2010; Agrawal, Varakantham, and Yeoh 2016). We compare on a hard artificial problem and on more realistic domains from literature: power-constrained planning (De Nijs, Spaan, and De Weerdt 2015), Mars rovers (Wu and Durfee 2010), and advertising (Boutilier and Lu 2016). The hard problem demonstrates that deterministic resource allocations may perform arbitrarily worse than conditional allocations. On the realistic domains our constraint relaxation in combination with Column Generation and pruning is the only algorithm capable of scaling to the largest instances while still providing guarantees on the probability of violations.

MMDPs with Global Resource Constraints

We consider finite-horizon Constrained Multi-agent Markov Decision Processes, which consist of n independent agents, each modeled as a Markov Decision Process (MDP, Bellman 1957), and a set of k global resource constraints. Our model generalizes earlier models of Constrained Multi-agent MDPs (Boutilier and Lu 2016; De Nijs, Spaan, and De Weerdt 2015) because it allows for modeling two types of constraints: instantaneous and budget constraints. Instantaneous constraints must be respected at any point in time, such as the capacity limits of energy grids. Budget constraints define a global resource budget that is defined over multiple time steps, which occurs in money investment problems.

The tuple $\langle S_i, A_i, T_i, R_i, h, s^{i,1} \rangle$ defines the MDP M_i for agent i. Each agent has its own sets of states S_i and actions A_i. The transition function $T_i : S_i \times A_i \times S_i \to [0, 1]$ gives the probability of advancing to state $s' \in S_i$ from state $s \in S_i$ by choosing action $a \in A_i$, thus $T_i(s, a, s') = P(s' \mid s, a)$. The choice of action a in state s is rewarded through reward function $R_i : S_i \times A_i \to \mathbb{R}$. The horizon h specifies the total number of decisions and the initial state of agent i is defined by $s^{i,1}$. A solution to the multi-agent planning problem takes the form of a policy $\pi_i : \{1, \ldots, h\} \times S_i \to A_i$ for each agent i, which can be used by the individual agents to select their actions. Optimal policies in the finite-horizon case are non-stationary, and therefore they should be time-dependent. The policies should maximize the total expected sum of rewards of the agents:

$$\sum_{i=1}^{n} \left(E_{\pi_i} \left[\sum_{t=1}^{h} R_i(s_{i,t}, \pi_i(t, s_{i,t})) \,\middle|\, s_{i,1} = s^{i,1} \right] \right), \quad (1)$$

where $s_{i,t}$ denotes the state of agent i at time t. This expression is also known as the total expected value of the agents.

Global resource constraints force the agents to coordinate their decisions, which means that the policies used by the agents should maximize the total expected value while staying below global resource limits. The agents have access to k resource types. For each agent i the consumption of resource type j is defined using a function $C_{i,j} : S_i \times A_i \to [0, c_{\max,i,j}]$, where $c_{\max,i,j}$ denotes the maximum instantaneous consumption of resource type j for agent i. For instantaneous constraints the resource limit at time t is defined by $L_{j,t}$, where the usage at time t does not affect the limit at $t' > t$. Budget constraints are defined by a single limit L_j.

A resource violation occurs if the agents collectively use more units of a resource than available. For instantaneous constraints the limit of resource type j is violated at time t if

$$\sum_{i=1}^{n} C_{i,j}(s_{i,t}, a_{i,t}) > L_{j,t}, \quad (2)$$

where $s_{i,t} \in S_i$ represents the state of agent i at time t and $a_{i,t} \in A_i$ is the action executed by agent i at time t. Budget constraints are violated if

$$\sum_{t=1}^{h} \sum_{i=1}^{n} C_{i,j}(s_{i,t}, a_{i,t}) > L_j. \quad (3)$$

In the remainder of the paper we present our algorithms for instantaneous constraints only. The application to budget constraints is almost identical since one budget constraint involves all time steps rather than one.

Algorithms for Computing Policies

We discuss two methods which can be used to compute policies satisfying the resource constraints in expectation. In the next section we use these methods as a starting point when bounding constraint violation probabilities.

Constrained Markov Decision Processes

The Constrained MDP (CMDP, Altman 1999) framework defines a linear program to solve MDPs and can be used to impose additional constraints on the resulting stochastic policies. The main idea is to introduce a variable $x^i_{t,s,a}$ representing the probability that agent i reaches state s at time t and executes action a. These variables can be used in probability flow conservation constraints based on the stochastic MDP transition function. The flow is initialized using a known initial state probability distribution.

The CMDP formulation corresponds to the linear program below, which maximizes the sum of expected rewards of the agents while ensuring that the expected resource consumption stays below the limit for each resource type. The solution of the linear program can be used to define a stochastic policy π_i for each agent i. The stochastic policy selects action a in state s at time t with probability $P(\pi_i(t, s) = a) = x^i_{t,s,a} \,/\, \sum_{a' \in A_i} x^i_{t,s,a'}$.

$$\max \sum_{i=1}^{n} \sum_{t=1}^{h} \sum_{s \in S_i} \sum_{a \in A_i} x^i_{t,s,a} \cdot R_i(s, a)$$

$$\text{s.t.} \sum_{a' \in A_i} x^i_{t+1,s',a'} = \sum_{s \in S_i} \sum_{a \in A_i} x^i_{t,s,a} \cdot T_i(s, a, s') \quad \forall i, t, s' \in S_i$$

$$\sum_{a \in A_i} x^i_{1,s,a} = P(s^{i,1} = s) \quad \forall i, s \in S_i$$

$$\sum_{i=1}^{n} \sum_{s \in S_i} \sum_{a \in A_i} x^i_{t,s,a} \cdot C_{i,j}(s, a) \leq L_{j,t} \quad \forall j, t$$

Column Generation for Linear Programming

An alternative linear programming formulation has been proposed by Yost and Washburn (2000). In this formulation we use Z_i to denote the finite set containing all possible deterministic policies of MDP M_i. However, we never actually

enumerate all policies in this set. In contrast to CMDPs, the solution defines a probability distribution over deterministic policies, which requires less randomization during execution. For a policy $\pi_i \in Z_i$ we define V_{i,π_i} as the expected reward of agent i when using policy π_i starting from the initial state $s^{i,1}$. For agent i we define $C^{j,t}_{i,\pi_i}$ as the expected consumption for resource type j at time t when using policy $\pi_i \in Z_i$ from $s^{i,1}$. This expectation is equal to $H_{i,j}(s^{i,1}, 1)$, defined by the following recurrence:

$$H_{i,j}(s,t') = \begin{cases} C_{i,j}\big(s, \pi_i(t',s)\big) & t' = t \\ \sum_{s' \in S_i} P\left(s' | s, \pi(t',s)\right) H_{i,j}(s', t'+1) & t' < t \end{cases}$$

Rather than selecting a policy $\pi_i \in Z_i$ for each agent i, Yost and Washburn propose to search for a probability distribution over (deterministic) policies $\pi_i \in Z_i$ as follows:

$$\begin{aligned} \phi = \max & \sum_{i=1}^{n} \sum_{\pi_i \in Z_i} V_{i,\pi_i} \cdot x_{i,\pi_i} \\ \text{s.t.} & \sum_{i=1}^{n} \sum_{\pi_i \in Z_i} C^{j,t}_{i,\pi_i} \cdot x_{i,\pi_i} \leq L_{j,t} \qquad \forall j,t \\ & \sum_{\pi_i \in Z_i} x_{i,\pi_i} = 1 \quad \forall i, \quad x_{i,\pi_i} \geq 0 \quad \forall i, \pi_i, \end{aligned} \tag{4}$$

in which the variables x_{i,π_i} define this probability distribution. The LP contains a column for each policy $\pi_i \in Z_i$ for each agent i. The number of deterministic policies of an agent scales exponentially in the number of states, which makes directly solving the LP intractable.

In order to prevent full policy enumeration, a Column Generation algorithm can be used which incrementally adds columns corresponding to policies. It keeps track of a lower bound ϕ_l and an upper bound ϕ_u on the optimal objective value of (4) and adds columns until convergence. A lower bound ϕ_l on the objective value can be obtained by solving the LP with only a subset of policies. An upper bound ϕ_u can be obtained using a Lagrangian relaxation:

$$\begin{aligned} \max & \sum_{i=1}^{n} \sum_{\pi_i \in Z_i} V_{i,\pi_i} \cdot x_{i,\pi_i} \\ & + \sum_{j,t} \lambda_{j,t} \left(L_{j,t} - \sum_{i=1}^{n} \sum_{\pi_i \in Z_i} C^{j,t}_{i,\pi_i} \cdot x_{i,\pi_i} \right) \\ \text{s.t.} & \sum_{\pi_i \in Z_i} x_{i,\pi_i} = 1 \quad \forall i, \qquad x_{i,\pi_i} \geq 0 \quad \forall i, \pi_i, \end{aligned} \tag{5}$$

where $\lambda_{j,t}$ is the Lagrangian multiplier corresponding to a constraint in (4), which can be obtained from the dual solution. Since the constraints in the relaxation only affect individual MDPs the upper bound is equal to:

$$\sum_{j,t} \lambda_{j,t} L_{j,t} + \sum_{i=1}^{n} \left(\max_{\pi_i \in Z_i} \left[V_{i,\pi_i} - \sum_{j,t} \lambda_{j,t} C^{j,t}_{i,\pi_i} \right] \right). \tag{6}$$

The maximization problem in (5) decouples into n separate subproblems for which a maximizing policy should be found. Such a maximizing policy can be found by solving the MDP using a time-dependent reward function $G_{i,t}(s,a) = R_i(s,a) - \sum_j \lambda_{j,t} C_{i,j}(s,a)$, which can be solved using standard MDP algorithms (e.g., value iteration).

The Column Generation algorithm initializes an empty master LP. In each iteration it solves the LP to obtain the multipliers $\lambda_{j,t}$ and a lower bound ϕ_l. The multipliers are used to solve n separate MDPs M_i using the reward function $G_{i,t}$, also resulting in a new upper bound ϕ_u. Each computed policy then becomes a new column based on its expected reward and expected resource consumption. The algorithm is anytime, guaranteed to converge and subproblems can be solved in a distributed fashion. Column Generation terminates when the dual prices $\lambda_{i,t}$ stabilize, leading to an equal lower and upper bound (Vanderbeck 2005; Liang and Wilhelm 2010). More details and pseudocode are provided in the supplement, which is available on the homepage of the authors.

Bounding Constraint Violation Probabilities

The methods discussed in the previous section compute policies which ensure that the *expected* resource consumption does not violate the limits. However, the methods do not provide any guarantees regarding the probability that a resource limit is violated during execution. In this section we introduce a method which ensures that the probability of violating any individual constraint is upper bounded by a given parameter α. First we use Hoeffding's inequality to derive tighter resource limits to bound violation probabilities. As this bound is unnecessarily conservative, we then describe a technique to iteratively relax it, resulting in policies with higher expected value and constraint violation probabilities closer to the given tolerance α.

Computing Resource Limit Reductions

We consider arbitrary constraints $\sum_{i=1}^{n} E_{i,j,t} \leq L_{j,t}$, where $E_{i,j,t}$ denotes the expected resource consumption of agent i at time t for resource type j. In the CMDP formulation this is the constraint $\sum_{i=1}^{n} \sum_{s \in S_i} \sum_{a \in A_i} x^i_{t,s,a} \cdot C_{i,j}(s,a) \leq L_{j,t}$, and in the master LP for column generation in Equation 4 this is the constraint $\sum_{i=1}^{n} \sum_{\pi_i \in Z_i} C^{j,t}_{i,\pi_i} \cdot x_{i,\pi_i} \leq L_{j,t}$. Our technique can be applied to any such constraint and thus to both CMDPs and Column Generation.

We introduce a reduced resource limit $0 \leq L^*_{j,t} \leq L_{j,t}$ such that the LP constraint becomes $\sum_{i=1}^{n} E_{i,j,t} \leq L^*_{j,t}$. Next, we define this reduced resource limit $L^*_{j,t}$ in such a way that the original limit violation probability is bounded by α. The proof relies on the fact that we can define the actual consumption of MDP M_i during execution as a random variable $0 \leq X_{j,t,i} \leq c_{\max,i,j}$. The total consumption for resource type j at time t then is their sum:

$$X_{j,t} = X_{j,t,1} + X_{j,t,2} + \ldots + X_{j,t,n}, \tag{7}$$

with an expected value of $E[X_{j,t}] = \sum_{i=1}^{n} E_{i,j,t} \leq L^*_{j,t}$. Since these random variables follow from executing policies after these have been computed (either by the CMDP or the Column Generation method), and each of the variables relate to different agents executing their policies independently,

these random variables are independent. We can thus use Hoefding's inequality to bound the probability that the sum of the random variables exceeds $L_{j,t}$ (Hoeffding 1963).

Theorem 1. *Given a resource type j and a timestep t for which the reduced limit is defined by*

$$L^*_{j,t} = L_{j,t} - \sqrt{\frac{\ln(\alpha) \cdot (\sum_{i=1}^{n} c_{\max,i,j})}{-2}}, \tag{8}$$

it holds that $P\left(X_{j,t} > L_{j,t} \mid E[X_{j,t}] \leq L^*_{j,t}\right) \leq \alpha$.

Proof. Without loss of generality we assume that $E[X_{j,t}] = L^*_{j,t} - \theta = L_{j,t} - \hat{L}_{j,t} - \theta$ for $\theta \geq 0$. Hoeffding's inequality provides a bound on the probability that the sum of n independent random variables deviates from its expectation (Hoeffding 1963). We use the relation $L_{j,t} = E[X_{j,t}] + \hat{L}_{j,t} + \theta$ and Hoeffding's inequality to derive the following:

$$\begin{aligned}
&P\left(X_{j,t} > L_{j,t} \;\middle|\; E[X_{j,t}] = L_{j,t} - \hat{L}_{j,t} - \theta\right) \\
&= P\left(X_{j,t} > E[X_{j,t}] + \hat{L}_{j,t} + \theta\right) \\
&= P\left(X_{j,t} - E[X_{j,t}] > \hat{L}_{j,t} + \theta\right) \\
&\leq \exp\left(\frac{-2 \cdot \left(\hat{L}_{j,t} + \theta\right)^2}{\sum_{i=1}^{n} c_{\max,i,j}}\right) \leq \exp\left(\frac{-2 \cdot \left(\hat{L}_{j,t}\right)^2}{\sum_{i=1}^{n} c_{\max,i,j}}\right).
\end{aligned} \tag{9}$$

To ensure that the probability is upper bounded by α, we need to find a reduction $\hat{L}_{j,t}$ for which it holds that $\exp\left(-2 \cdot (\hat{L}_{j,t})^2 \,/\, \sum_{i=1}^{n} c_{\max,i,j}\right) = \alpha$. Rewriting this equality yields the term $\hat{L}_{j,t}$ that is subtracted in Equation 8. □

Dynamic Constraint Relaxation

Hoeffding's inequality bounds the constraint violation probability from above, and therefore the resulting resource constraint may be too conservative. This is due to the fact that the inequality defines a general bound on the probability that the sum of n independent random variables deviates from its expectation, regardless of their distribution. In practice this means that the bound can be relatively loose. We propose a dynamic constraint relaxation technique which adjusts the reduced resource limit $L^*_{j,t}$ on the basis of empirical evidence of actual violations during simulation. Our technique adapts the resource limits until the LP solution corresponds perfectly with the desired violation probability.

We start with reduced resource limits $L^*_{j,t}$ obtained using Hoeffding's inequality, as shown in Equation 8, and compute a policy for each agent, which can be done using either CMDPs or Column Generation. After obtaining the policies our algorithm runs m Monte Carlo trials to obtain an estimate of the probability distribution of the actual resource consumption. Figure 1 (left) illustrates such a probability distribution, which also illustrates that the resource limit reduction obtained using Hoeffding's inequality can be very conservative. After estimating the distribution we determine

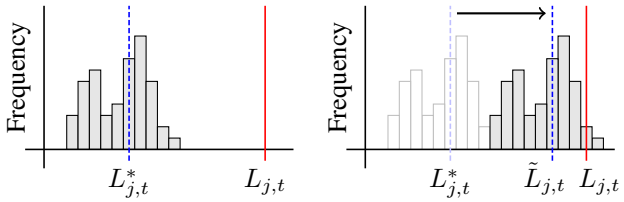

Figure 1: Relaxing the resource limit $L^*_{j,t}$ on the basis of an empirical estimate of the distribution.

the limits $L^*_{j,t} \leq \tilde{L}_{j,t} \leq L_{j,t}$ for which αm violations of the true limit $L_{j,t}$ occur, as illustrated in Figure 1 (right). In practice the distribution may change significantly by increasing the limit. Hence, we propose to iteratively relax the resource constraints as follows:

$$\begin{aligned}
L^0_{j,t} &= L^*_{j,t}, \\
L^{\gamma+1}_{j,t} &= \left(L^{\gamma}_{j,t} + \tilde{L}_{j,t}\right)/\beta.
\end{aligned} \tag{10}$$

where $\tilde{L}_{j,t}$ is determined based on Monte Carlo trials. The auxiliary variable γ represents the iteration index and is used to keep track of the previously chosen resource limit. The parameter β controls the speed of the convergence.

The algorithm starts iteration $\gamma = 0$ with the resource limits $L^0_{j,t}$ obtained using Equation 8. Based on these limits it computes a policy for each agent, after which it executes the Monte Carlo trials to obtain the limits $\tilde{L}_{j,t}$. Finally, the relaxed resource limits $L^{\gamma+1}_{j,t}$ are computed. When starting iteration $\gamma + 1$, the algorithm computes policies based on the newly obtained resource limits, after which the entire procedure starts again. This process is repeated until the policies meet the desired violation tolerance α.

Approaching the ideal resource limit from below instead of from above is preferable because in the worst case the resource limit equals the limit obtained using Hoeffding's inequality. Additionally, by relaxing constraints the LP solution obtained in the previous iteration remains feasible, allowing for efficient warm restarts. This is an anytime algorithm because we never tighten constraints, and thus every iteration can only improve the expected value of the policies.

Accelerating the Column Generation Method

The Column Generation algorithm executes several iterations when computing policies, and in each iteration it adds n columns to the master LP, which correspond to the policies of the agents. Unfortunately, the total number of columns in the master LP grows very large during planning. As a consequence, the time required to solve the master LP increases during the execution of our algorithm. In the remainder of this section we discuss a novel technique to *prune columns* from the master LP to accelerate the algorithm. In contrast to literature that mentions removing columns (e.g., Barnhart et al. 1998), we also present necessary conditions and prove that convergence of Column Generation is unaffected.

An LP solver makes the distinction between basic variables and non-basic variables (Papadimitriou and Steiglitz 1982). The basis of an LP corresponds to the set of basic

variables, having a value assigned that is non-zero. The non-basic variables are not part of the basis and their value is zero. Given an LP with g rows, the number of basic variables is at most g. This means that any LP solution found during Column Generation has many variables with zero probability assigned, most of which will never enter the basis again.

We propose a subroutine to remove columns from the LP. Column Generation requires an LP that has a feasible solution at all times, and therefore we only remove columns if it does not affect the feasibility of the LP. The LP is feasible if there exists an assignment of probabilities to variables which satisfies the constraints in the master LP. Non-basic variables do not contribute to the objective value and can be removed without changing the current solution of the master LP.

This pruning operation removes columns, which may affect the convergence characteristics. However, we can preserve convergence characteristics if pruning is only executed under specific circumstances. We define $\phi_{l,\rho}$ as the lower bound found in iteration ρ of Column Generation. If the pruning operation is only executed after a strict increase of the lower bound (i.e., $\phi_{l,\rho} > \phi_{l,\rho-1}$), then the algorithm still converges, as shown below.

Theorem 2. *Consider the lower bounds $\phi_{l,\rho-1}$ and $\phi_{l,\rho}$ found in two successive iterations $\rho - 1$ and ρ. Pruning does not affect the feasibility of the master LP, and Column Generation converges to an optimal solution if the pruning subroutine is only executed in case $\phi_{l,\rho} > \phi_{l,\rho-1}$.*

Proof. Pruning only removes columns corresponding to non-basic variables, and removing such columns never affects feasibility of a linear program. Now we will show that Column Generation combined with pruning still converges. We let ϕ denote the optimal objective that needs to be found. The number of distinct lower bounds ϕ_l is finite since each MDP has a finite number of policies. Consider two iterations $\rho - 1$ and ρ for which $\phi_{l,\rho} > \phi_{l,\rho-1}$. The algorithm executes pruning and may remove columns from the linear program, after which it continues generating columns. Consider a lower bound $\phi_{l,\rho} < \phi_{l,q} \leq \phi$ for which $q > \rho$. The algorithm is guaranteed to reach $\phi_{l,q}$ since it converges towards ϕ and it only executes pruning when actually reaching $\phi_{l,q}$. Since the number of lower bounds is finite, the algorithm is also guaranteed to reach a lower bound that is equal to ϕ. □

Nevertheless, pruning *all* unselected columns can cause recently used columns to be recomputed. Therefore, we retain columns which have entered the basis in the past δ iterations.[1]

Experiments

To test our proposed algorithms, we compare with two state-of-the-art off-line resource preallocation strategies: an optimal preallocation Mixed-Integer Linear Program (MILP, Wu and Durfee 2010), and its Lagrangian Relaxation using a greedy strategy to obtain feasible allocations, called LDD+GAPS (Agrawal, Varakantham, and Yeoh 2016). We first evaluate performance on three problem domains to study the robustness of the algorithms. Subsequently, we show the performance benefit of column generation and pruning over direct encoding as CMDP. We end with a scalability experiment on a large-scale advertising domain.

In our graphs 'CMDP' refers to solving the problem using CMDPs without reduced limits. 'Hoeffding (CMDP)' combines CMDPs with reduced limits obtained through Hoeffding's inequality. 'Dynamic' combines constraint relaxation with either CMDPs or Column Generation with Pruning (using cut-off $\delta = 50$, CG). All LPs were solved using Gurobi 7.0 on a 2.1Ghz quad-core i7. Parameter β was determined to fit the domain, ranging from 3 (advertising) to 100 (lottery). As a rule, a problem with fewer agents requires a higher β to avoid undershooting the tolerance α.

Hard Artificial Domain The *Lottery* problem[1] has only 1 resource (the prize), which can be used by an agent in the winning state to redeem a large reward. Which agent reaches the winning state is determined by nature, however in expectation only 1 agent will reach the winning state. Off-line coordination on this type of problem is very difficult because the resource demand depends completely on stochastic transitions. Preallocation strategies can only nominate 1 agent as the potential winner, which means that it can perform arbitrarily worse than on-line coordination.

The leftmost column of Figure 2 presents the results of all algorithms on *Lottery*. Expected values are normalized to the CMDP policy, which returns the optimal expected value. Frequency of violations is observed through 500,000 Monte Carlo trials. We observe that the lottery problem is computationally easy to solve even for over 500 agents. As expected, the obtained value decreases as $\frac{1}{n}$ for both preallocation strategies (MILP and LDD+GAPS) in an equal manner. We also observe that the Hoeffding bound is very conservative on this problem, resulting in a low value and significantly less violations than the tolerance α. Dynamic constraint relaxation on the other hand is able to obtain constant value with a stable bounded constraint violation probability, both only surpassed by the unbounded CMDP, illustrating the expected trade-off between expected value and number of violations.

Two Benchmark Domains In the Thermostatically Controlled Loads (*TCL*) problem (De Nijs, Spaan, and De Weerdt 2015), the planner is tasked with finding an activation schedule for heating devices that maximizes comfort subject to a power constraint. In the Mars rover *Maze* (Wu and Durfee 2010) the planner must assign a limited set of tools to exploration rovers to maximize value of research tasks performed. To explore the scalability of the algorithms, we increase the number of agents while keeping the other dimensions constant. Each TCL agent has 24 states and 2 actions, horizon 24 and 1 resource type (24 resources in total). Our Maze problems have 26 states and 10 actions per agent, horizon 15, and 3 resource types (resulting in 45 resource constraints).

The middle and right columns of Figure 2 compare the algorithms on these respective problems. We observe that the TCL problem is computationally hard for the preallocation strategies. Both MILP and LDD+GAPS exceed the 1 hour timeout for 4 agents. The Hoeffding bound is less conservative here, but it still comes an order of magnitude short of the

[1] See authors' homepages for supplementary material on integrating pruning in Column Generation and on the Lottery domain.

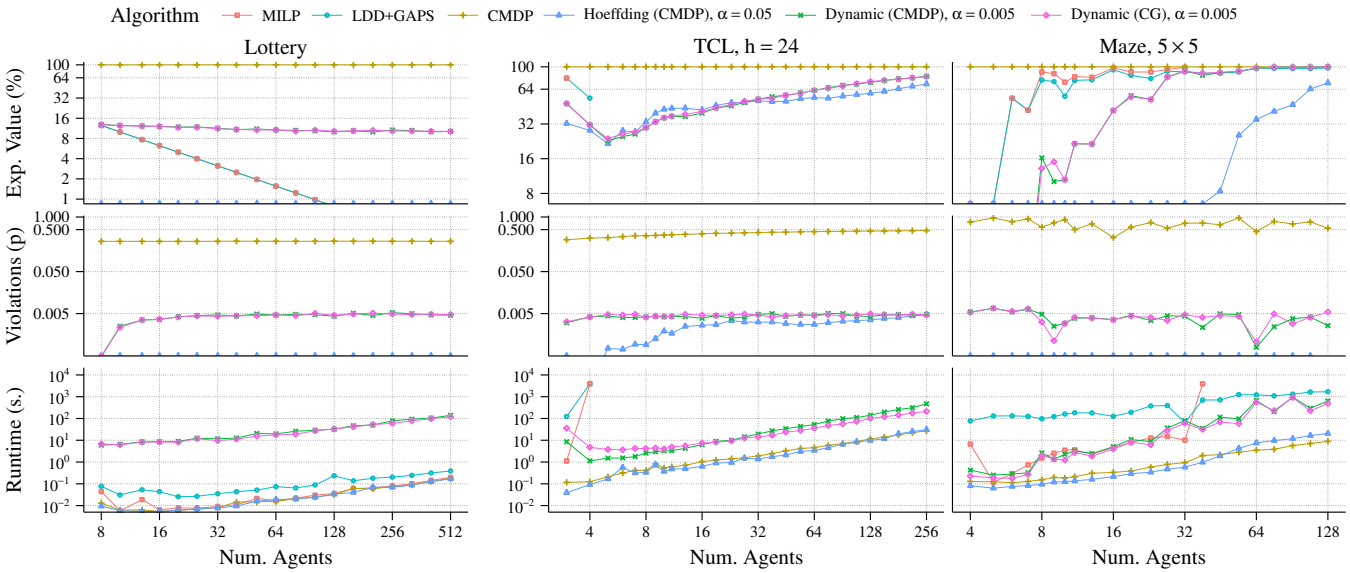

Figure 2: Comparison of algorithm performance on three problem domains. Policy value normalized to CMDP solution. All graphs set on log-log scales.

target tolerance. Dynamic constraint relaxation approaches the (stricter) tolerance and additionally obtains a higher value. We observe that for large numbers of agents, the value of the bounded approaches tends towards the optimal value. When more agents are available to spread the load of the reduced resource limit, their individual rewards are compromised less.

In the TCL domain agents are penalized for consuming too many resources, because their temperature would exceed their setpoint. In contrast, for Maze domain agents using more resources is strictly better than using less. This translates into a higher violation probability for the CMDP approach, and an easier problem in general as observed by the better scalability of the preallocation approaches. Nevertheless, for a sufficient number of agents our dynamic constraint relaxation obtains an expected value not significantly lower than the CMDP approach with two orders of magnitude fewer violations.

Column Pruning Performance To demonstrate the impact of pruning on the performance of Column Generation, and to compare this approach with CMDPs, we present results on larger TCL instances (100 agents, 80 states, up to 128 hours instead of 24) in Figure 3. Here the problem is scaled by increasing the planning horizon. Increasing the number of time steps corresponds to increasing the number of resource constraints in the model, which results in larger columns in the linear program. We observe that as the horizon increases, TCL problems quickly become intractable unless pruning is used. In particular, CMDPs appear to suffer dramatically.

Large-scale Advertising Domain Using column pruning, we are able to tackle large-scale planning problems, such as the synthetic advertising domain presented by Boutilier and Lu (2016). Their domain consists of assigning advertisement budget to potential customers to maximize the amount of sales, where each individual customer is modeled as an MDP. In this domain the resource constraint is not time-dependent,

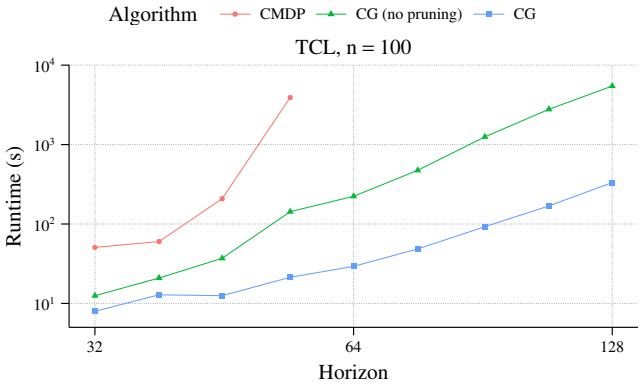

Figure 3: Comparison of CMDPs with Column Generation, including runtime benefit as a consequence of pruning.

but applies to all time steps. Because of the scale of the model (1000 agents, each with 15 states and 5 actions), direct application of the CMDP algorithm is intractable. Additionally, the preallocation strategies can not be applied to this problem because of non-binary resource consumption. However, as Figure 4 shows, using Column Generation with pruning it is possible to solve this problem in less than a second (both with and without Hoeffding). We see that the version without constraint reductions violates the selected budget level half of the time. The Hoeffding bound, however, is again very conservative, because of the large potential maximum consumption relative to the expectation. This is addressed by the dynamic constraint relaxation method, which reduces violations significantly compared to the version without constraint reductions. We also observe that there is only a very small reduction in expected value compared to the solution without a bound on the violation probability.

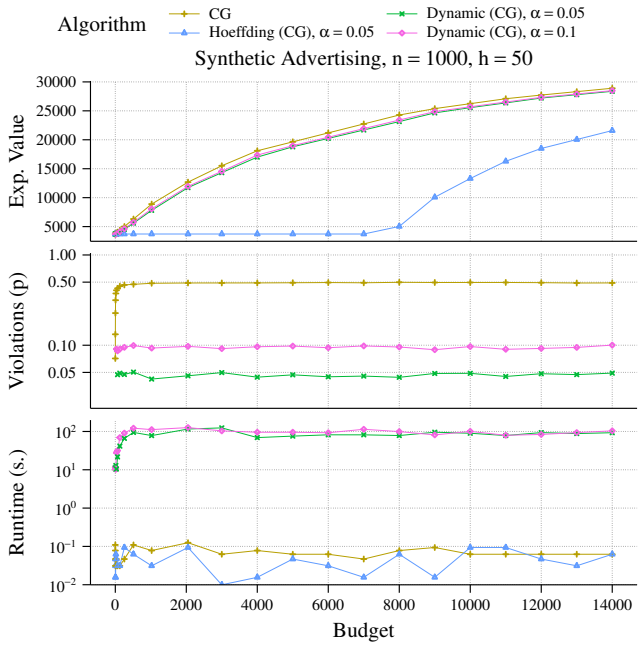

Figure 4: Scalability of Column Generation with dynamic constraint relaxation on a large-scale advertising domain.

Conclusions

We study multiple agents in stochastic domains that need to coordinate their actions off-line due to limited availability of resources and lack of communication. CMDPs and Column Generation algorithms compute policies which satisfy resource constraints in expectation, but these policies provide no guarantees on the probability that constraint violations occur. We therefore propose a new method to bound constraint violation probabilities. Our method is based on Hoeffding's inequality and uses a dynamic constraint relaxation technique to ensure that constraint violation probabilities are tightly bounded by a given tolerance. Policies computed by our method ensure bounded constraint violation probabilities, even if these policies are executed independently without communicating. Since Column Generation has several attractive properties when combining it with our method, we introduced a column pruning technique to accelerate the algorithm. Experiments on hard instances and more realistic problems have shown that our method outperforms two existing state-of-the-art methods for computing deterministic resource allocations.

Studying how constraints on violation probabilities can be encoded directly in a linear program is an interesting future work, because it yields non-convex problem formulations. Secondly, column generation techniques have also been used to solve large security games (Jain et al. 2010), and it remains to be studied if column pruning also further accelerates these and other algorithms based on column generation.

Acknowledgments

This research is funded by distribution system operator Alliander, and by the Netherlands Organisation for Scientific Research (NWO), as part of the Uncertainty Reduction in Smart Energy Systems program. We would like to thank Pritee Agrawal for the information about LDD+GAPS, and Craig Boutilier for the description of the advertising domain.

References

Agrawal, P.; Varakantham, P.; and 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*, 10–16.

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

Barnhart, C.; Johnson, E. L.; Nemhauser, G. L.; Savelsbergh, M. W.; and Vance, P. H. 1998. Branch-and-price: Column generation for solving huge integer programs. *Operations research* 46(3):316–329.

Bellman, R. 1957. A Markovian Decision Process. *Journal of Mathematics and Mechanics* 6(5):679–684.

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

De Nijs, F.; Spaan, M. T. J.; and 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*, 615–621.

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

Jain, M.; Kardes, E.; Kiekintveld, C.; Ordónez, F.; and Tambe, M. 2010. Security Games with Arbitrary Schedules: A Branch and Price Approach. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, 792–797.

Liang, D., and Wilhelm, W. E. 2010. A generalization of column generation to accelerate convergence. *Mathematical programming* 122(2):349–378.

Meuleau, N.; Hauskrecht, M.; Kim, K.; Peshkin, L.; Kaelbling, L. P.; Dean, T.; and Boutilier, C. 1998. Solving Very Large Weakly Coupled Markov Decision Processes. In *Proceedings of the 15th National Conference on Artificial Intelligence*, 165–172.

Papadimitriou, C. H., and Steiglitz, K. 1982. *Combinatorial Optimization: Algorithms and Complexity*. Courier Corporation.

Vanderbeck, F. 2005. Implementing mixed integer column generation. In Desaulniers, G.; Desrosiers, J.; and Solomon, M. M., eds., *Column Generation*. Boston, MA: Springer US. 331–358.

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

Bounding the Probability of Resource Constraint Violations in Multi-Agent MDPs Supplementary Material

Frits de Nijs and **Erwin Walraven** and **Mathijs M. de Weerdt** and **Matthijs T. J. Spaan**
Delft University of Technology
Mekelweg 4, 2628 CD
Delft, The Netherlands

This document contains a full description and pseudocode of the Column Generation algorithm. We also provide more details about Lottery domain used in the experiments.

Column Generation Pseudocode

Algorithm 1 shows how columns can be incrementally added to the master LP. The algorithm is an adaptation of the method introduced by Yost and Washburn (2000). Since columns correspond to policies, we will use both terms interchangeably. For each MDP M_i the columns are stored in the set Z_i, and the algorithm adds initial policies on lines 3–7, which does not consume any resources. The latter is necessary to ensure that the linear program has a feasible solution satisfying the constraints. The master LP is solved on line 10 to obtain dual prices $\lambda_{j,t}$, after which new columns are generated for each MDP on lines 14–23. The function call on line 17 is a call to our pruning subroutine, shown in Algorithm 2. The algorithm uses the alternative reward function $G_{i,t}$ on line 19. However, the expected reward computed on line 20 does not involve cost components that were subtracted in $G_{i,t}$. The algorithm keeps track of a lower bound ϕ_l and upper bound ϕ_u on the optimal objective value ϕ to detect convergence. Eventually the algorithm returns a probability distribution over policies for each MDP M_i, which is represented by tuples in a set Y_i.

input : Z_i, $\{W_{i,1}, \ldots, W_{i,\rho}\}$, δ
output : column set Z_i'

1 $Z_i' \leftarrow W_{i,\rho} \cup W_{i,\rho-1} \cup \ldots \cup W_{i,\rho-(\delta-1)}$
2 $E \leftarrow Z_i \setminus Z_i'$
3 **foreach** $\pi \in E$ **do**
4 | remove column from master LP: $Z_i \leftarrow Z_i \setminus \{\pi\}$
5 **end**
6 **return** Z_i'

Algorithm 2: Column pruning function `prune`.

input : MDP M_i for each agent i, prune parameter δ
output : policies for each M_i

1 $\phi_l \leftarrow -\infty,\ \phi_u \leftarrow \infty,\ \rho \leftarrow 0,\ W_{i,\rho} \leftarrow \emptyset\ \forall i$
2 initialize empty master LP: $Z_i \leftarrow \emptyset\ \forall i$
3 **foreach** $i = 1, \ldots, n$ **do**
4 | $\pi_i \leftarrow$ policy for M_i consuming no resources
5 | compute V_{i,π_i} and $C^{j,t}_{i,\pi_i}$ using $\pi_i\ (\forall j, t)$
6 | add column: $Z_i \leftarrow Z_i \cup \{\pi_i\}$
7 **end**
8 **do**
9 | $\rho \leftarrow \rho + 1$
10 | solve the master LP to obtain $\lambda_{j,t}\ (\forall j, t)$
11 | $\phi_l \leftarrow$ objective value of the master LP
12 | $\phi_{l,\rho} \leftarrow \phi_l$
13 | $\phi_u \leftarrow \sum_{j,t} \lambda_{j,t} L_{j,t}$
14 | **foreach** $i = 1, \ldots, n$ **do**
15 | | $W_{i,\rho} \leftarrow \{\pi_i \mid \pi_i \in Z_i \text{ and } x_{i,\pi_i} > 0\}$
16 | | **if** $\rho > 1\ \wedge\ \phi_{l,\rho} > \phi_{l,\rho-1}$ **then**
17 | | | $Z_i \leftarrow$ `prune`$(Z_i, \{W_{i,1}, \ldots, W_{i,\rho}\}, \delta)$
18 | | **end**
19 | | solve M_i using $G_{i,t}$ to obtain π_i
20 | | compute V_{i,π_i} and $C^{j,t}_{i,\pi_i}$ using $\pi_i\ (\forall j, t)$
21 | | add column: $Z_i \leftarrow Z_i \cup \{\pi_i\}$
22 | | $\phi_u \leftarrow \phi_u + (V_{i,\pi_i} - \sum_{j,t} \lambda_{j,t} C^{j,t}_{i,\pi_i})$
23 | **end**
24 **while** $\phi_u - \phi_l > 0$;
25 $Y_i \leftarrow \{(\pi_i, x_{i,\pi_i}) \mid \pi_i \in Z_i \text{ and } x_{i,\pi_i} > 0\}\ \ \forall i$
26 **return** $\{Y_1, \ldots, Y_n\}$

Algorithm 1: Column Generation method

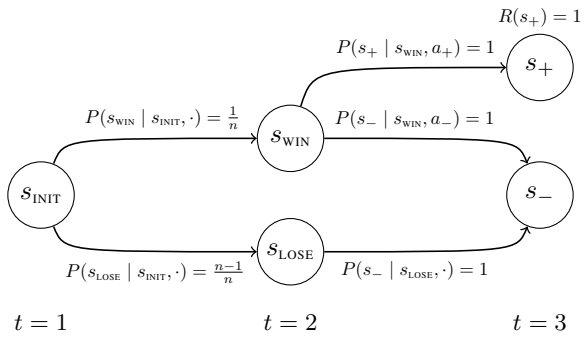

Figure 1: Graphical description of the *Lottery* agents' MDP model, presenting the transition and reward functions. Unspecified transitions and rewards have value 0.

Lottery Domain

The *Lottery* problem consists of n identical agents, which are jointly trying to win a lottery. The agents know the odds of the lottery, so that they are able to purchase sufficient tickets to win the jackpot *in expectation*. They divide the load of purchasing the tickets equally amongst themselves, so that each agent has probability $\frac{1}{n}$ to own a winning ticket. The lottery draw is distributed, so that each agent can individually determine whether or not it has a winning ticket. Agents owning winning tickets are expected to coordinate their actions so that only one redeems the indivisible prize.

Formally, the Lottery problem is modeled as an MMDP with Global Resource Constraints as follows:

$$\begin{aligned} S_i &= \langle s_{\text{INIT}}, s_{\text{LOSE}}, s_{\text{WIN}}, s_-, s_+ \rangle \\ A_i &= \langle a_-, a_+ \rangle \quad C_{i,1}(\cdot, a_-) = 0 \\ h &= 3 \qquad\quad C_{i,1}(\cdot, a_+) = 1 \\ s^{i,1} &= s_{\text{INIT}} \qquad\quad L_{1,t} = 1 \end{aligned} \tag{1}$$

Figure 1 presents the transition function and the rewards of the model. The action a_+ is used to claim the prize, which is constrained by the resource limit function to be used by at most 1 agent. Using action a_+ in the state s_{WIN} indicating ownership of a winning ticket results in reaching the only state with positive reward, s_+.

The goal of the agents is to have one of them use the resource to reach s_+, to distribute the reward across all of them. The problem the agents face is that they cannot know a priori which agent(s) will transition to state s_{WIN} since that is up to chance. All they know is that in expectation, one of them will reach s_{WIN}.

References

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