

MIT Open Access Articles

An online algorithm for constrained POMDPs

The MIT Faculty has made this article openly available. **Please share** how this access benefits you. Your story matters.

Citation: Undurti, Aditya, and Jonathan P How. "An Online Algorithm for Constrained POMDPs." 2010 IEEE International Conference on Robotics and Automation. Anchorage, AK, 2010. 3966-3973. © Copyright 2010 IEEE

As Published: <http://dx.doi.org/10.1109/ROBOT.2010.5509743>

Publisher: Institute of Electrical and Electronics Engineers

Persistent URL: <http://hdl.handle.net/1721.1/65564>

Version: Final published version: final published article, as it appeared in a journal, conference proceedings, or other formally published context

Terms of Use: Article is made available in accordance with the publisher's policy and may be subject to US copyright law. Please refer to the publisher's site for terms of use.



An Online Algorithm for Constrained POMDPs

Aditya Undurti and Jonathan P. How

Abstract—This work seeks to address the problem of planning in the presence of uncertainty and constraints. Such problems arise in many situations, including the basis of this work, which involves planning for a team of first responders (both humans and robots) operating in an urban environment. The problem is framed as a Partially-Observable Markov Decision Process (POMDP) with constraints, and it is shown that even in a relatively simple planning problem, modeling constraints as large penalties does not lead to good solutions. The main contribution of the work is a new online algorithm that explicitly ensures constraint feasibility while remaining computationally tractable. Its performance is demonstrated on an example problem and it is demonstrated that our online algorithm generates policies comparable to an offline constrained POMDP algorithm.

I. INTRODUCTION

Partially Observable Markov Decision Processes (POMDPs) provide a broad and generalized framework within which to formulate and solve planning problems in uncertain environments. In these cases the planning system must be capable of deciding when to take actions that improve knowledge of the world and when to act upon information that is already available. POMDPs are particularly well-suited to making such trade-offs in a way that maximizes the performance of the system [10]. However, the standard POMDP formulation does not allow for the incorporation of hard constraints, such as fuel constraints and path constraints.

This work enhances the standard POMDP by adding constraints, and proposes an online algorithm with finite look-ahead for solving the constrained POMDP. This algorithm consists of two parts - an offline approximate solution that predicts whether a constraint-feasible solution exists from each belief state, and an online branch-and-bound algorithm that computes finite-horizon plans and uses the approximate offline solution to guarantee that actions taken at the current time will not make the constraints infeasible at some future time beyond the planning horizon.

The rest of the paper is organized as follows. Section II motivates the interest in constrained POMDPs as a technique for planning for teams of first responders in the aftermath of a Chemical, Biological, Radiological, Nuclear, Explosive (CBRNE) incident. Section III provides a literature review of the work done in POMDPs, online POMDPs and constrained POMDPs. Section IV uses a simple example problem to show that constraints cannot always be treated as large

penalties in a standard unconstrained POMDP. Sections V and VI present the main contribution, the online algorithm combined with an offline approximate solution to generate constraint-feasible plans. Finally Section VII shows that the algorithm presented generates constraint-feasible, high-reward plans in the example problem that are comparable in performance to the offline constrained POMDP algorithms.

II. MOTIVATION

One of the primary advantages of autonomous vehicles is that they can be deployed in situations where there are potential dangers for human agents. One such situation is a Chemical, Biological, Radiological, Nuclear or Explosive (CBRNE) incident in an urban area [1]. Such a situation is challenging because it is dynamic and uncertain [2]. The number of victims that need to be rescued and their locations might not be known, or are known only approximately. Spreading fires and contaminants could impact the numbers of victims and their locations even as the search and rescue mission is underway. Furthermore, the victims might be mobile or secondary devices might go off - both being unanticipated factors that must be accounted for in real-time during the mission. Also, the humans on the team of responders themselves have to stay safe. The team cannot risk sending first responders to areas that are suspected of having contaminants, thus imposing constraints on the mobility of some of the agents. Therefore, responding to a CBRNE incident requires planning and team coordination in a dynamic, uncertain environment in the presence of constraints.

Planning for a heterogeneous team of first responders in a CBRNE search and rescue problem is challenging because the environment is *partially observable*, *stochastic*, *dynamic* and imposes *constraints*. A planning algorithm for the team of first responders must therefore be able to plan in a partially observable environment while also guaranteeing that hard constraints will be satisfied at all times. Furthermore, the tasks and missions to be performed are likely to change as the incident response progresses. Therefore the algorithm must be fast enough to recompute plans. In other words, the planning algorithm must not only plan with partial information and guarantee hard constraints, but also be fast enough to compute plans online. This complex planning problem serves as the motivation for fast online algorithms for solving constrained POMDPs.

III. LITERATURE REVIEW

A solution to a POMDP consists of a *policy* π which is defined as a mapping from the belief space to actions, and

This work was supported by the Office of Naval Research
Aditya Undurti is a PhD Candidate in Aeronautics and Astronautics,
Massachusetts Institute of Technology aundurti@mit.edu
Jonathan How is a professor in the Department of Aeronautics and
Astronautics, Massachusetts Institute of Technology jhow@mit.edu

a *value function* V_π associated with that policy and defined over the belief space. The value function gives the expected reward of executing the policy π from each point in the belief space. However, a major disadvantage of POMDPs is their computational complexity [10]. Several point-based value iteration methods exist for finding approximate solutions, for example Perseus [8] and SARSOP [9]. Other approximate methods include Q-MDP [4] which accounts for uncertainty in the current state but not in future states, and therefore provides an upper bound on the true value function. Braatz et al. have worked on incorporating constraints into POMDP value iteration [3]. The incorporation of constraints increases the computational complexity and involves solving a Mixed Integer Linear Program (MILP) on every iteration. These algorithms are all *offline* algorithms, i.e. they compute a policy based on a model before execution time, and at execution time simply use the previously-computed policy [10]. Thus most computation is done offline.

Several online algorithms for fast computation of approximate policies have been developed, for example Real-Time Belief Space Search (RTBSS) [7] and rollout [6]. Several other algorithms are summarized in a review paper by Ross et al. [5]. Online algorithms compute the approximate expected value of executing an action in the current belief space by searching through a finite depth AND-OR tree with actions and observations as the branches and belief states as the nodes. The estimated value-to-go from each of the leaves at the end of the tree is estimated with the approximate offline solution obtained previously. Online algorithms are well-suited to the problem being considered because they can quickly recompute plans in real time. However, when applying online algorithms to a problem with constraints, care needs to be taken to ensure that actions being executed currently will not lead to constraint infeasibility at some time beyond the planning horizon. This paper presents an online algorithm to solve constrained POMDPs while providing guarantees that the constraints will remain feasible.

IV. PROBLEM STATEMENT

As stated above, this work seeks to address the problem of using online algorithms to solve a constrained POMDP. A constrained POMDP is defined as the tuple $\langle S, A, Z, R, T, O, C \rangle$, where S is the state space, A is the set of actions, Z is the set of observations, $R(s, a) \forall s \in S, a \in A$ is the reward function, $T(s'|s, a) \forall s', s \in S, a \in A$ is the state transition model, and $O(z|s) \forall z \in Z, s \in S$ is the observation model. $C(s) \forall s \in S$ is the *constraint model*, and the value it returns is defined as the *constraint penalty*. The constraint penalty is defined as follows. If state s is disallowed for the system, i.e. violates a constraint, $C(s) = 1$, and 0 otherwise.

With these definitions, we can state that the objective is to compute a policy π^* to maximize the cumulative expected reward while keeping the expected constraint penalty below α , i.e.

$$\pi^* = \arg \max_{\pi} E \left[\sum_{t=0}^T \gamma^t R(s_t, a_t) \right] \quad (1)$$

$$\text{s.t. } E \left[\sum_{t=0}^T C(s_t) \right] \leq \alpha \quad (2)$$

Since the constraint penalty is always greater than 0, $E \left[\sum_{t=0}^T C(s_t) \right] \leq 0$ if and only if $C(s_t) = 0 \forall t$. Thus by setting $\alpha = 0$, we can impose hard constraints. By setting α to a value between 0 and 1, we can specify the probability with which we want constraints to be enforced.

Two distinct approaches to solving the above constrained POMDP exist in the literature. Each of these methods are briefly discussed below.

A. Constraints as Penalties

The first is to convert the constraint penalty into a large negative reward. Thus a new reward function is defined as

$$\hat{R}(s_t, a_t) = R(s_t, a_t) - MC(s_t) \quad (3)$$

where M is a large positive number. This new reward function $\hat{R}(s_t, a_t)$ is used to solve the standard unconstrained POMDP $\langle S, A, Z, \hat{R}, T, O \rangle$. Varying the value of M can make the policy more risky or more conservative. However, a counter example presented below and shown in Figure 1 illustrates that an M value does not always exist for all risk levels.

An agent starts in state s_1 at time 0, and the goal is to reach state s_3 at time 2 with a probability of at least 0.9. The agent can take two actions in states s_1 and s_2 - a *walk* action that reaches the goal state with probability 1, and a *hop* action that reaches the goal state with probability 0.9 and the absorbing state s_4 with probability 0.1. The *hop* action is therefore rewarding but risky, while the *walk* action is safe but unrewarding.

Since s_4 is an absorbing state, satisfying the goal is impossible once the agent falls into state s_4 . Therefore we would like to prevent the agent from entering state s_4 by imposing the constraint penalty M on that state. By inspection, the optimal policy π^* corresponding to a risk level $\alpha = 0.1$ is to *hop* in state s_1 but *walk* in state s_2 . It can be shown that *hop* is chosen as the best action from state s_2 for $M < 900$, but *walk* is chosen as the best action in state s_1 for $M \geq 990$. There is *no* value of M for which the optimal policy is obtained - the planner switches between being too conservative (only *walks*) to too risky (only *hops*). The intent of this simple example is to illustrate that incorporating constraints as negative penalties does not always yield good policies.

B. Constrained POMDP Value Iteration

A second approach is to explicitly account for the constraint and compute a policy offline [3]. This approach maintains two value functions - $V_R(s)$ associated with the reward function, and $V_C(s)$ associated with the constraint penalty. Isom, Meyn and Braatz present a modification to POMDP Value Iteration to obtain a policy for the constrained POMDP. However, this algorithm is computationally demanding - value iteration for unconstrained POMDPs becomes computationally expensive as the size of the problem grows, and

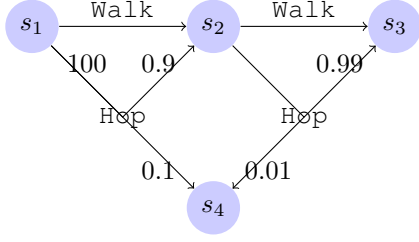


Fig. 1: A counter example to show that a penalty value corresponding to the optimal policy does not always exist

the incorporation of constraints adds to the complexity by requiring that a Mixed-Integer Linear Program (MILP) be solved in every iteration.

The shortcomings of the two approaches to solving constrained POMDPs motivates the development of a fast, online algorithm for solving constrained POMDPs while explicitly accounting for the constraints. The proposed solution technique is presented in the next section.

V. PROPOSED SOLUTION

The proposed solution to the constrained POMDP defined in Equations 1 and 2 is to use an online forward search to optimize the reward and check for constraint feasibility upto a horizon of length D . Constraint feasibility for times beyond D is ensured by using a constraint-penalty-to-go estimate that is computed offline. The solution requires two quantities to be tracked - the reward R and the constraint value C . For this reason, we maintain two separate value functions - one associated with the reward $R(s, a)$ which we call $V_R(s)$ and another associated with the constraint value $C(s, a)$ which we call $V_C(s)$. The algorithm presented has two components - an offline component where an approximate solution for V_C is obtained, and an online component where the offline estimate for V_C is refined and V_R is computed over the finite horizon to select the best action.

A. Offline Constraint Penalty Estimate

The overall goal of the planner is to maximize the expected reward obtained while keeping the constraint penalty below a threshold, in this case 0. In the offline component of the algorithm, we solve for a policy π_c^* that will *minimize* the constraint penalty. If the minimum constraint penalty from a certain state is below the threshold, in this case ≤ 0 , we can guarantee that there exists at least one action in that state (the action associated with the policy π_c^*) that will satisfy the constraint. During the online portion of the algorithm (described in detail the next section) we use this guarantee to ensure constraint feasibility for times beyond the online planning horizon.

Therefore, we first obtain the best constraint penalty possible, $U_C(s_v)$, by solving the following unconstrained optimization problem:

$$U_C(s_v) = \min_{\pi} E \left[\sum_{t=0}^T C(s_t) \right] \quad (4)$$

If, for any state s , $U_C(s) = 0$, then there exists at least one policy (the optimal policy π_c^* that solves problem 4) that guarantees that starting from s , the constraints will never be violated. In solving the optimization problem shown in Equation 4, we use a Point-Based Value Iteration (PBVI) algorithm. Since PBVI provides an upper bound when minimizing [5], we know that if the value for $U_C(s)$ returned by PBVI remains ≤ 0 , we can guarantee that the problem will remain feasible if we start executing the computed policy π_c^* in state s .

B. Online Reward Optimization

During the online part of the algorithm, we compute the reward and the constraint penalty for belief nodes encountered within the planning horizon. The previously-obtained PBVI upper bound on the constraint penalty is then evaluated at the leaves of the tree to ensure constraint feasibility beyond the planning horizon.

Algorithm 1 is the main online planning algorithm. The inputs to this algorithm are the current belief state b_c , the planning horizon length D , and the upper bound U_C on the constraint value function V_C . The current belief state is initialized to the initial belief b_0 (line 2) and the forward search tree contains only the current belief node. The planning algorithm then begins the cycle of planning and executing (lines 4 and 5). First, the function `Expand` is called on the current belief node. This function, shown in Algorithm 2, builds a tree of depth D and computes the best action to execute. It is discussed in more detail in the next paragraph. Once the action is executed and a new observation is received, the current belief is updated [5] (line 8) as

$$\begin{aligned} b_t(s') &= \tau(b_{t-1}, a_{t-1}, z_t)(s') \\ &= \frac{O(z_t|s') \sum_{s \in S} T(s'|s, a) b_{t-1}(s)}{P(z_t|b_{t-1}, a_{t-1})} \end{aligned} \quad (5)$$

$$P(z|b, a) = \sum_{s' \in S} O(z|s') \sum_{s \in S} T(s'|s, a) b(s) \quad (6)$$

Once the belief is updated, the `Expand` routine is again called on the most current belief. This cycle (lines 5-8) is repeated until execution is terminated.

When the `Expand` subroutine is called with a depth of 0, i.e. if the belief node on which the function has been called is a leaf node, the function simply returns the offline constraint penalty approximation U_C (lines 3-4). For nodes that are not leaf nodes, the algorithm generates all possible successor nodes $\tau(b, a, z)$ by looping through all possible actions (line 10) and observations. Any successor node that 1) has a V_R value that is lower than the best V_R found so far in the tree, or 2) that does not satisfy the constraints, is not considered (line 10). For those nodes that do satisfy these criteria, the `Expand` routine is called recursively to get the expected reward value and the upper bound on the constraint penalty for the successor nodes. The reward is propagated from the successor nodes to the current node according to

$$L_T(b, a_i) = R_B(b, a_i) + \gamma \sum_{z \in Z} P(z|b, a_i) L_T(\tau(b, a, z))$$

Algorithm 1 Pseudocode for an online algorithm to solve constrained POMDPs

```

1: Function ConstrainedOnlinePOMDPSolver()
   Static:
    $b_c$  : The current belief state of the agent
    $T$  : An AND-OR tree representing the current search tree
    $D$  : Expansion Depth
    $U_C$  : An upper bound on  $V_C$ 
2:  $b_c \leftarrow b_0$ 
3: Initialize  $T$  to contain only  $b_c$  at the root
4: while ExecutionTerminated() do
5:   Expand( $b_c, D$ )
6:   Execute best action  $a$  for  $b_c$ 
7:   Receive observation  $z$ 
8:    $b_c \leftarrow \tau(b_c, a, z)$ 
9:   Update tree  $T$  so that  $b_c$  is the new root
10: end while

```

The upper bound on the constraint is propagated according to the equation

$$C(b, a_i) = C_B(b, a_i) + \gamma \left[\max_{z \in Z} U_C(\tau(b, a, z)) \right]$$

Notice that the upper bound on the constraint value is propagated differently than the reward. $C_B(b, a_i)$ shown above is the immediate constraint value gathered by performing action a_i in belief b . This is then added to the *maximum* constraint value over all possible observations. The *max* operator over the observations ensures that the highest possible constraint value is returned, thus ensuring that the $C(b, a_i)$ remains an upper bound.

The action that provides the best reward V_R is returned as the best action to execute in the current node (lines 15 and 21). The constraint and reward values associated with taking that action in the current node are also returned (line 14, 17 and 21), since these values have to be propagated to the parent node.

The algorithm presented here is applied to a robot navigation problem with constraints. First, it will be shown that incorporating constraints as large penalties does leads to policies that are either too conservative or too risky. We will then show that the online algorithm proposed arrives at the same solution as the constrained POMDP algorithm presented in [3] but is less computationally expensive.

VI. EXAMPLE PROBLEM

The example problem considered is a robot navigation problem with some constraints. The dynamic model for the agent is shown in Figure 2. When the agent is given a command to move in a certain direction, there is a probability of 0.9 that it will move in the intended direction, and a probability of 0.05 that it will move in one of the two perpen-

	0.05	
	→	0.9
	0.05	

Fig. 2: Vehicle dynamic model

Algorithm 2 The expand routine for solving a constrained POMDP

```

1: Function Expand( $b, d$ )
   Static:
    $L(b), U(b), U_C(b)$ 
2: if  $d = 0$  then
3:    $L_T(b) \leftarrow 0$ 
4:    $C(b) \leftarrow U_C(b)$ 
5: else
6:    $i \leftarrow 1$ 
7:    $L_T(b) \leftarrow -\infty$ 
8:    $C(b) \leftarrow -\infty$ 
9:   while  $i \leq |A|$  and  $L_T(b, a_i) > L_T(b)$  and  $C(b) \leq 0$ 
       do
10:     $L_T(b, a_i) \leftarrow R_B(b, a_i) +$ 
        $\gamma \sum_{z \in Z} P(z|b, a_i) \text{Expand}(\tau(b, a_i, z), d - 1)$ 
11:     $C(b, a_i) \leftarrow C_B(b, a_i) +$ 
        $\gamma \max_{z \in Z} P(z|b, a_i) \text{Expand}(\tau(b, a_i, z), d - 1)$ 
12:    if  $L_T(b, a_i) = \max(L_T(b), L_T(b, a_i))$  then
13:       $C(b) = C(b, a_i)$ 
14:       $a^* \leftarrow a_i$ 
15:    end if
16:     $L_T(b) \leftarrow \max(L_T(b), L_T(b, a_i))$ 
17:     $i \leftarrow i + 1$ 
18:  end while
19: end if
20: return  $a^*, L_T(b), C(b)$ 

```

dicular directions. In the case shown in Figure 2, when the agent is commanded to move right, there is a probability of 0.9 that it will move right, probability of 0.05 that it will move up and a probability of 0.05 that it will move down.

The environment used in the example is shown in Figure 3. The vehicle starts in location (1, 3), shown with S . There are three locations where a reward can be acquired - locations (5, 3) and (4, 4) give a high reward of 100, whereas location (4, 2) gives a lower reward of 50. Furthermore, locations (3, 3), (4, 3) and (4, 5) are constrained locations - the vehicle is not allowed to enter these states with a probability of more than 0.05, i.e. the constraint violation probability $\alpha \leq 0.05$.

It can be easily verified that a path through all the reward locations violates the constraints with $\alpha > 0.05$. Two constraint-feasible paths, by inspection, are shown in Figure 4. Both paths incur a constraint violation probability of 0.05, since there is a probability of veering into location (4, 3) during the transition out of location (4, 2). However, the vehicle cannot proceed any farther, because any action taken in location (5, 3) will lead to a greater probability of constraint violation. We now show that even in this relatively simple problem, modeling the constraints as negative rewards will lead to either very conservative or very risky behavior. We will also show that the proposed online algorithm with an offline constraint-feasible approximate solution achieves high performance in problems such as this example that require operating close to constraint boundaries.

In a standard MDP formulation, constraints may be mod-

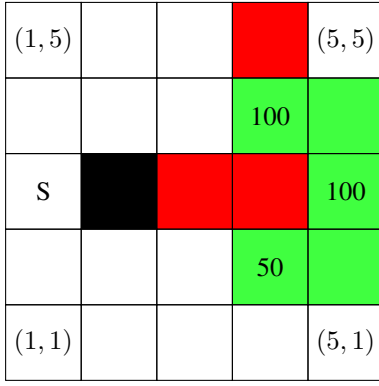


Fig. 3: The MDP problem set up

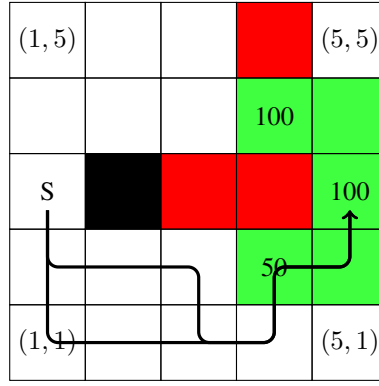


Fig. 4: Constraint-feasible paths

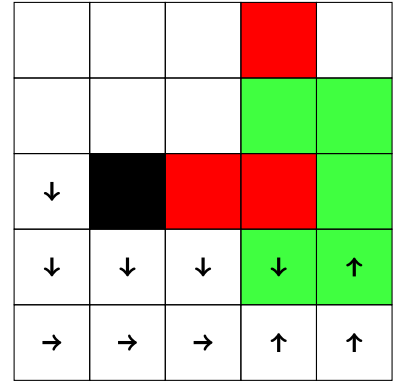


Fig. 5: The policy computed by MDP value iteration when the constraint is modeled as a high negative reward leads to a very conservative policy that fails to achieve high reward

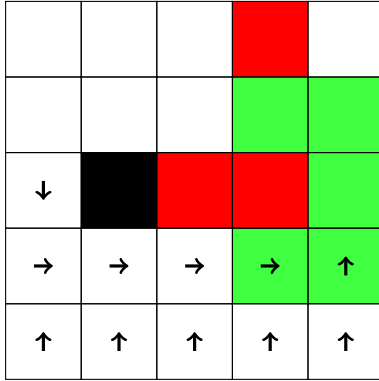


Fig. 6: The policy computed by MDP value iteration when the constraint penalty is lowered

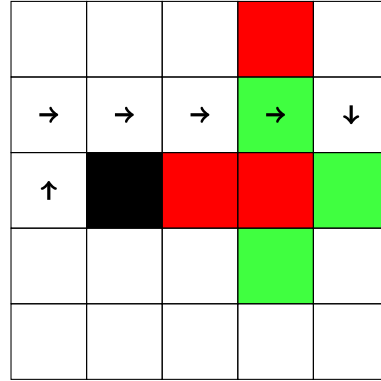


Fig. 7: The policy computed by MDP value iteration when the constraint is modeled as a low negative reward leads to a policy that violates the safety constraint

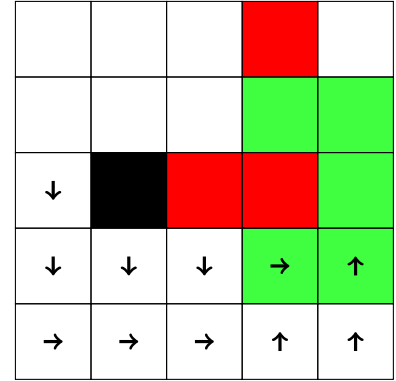


Fig. 8: The policy computed by the online constrained solver. The solver correctly recognizes that going right from location (4,2) is constraint-feasible and yields high reward

eled as large negative rewards. For instance, in this case, we give a reward of -5000 for entering any of the constrained states. The resulting policy is shown in Figure 5. The vehicle moves along the bottom, away from the constrained states, until reaching (4,1). At that point, the vehicle moves up into (4,2) to reach the reward of 50. However, after reaching this state, the vehicle moves back into (4,1). This action is chosen because it is the only action that guarantees that the vehicle will not enter one of the constrained states and acquire the associated penalty. The vehicle decides that the reward of 100 that awaits two steps away is not worth the large negative reward that could be incurred if the vehicle veers into the constrained location (4,3). Due to this conservative behavior, the planner fails to achieve the higher reward in location (5,3). It might seem that lowering the

constraint violation penalty is one potential solution to this conservatism, but we show that lowering the penalty causes the planner to switch from being too conservative to too risky.

Figure 6 shows the outcome of lowering the constraint penalty. Since the constraint violation penalty is lower, the planner decides to follow a path adjacent to the two constrained states with the probability of a constraint violation of 0.095. Lowering the penalty even farther leads to the outcome shown in Figure 7. The planner assumes that the reward at location (4,4) (which is higher than the reward at location (4,2)) is now feasible, and thus switches to the top path which is also constraint-infeasible.

This abrupt switch from conservative to risky is because an MDP planner has only a single reward function that

must capture both performance and constraint feasibility, and therefore lacks the fidelity to accurately specify an acceptable level of risk. This lack of fidelity becomes an important factor in problems where high performance requires operating close to constraint boundaries, as is the case in this example problem. The online constrained MDP algorithm presented in this work provides a general way to constrain risk while optimizing performance.

The online algorithm presented in this work generates the policy shown in Figure 8. The algorithm first generates a conservative offline solution that minimizes the risk. The conservative solution is a policy that minimizes the total constraint penalty. This policy is the same as the conservative policy shown in Figure 5. The online forward search algorithm with a finite search horizon (which for computational reasons was set to 3 in the current problem) uses this approximate solution to identify that location (4, 4) is constraint-infeasible. The forward search sees that a constraint-feasible, high-reward path to (4, 2) exists and therefore begins executing this path. As the search horizon reaches state (5, 3) (for a horizon of 3, this happens when the vehicle is in location (4, 1)), the forward search recognizes that a path to the high-reward location (5, 3) exists, and that path is also constraint-feasible (since the risk of constraint violation is only 0.05). The online planner therefore chooses to move right and claim the reward at (5, 3). Thus the offline approximate solution provides a conservative, constraint-feasible policy while the online algorithm adjusts this policy for better performance.

VII. CBRNE URBAN RESCUE EXAMPLE

In this section, we apply the methods developed in the previous sections to a more complex problem that captures some features of a CBRNE search and rescue scenario. The environment is shown in Figure 9. There are two agents, both of which start in location (5, 8), shown as *A* and *B*. The dynamics of these agents are the same as shown in Figure 2. There are several locations where a reward can be acquired, marked with R_A (reward acquired if the location is visited by Agent *A*) and R_B (reward acquired if the location is visited by Agent *B*) in Figure 9. The locations shown in red indicate danger zones for Agent *A*, i.e. once Agent *A* enters one of these zones, it can execute no further actions. In terms of the motivating CBRNE scenario, Agent *A* can be viewed as a human first responder and Agent *B* an autonomous vehicle. The danger zones represent contaminated areas that are dangerous for a human responder to enter, but can be accessed by a robotic vehicle. The rewards correspond to victims that need to be treated.

Agent *A* is not allowed to enter the danger zone with a probability of more than 0.1, i.e. the constraint violation probability $\alpha \leq 0.1$. However, the rewards acquired in the danger zones are greater when visited by Agent *A* than when visited by Agent *B* ($R_A > R_B$). Also, the rewards R_A and R_B become 0 with a probability of 0.9 at time $T = 10$. In other words, there is a strong incentive for the agents to acquire all rewards before time $T = 10$. The uncertainty in the vehicle dynamics is of the same order

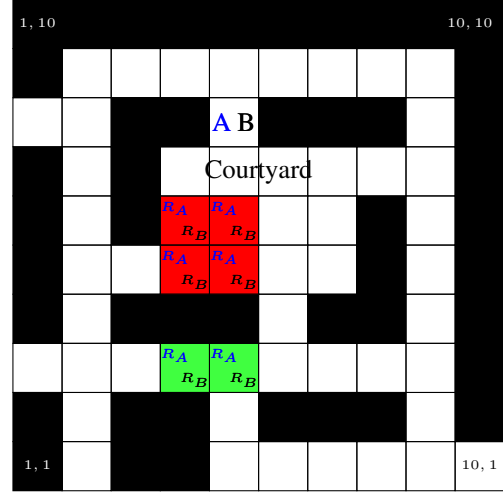


Fig. 9: The MDP problem set up

of magnitude as the constraint violation probability, thus making the interaction between the uncertain dynamics and the constraints an important factor in computing policies. Furthermore, the layout of the problem is such that acquiring high rewards entails taking some risk, as would be expected in a realistic CBRNE situation.

The two highest-reward paths are shown in Figure 10. The highest reward is obtained by having Agent *A* enter the danger zone and acquire a high reward, and using Agent *B* to acquire all other rewards. However, it can be easily seen that this policy violates the constraints with $\alpha > 0.1$. Therefore the highest-reward policy is constraint infeasible. In the next section, we show that an MDP with relatively small negative rewards for constraint violations leads to one of these paths. Two constraint-feasible paths, by inspection, are shown in Figure 11.

The optimal, constraint feasible paths for the agents are for Agent *B* to gather the rewards located inside the danger zone, and for Agent *A* to travel through the inner courtyard area (which involves taking some risk) to reach the rewards on the other side of the courtyard. The path around the courtyard is suboptimal because it is longer. In the next section, we will show that modeling the constraints as negative rewards will lead to either very conservative or very risky behavior, but fails to achieve an optimal balance, i.e. fails to maximize the reward while meeting all constraints. Finally, in the last section, we will show that the proposed online algorithm with an offline constraint-feasible approximate solution achieves high performance in problems such as this example that require operating close to constraint boundaries.

A. Unconstrained Solution

We first attempt to treat the problem as an unconstrained MDP and impose a high penalty $C \gg R$ for entering any of the constrained states. The resulting policy is shown in Figure 12. First thing to note is that the planner does not send

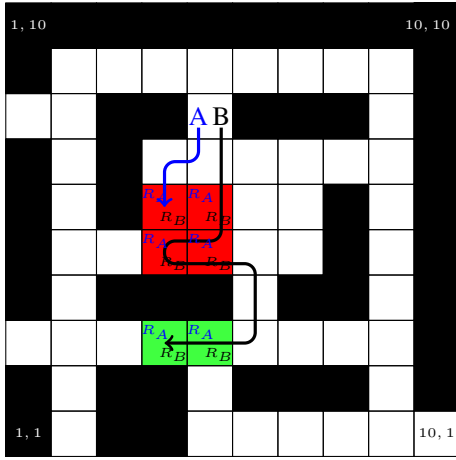


Fig. 10: The highest-reward paths through the rewards are constraint-infeasible

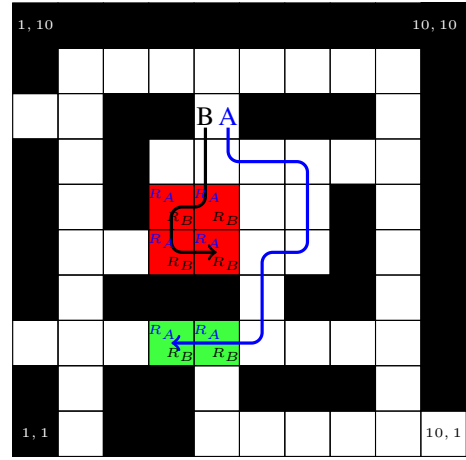


Fig. 11: Constraint-feasible paths

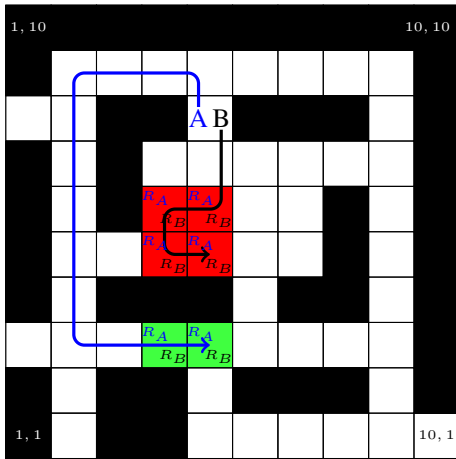


Fig. 12: Policy computed by value iteration when the constraint is a high negative reward

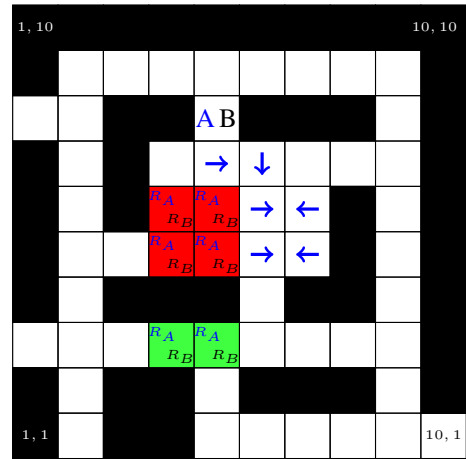


Fig. 13: Policy computed by MDP value iteration when the constraint penalty is lowered

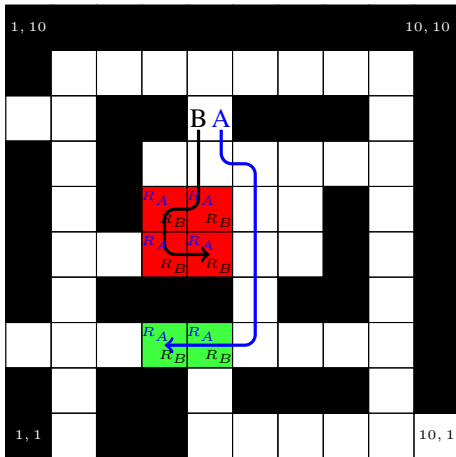


Fig. 14: Policy computed by value iteration when the constraint is a low negative reward

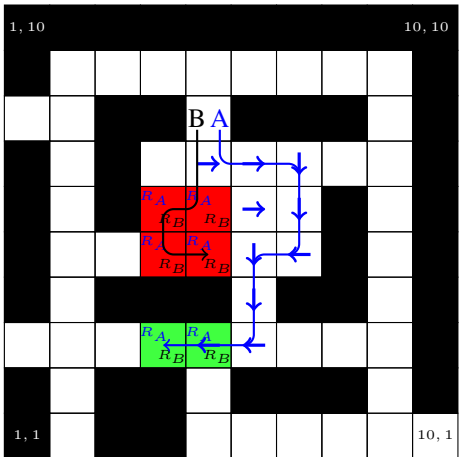


Fig. 15: Policy computed by the online constrained solver

Agent A into the courtyard because of the large penalties that are incurred by following that path. Therefore the vehicle moves around the courtyard, away from the danger zones, and reaches the rewards outside the courtyard. This policy is chosen because it is the only policy that guarantees that Agent A will not enter the heavily-penalized danger zone. The path through the courtyard is regarded as not worth the large negative reward that could be incurred if the Agent were to veer into the danger zone. In other words, the MDP planner fails to see that the risk associated with moving through the courtyard is an acceptable amount of risk, and acts too conservatively.

Figure 13 shows the outcome of lowering the constraint penalty. Since the constraint violation penalty is lower, the planner decides to follow a path through the courtyard. However, after reaching location (6, 6), the planner decides to move to the right since this action offers no risk of entering the danger zone, while still giving a probability of 0.05 of moving downwards and therefore closer to the rewards outside the courtyard. Therefore lowering the penalty makes the planner less conservative in one aspect (entering the courtyard), but the resulting policy is still inefficient. Lowering the penalty even farther leads to the outcome shown in Figure 14. A low penalty leads the planner to generate a policy that travels adjacent to the danger zone. Clearly, this policy is too risky and gives a constraint violation probability $\alpha > 0.1$. Thus in this problem as well, treating constraints as penalties generates policies that are either too conservative or too risky. The results of applying the proposed algorithm to the current problem are presented in the next section.

VIII. CONSTRAINED ONLINE ALGORITHM SOLUTION

The online algorithm presented in this work generates the policy shown in Figure 15. The online forward search algorithm with a finite search horizon (which for was set to 10 in this case) identifies that the path through the courtyard - moving right in state (5, 7) - is in fact constraint-feasible since the constraint violation probability of moving right in state (5, 7) is 0.05, and therefore lower than the threshold of 0.1. The forward search also uses the offline solution to see that at least one constraint-feasible action exists once state (5, 7) is reached, and therefore continues to explore this path. As the forward search reaches state (5, 3), it is recognized that a path to the reward at this location exists, and that path is also constraint-feasible (since the risk of constraint violation along this path from its starting location is 0.095). The online planner therefore switches its action at location (6, 5) to move down (out of the courtyard) to claim the reward at (5, 3). Thus the offline approximate solution provides a conservative, constraint-feasible policy while the online algorithm adjusts this policy for better performance. The offline exact value iteration algorithm presented in [3] also generates the same policy, but requires much more computational time (approximately 10 minutes, compared to approximately 5 seconds for the online algorithm) since it involves solving a Mixed Integer Linear Program (MILP) at every iteration.

IX. CONCLUSIONS

This work seeks to address the problem of planning in the presence of uncertainty and constraints. Such problems arise in many situations, including planning for a team of first responders (both humans and robots) operating in the aftermath of a CBRNE incident. The problem was framed as a POMDP with constraints, and it was shown that even in a relatively simple planning problem, constraints cannot be modeled simply as large penalties. The main contribution was an online algorithm that uses an approximate offline solution to ensure constraint feasibility while optimizing reward over a finite horizon. It was shown that our on-line algorithm generates policies comparable to an offline constrained POMDP algorithm, but is computationally more efficient.

X. ACKNOWLEDGEMENTS

This research was supported by the Office of Naval Research under grant number N00014-07-1-0749.

REFERENCES

- [1] *Applying Robotics to HAZMAT*, Richard V. Welch, Gary O. Edmonds, The Fourth National Technology Transfer Conference and Exposition, Volume 2 p 279-287
- [2] *Protecting Emergency Responders: Safety Management in Disaster and Terrorism Response*. Brian A. Jackson, John C. Baker, M. Susan Ridgely, James T. Bartis, Herbert I. Linn. Department of Health and Human Services, Centers for Disease Control and Prevention, National Institute for Occupational Safety and Health.
- [3] *Piecewise Linear Dynamic Programming for Constrained POMDPs*, Joshua D. Isom, Sean P. Meyn, Richard D. Braatz. Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (2008).
- [4] *Learning Policies for Partially Observable Environments: Scaling Up*, Michael L. Littman, Anthony R. Cassandra, Leslie Pack Kaelbling. In Proceedings of the 12th International Conference on Machine Learning (ICML-95), pp 362-370.
- [5] *Online Planning Algorithms for POMDPs*, Stephane Ross, Joelle Pineau, Sebastian Paquet, Brahim Chaib-draa. Journal of Artificial Intelligence Research 32 (2008) 663-704.
- [6] *Rollout Algorithms for Stochastic Scheduling Problems*. D. P. Bertsekas and D. A. Castanon. Journal of Heuristics, 5(1) 89-108 (1999).
- [7] *Real-Time Decision Making for Large POMDPs*, Sebastian Paquet, Ludovic Tobin, Brahim Chaib-draa. Advances in Artificial Intelligence, Volume 3501 (2005) pp 450-455.
- [8] *Perseus: Randomized Point-Based Value Iteration for POMDPs*, Matthijs T. J. Spaan, Nikos Vlassis. Journal of Artificial Intelligence Research 24 (2005) 195-220.
- [9] *SARSOP: Efficient Point-Based POMDP Planning by Approximating Optimally Reachable Belief Spaces*, H. Kurniawati, D. Hsu, W. Sun Lee. In Proceedings of Robotics: Science and Systems, 2008.
- [10] *Probabilistic Robotics*, S. Thrun, W. Burgard, D. Fox. MIT Press 2006.
- [11] *Time Constrained Markov Decision Processes*, A. Geramifard. Unpublished.