Planning under Uncertainty and Constraints for Teams of Autonomous Agents

by
Aditya Undurti

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Aeronautics and Astronautics
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June 2011 [September 2011]

Author .....
Department of Aeronautics and Astronautics
August 1, 2011

Certified by .....
Jonathan P. How
R. C. Maclaurin Professor of Aeronautics and Astronautics
Thesis Supervisor

Certified by .....
Cynthia L. Breazeal
Associate Professor, Program in Media Arts and Sciences

Certified by .....
Emilio Frazzoli
Associate Professor, Department of Aeronautics and Astronautics

Certified by .....
Nicholas Roy
Associate Professor, Department of Aeronautics and Astronautics

Accepted by .....
Eytan H. Modiano
Chair, Graduate Program Committee

Planning under Uncertainty and Constraints for Teams of Autonomous Agents

by

Aditya Undurti

Submitted to the Department of Aeronautics and Astronautics
on August 1, 2011, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Aeronautics and Astronautics

Abstract

One of the main advantages of unmanned, autonomous vehicles is their potential use in dangerous situations, such as victim search and rescue in the aftermath of an urban disaster. Unmanned vehicles can complement human first responders by performing tasks that do not require human expertise (e.g., communication) and supplement them by providing capabilities a human first responder would not have immediately available (e.g., aerial surveillance). However, for unmanned vehicles to work seamlessly and unintrusively with human responders, a high degree of autonomy and planning is necessary. In particular, the unmanned vehicles should be able to account for the dynamic nature of their operating environment, the uncertain nature of their tasks and outcomes, and the risks that are inherent in working in such a situation. This thesis therefore addresses the problem of planning under uncertainty in the presence of risk. This work formulates the planning problem as a Markov Decision Process with constraints, and offers a formal definition for the notion of "risk". Then, a fast and computationally efficient solution is proposed. Next, the complications that arise when planning for large teams of unmanned vehicles are considered, and a decentralized approach is investigated and shown to be efficient under some assumptions. However some of these assumptions place restrictions - specifically on the amount of risk each agent can take. These restrictions hamper individual agents' ability to adapt to a changing environment. Hence a consensus-based approach that allows agents to take more risk is introduced and shown to be effective in achieving high reward. Finally, some experimental results are presented that validate the performance of the solution techniques proposed.

Thesis Supervisor: Jonathan P. How
Title: R. C. Maclaurin Professor of Aeronautics and Astronautics

Acknowledgements

First, I would like to thank my advisor, Jon. His feedback, while sometimes harsh, was always fair and accurate and his guidance was invaluable in completing this thesis. I would also like to thank my committee members, Professors Cynthia Breazeal, Nick Roy and Emilio Frazzoli. It has been a pleasure working with them for the past four years. I am also grateful to Luca and Brett, the two readers of this thesis, for their timely feedback and for the many discussions over the years on problems addressed here and other related topics.

I would also like to express my gratitude to the students at the Aerospace Controls Lab, both current and former. I have learned far more from working with them than through formal coursework. They are all very bright and talented, and each brings with them their own special skills and strengths. Whether it be discussing MDPs with Alborz, getting coding help from Josh, or discussing random topics with Dan and Luke, they have all been great colleagues and made the lab an interesting place to be everyday. I would also like to give a special thanks to Philipp Robbel - all the things that we accomplished under the MURI project in the past four years would not have been possible had we not worked so well together.

Of course, lab and research were only one component of the MIT experience - perhaps more than anything else, the thing I will remember about my time at MIT in the years to come will be the amazing people I met and befriended. Neha, Parul, Dan, Sharmeen, Mrin and Simran - it looks like I finally did finish my (real) PhD thesis. Jen, Carolyn, Hoda - it was amazing seeing you all in India, we should do that more often. Fatima - I miss the late-night discussions, conversations and gossip we shared during the time you were here. Prabhod, Vimi, Swini, Sunaina - it has been an amazing experience working with you, and I regret having to leave behind the awesome team we built. I am also grateful that I got to spend time with Sheetal and Guru in the past couple of years, I only wish we had seen each other more often. I would also like to thank Ali Motamedi for the many pointless and content-free conversations that we had. Except of course for that one time when we solved all

problems known to science. I am also grateful to count among my friends Tara and Premal. Their friendship and affection are truly genuine - something I feel even now, years after we stopped seeing each other on a daily basis. I am also thankful for my long friendship with Kranthi, a friendship which has survived being on opposite sides of the world for four years. He managed to convince me (or should I say "konvince") that nothing should ever be spelled with a "C" when it can be spelled with a "K". And then of course there were Rajiv and Vikas - whether it be deciding where to go on a Friday night (and the question was always "where", never "whether"), or picking the right orange juice, or deciding which non-linear programming algorithm to use, or debating the existence of free will - life on the Cambridge side of the river would have been far less glorious had it not been for them.

Most importantly, I would like to thank my family. My grandparents in India have always had us in their prayers and I know I always have them in mine. My sister Arundhati and brother-in-law Sagar have always been caring, affectionate, and supportive. I know I do not call them or visit them as often as I would like, but they have always been and will always be in my thoughts. My wife Rasya has also been an invaluable source of support - her constant cheerfulness and humor never fail to lift my spirits, and for that I cannot imagine life without her. And finally, to my parents - twelve years ago they made an incredibly bold decision to leave everything they had in India and move to the US, trusting that we would succeed here. Life has never been the same since then, and I cannot even imagine where I would be today if they had not done so. The sacrifices they have made to give us the best opportunities in life are truly heroic and to them I dedicate this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

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 search and rescue in the aftermath of a Chemical, Biological, Radiological, Nuclear and Explosive (CBRNE) incident in an urban area [1, 2]. Examples from recent history of robots being deployed in such incidents include clean-up and reconnaissance operations after the 9/11 attacks in New York City [3] and the Fukushima nuclear reactor meltdown in Japan [4]. Sending human first responders to a location where such an incident has occurred could be dangerous [5]. Deploying autonomous vehicles, such as Unmanned Air Vehicles (UAVs) and Unmanned Ground Vehicles (UGVs) into a post-CBRNE search and rescue mission thus has obvious safety advantages. In addition, the human resources available might be limited - since it is likely that there will be a significant number of casualties spread over a substantial area, the available HAZMAT (Hazardous Materials) and EMS (Emergency Medical Services) personnel must be carefully and judiciously used. Using autonomous vehicles to supplement the human personnel, e.g. by performing some tasks that would otherwise have to be performed by a human, can be beneficial. Furthermore, UAVs and UGVs can perform tasks that humans would not be capable of performing, e.g. aerial surveillance and continuous vigilance. Thus there are multiple advantages to having autonomous vehicles when dealing with a CBRNE incident.

The objective for a team of first responders in the aftermath of a CBRNE incident

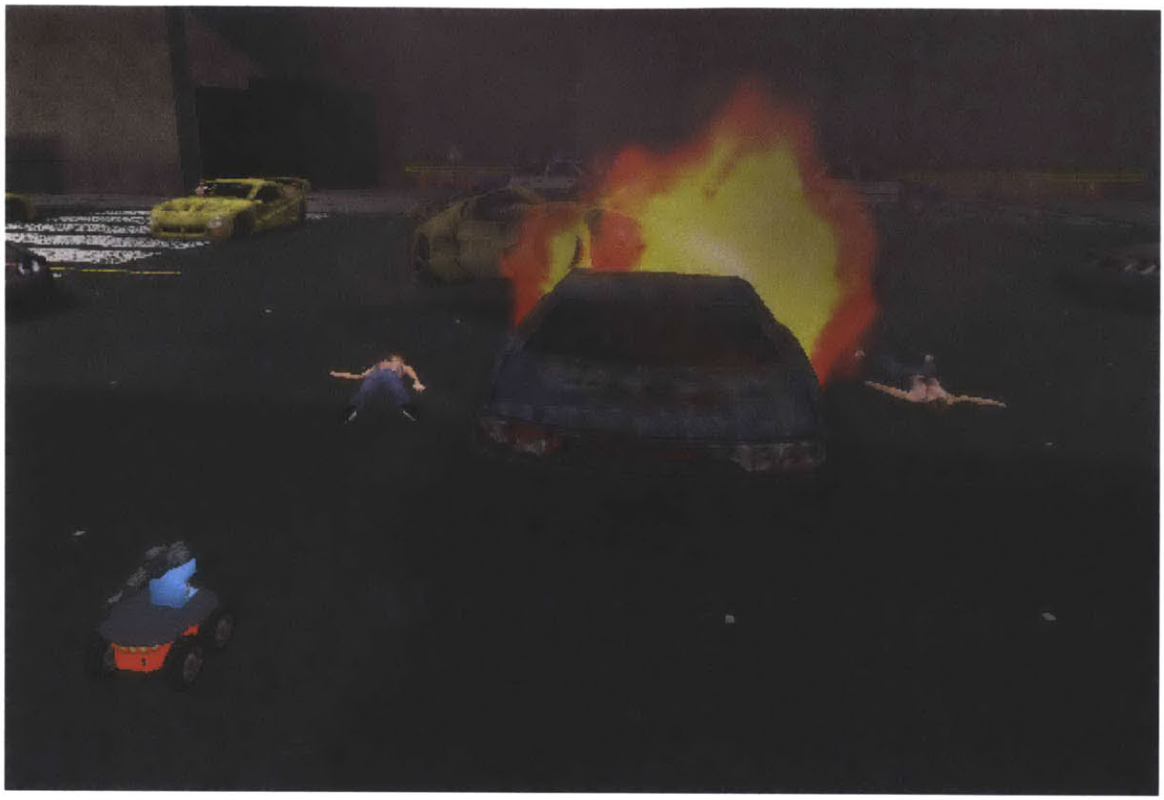

Figure 1-1: An autonomous robot operating in a dangerous post-CBRNE environment

is victim search and rescue, while keeping the first responders themselves safe [6]. This task is challenging because of the high uncertainty and dynamic nature of CBRNE incidents. The number of victims that need to be rescued, and their locations might not be known, or are known only approximately. The team of first responders might not have the ability to acquire this information directly and with certainty, making the environment **partially observable**. Spreading fires and contaminants could also be changing the numbers of victims and their locations even as the search and rescue mission is underway. Furthermore, the victims might be moving in ways that are hard to model, making the environment very **dynamic** and **stochastic** [7]. Acting in this fast-changing environment will require the agents to react quickly to local information, while retaining awareness of the overall mission. Achieving fast local response requires significant **decentralization** while retaining mission perspective requires team **communication and consensus**.

Furthermore, the responders and agents themselves have to stay safe. The team

cannot risk sending humans to areas that are suspected of having contaminants. Autonomous agents must also be used carefully, keeping away from known dangers. These safety considerations impose constraints on the mobility of the agents [8]. Victim rescue is also a time-critical task, and imposes time constraints within which the team must work. And finally, the incident commander, based on information that is not available to the team on the ground or the planner, might order agents to visit certain locations or take certain positions. All these factors set **constraints** within which the team must operate.

An additional complicating factor is that the response teams include both autonomous robots and human team mates. Treating a human the same as a robotic agent is unlikely to lead to good solutions, because of various factors. First, the performance of human agents will be affected by hard-to-quantify measures such as workload, stress, situational awareness, and skill level. Furthermore, these measures are not constant - the levels of stress and situational awareness, and perhaps even skill level, vary as functions of time. The exact dynamics are highly individual-dependent and history-dependent, e.g. a human agent that has just performed many tasks is much more likely to be stressed and exhausted than an agent that has not performed any tasks. Second, human agents have far more complex behaviors and responses to instructions than robotic agents. Even if a human is allocated a task by the central tasker, there is a chance that the human will not execute the instruction. This could be because the human is not paying attention, or because the human judges some other task to be of greater urgency and/or greater reward. The behaviors that a human agent is capable of are very diverse and extremely difficult to capture in any one model. Therefore, any planning scheme must take into account this **complex and unmodeled agent behavior**.

In summary, a response to a CBRNE scenario requires planning in an stochastic, dynamic, partially-observable and poorly-modeled environment for a decentralized team with complex agents in the presence of constraints. The objective of the team is to maximize the number of victims rescued. This is a very challenging and interesting problem that draws upon several areas of recent research, specifically 1) planning

in uncertain environments in the presence of constraints 2) planning for multi-agent teams that include complex unmodeled agents and 3) decentralized planning for multi-agent teams.

The key research question that will be addressed by this thesis is the following: *Can we generate plans in real-time for a large, decentralized team of complex agents while accounting for constraints and uncertainty in the system and the environment?* In the presence of various types of uncertainty, both modeled and unmodeled, creating plans that can guarantee constraints may be difficult. This thesis will begin by proposing a planning architecture that can provide guarantees on constraints in the presence of modeled uncertainty. Then, an extension of this architecture to centralized multi-agent teams will be proposed, and finally the architecture will be generalized to decentralized teams.

1.1 Challenges

Solving the research question posed above presents several significant challenges. They are:

- **Complexity**: Planning even for a single agent can be a difficult and computationally challenging problem. As we shall see in the Literature Review, much work has been done on the various aspects of planning, e.g. path planning and task planning. In all but the simplest cases, the number of possible paths and plans increases combinatorially, making an exhaustive search through all possibilities computationally infeasible.

- **Scaling**: Planning for multiple agents adds even more complexity. This is particularly true if some tasks require cooperation between several agents. The size of the planning problem grows exponentially in the number of agents, making real-time planning for multi-agent teams particularly difficult.

- **Uncertainty**: Uncertainty in the environment and the system's response to commanded plans must be accounted for in the planning process. Failure to

do so could lead to poor performance. Furthermore, the dynamics of the environment are poorly understood. Accounting for uncertain system response and unknown environment dynamics requires a combination of planning for known uncertainty and being able to replan fast to deal with unmodeled uncertainty. This in turn further motivates the need to reduce the computational complexity of the planner.

- **Feasibility**: An important feature of the CBRNE planning problem is the presence of safety constraints. Maintaining these safety constraints restricts the number of possible plans. Also, guaranteeing safety constraints in the presence of uncertainty is difficult. For instance, if the response of an agent to a command is uncertain, and that agent happens to be operating in an unsafe environment, even a "safe" command might lead to the execution of an unintended unsafe action. Thus a planner would have to account for such unintended behavior to maintain safety. On the other hand, the mission objectives entail operating in a risky environment and good performance may require taking some risk. Thus achieving a balance between risk and performance is an important consideration when planning in constrained environments.

- **Consensus**: Since the environment is dynamic, agents need to be able to respond quickly to local changes. However, they must not lose sight of the team objective. In addition, they must not take any action that might jeopardize other agents working on other tasks. Thus agents must not only respond rapidly, but achieve consensus with other team members. This needs to happen sufficiently fast to react to the environment. Achieving consensus while also acting in real time poses a significant challenge.

In the next section, we review some of the approaches and formulations that have been taken to address some of the challenges described above.

1.2 Literature Review

As stated above, planning for a team of first responders require the ability to plan in an uncertain, dynamic environment with constraints. The problem of planning under uncertainty and constraints has been formulated and addressed in various ways, including Travelling Salesman Problems (TSPs), Vehicle Routing Problems (VRPs), Linear Programming (LP), Model Predictive Control (MPC), Consensus, Markov Decision Processes (MDPs) and Constrained Markov Decision Processes. The work in each of these fields is reviewed below.

1.2.1 Probabilistic Travelling Salesman Problem

Work in the existing literature has looked extensively at various aspects of the task allocation and vehicle routing problem in the presence of uncertainty. Related problems include the Probabilistic Travelling Salesman Problem (PTSP) presented in [9] which is similar to the well-known Travelling Salesman Problem (TSP) with the exception that every customer i has a probability $p_i \neq 1$ of being realized. These p_i are assumed to be independent. It is shown that in some cases, not accounting for the stochasticity of the customers (and solving the problem as a standard TSP) results in expected cost being as much as 30% higher [9]. However, the problem considered in [9], and the TSP in general, does not contain any information about demands at specific nodes, and only a single vehicle (or salesman) is assumed. While [9] does not consider time constraints on visit times, this is partly addressed in [10]. However neither [9] nor [10] allow for time to be spent at each customer (i.e. no loiter time constraints are considered). And finally, the p_i are assumed to be independent, whereas in the CBRNE scenario dependencies between tasks might be critical.

1.2.2 Stochastic Vehicle Routing Problem

Another related problem that has been studied extensively is the Stochastic Vehicle Routing Problem (SVRP) [11, 12]. Many variations of this problem exist - the SVRP with stochastic demands [13], with stochastic customers [14], with stochastic travel

times [15], and with both stochastic customers and demands [12]. The SVRP also allows for multiple vehicles [11]. Thus the SVRP is better suited for the CBRNE scenario, and although algorithms exist to solve the problem, it is computationally much harder to solve [12]. The SVRP also assumes the probability of customers' existence is independent.

Due to the computational complexity of the PTSP and SVRP, the algorithms are intended to be run just once to come up with an allocation, with no replanning as the plan is being executed. Indeed, the desire to eliminate online reallocation and replanning is one of the motivations for developing algorithms dedicated to solving the PTSP and the SVRP [9, 16]. Thus these algorithms are incapable of adapting to completely unexpected information - for instance, in the PTSP case, if a customer were to appear in a city that was not part of the TSP tour at plan time, this new information would never be accounted for and the customer would be missed. Since unexpected incidents are characteristic of a CBRNE scenario, a planning technique that plans only once initially and is too computationally complex to be run on-line would not be suitable.

1.2.3 Dynamic Vehicle Routing

Other approaches that explicitly account for the stochasticity in tasks include work by Pavone, Bisnik, Frazzoli and Isler [17] and by Pavone, Frazzoli and Bullo [18]. In this work, the objective of the planner in [17] is to minimize the average waiting time between the appearance of a demand and the time it is visited by a vehicle. For a single vehicle, a Single Vehicle Receding Horizon (sRH) policy is developed, that plans a TSP tour through all existing demands and executes the most optimal fragment of length η of this tour before replanning. When there are multiple vehicles available, the space is divided into Voronoi regions and areas of dominance for each vehicle are established. Upper bounds on the average waiting time are also derived when the spatial distribution of the tasks is known. The work presented in [18] is an extension of [17] because it allows for customer impatience, i.e. existing tasks disappear after a certain time, with the time itself being a random variable. The minimum number of

vehicles required to service each task with a probability of at least $1 - \epsilon$. A strategy to assign tasks to vehicles is also presented. The strategy is essentially similar to that presented in [17], with a TSP tour over all existing demands at each iteration [18].

1.2.4 Linear Programming

Another approach to solving planning problems is to formulate the problem as a Linear Program (LP), possibly with integer variables thus making it a Mixed-Integer Linear Program (MILP). However, the size of the LP (the number of decision variables) that needs to be solved grows combinatorially in the number of agents and tasks. Thus several approximate algorithms exist that provide a feasible solution to the LP without necessarily achieving the optimal solution. Algorithms that make use of such an approach include the Receding Horizon Task Allocation (RHTA) algorithm and its decentralized version, the Receding Horizon Decentralized Task Allocation (RDTA) algorithm. RHTA only allocates those m tasks that are nearest to each of the vehicles - nearest either spatially or temporally [19]. RHTA is typically formulated as a Mixed-Integer Linear Program, and by restricting the size of the look-ahead horizon, the computational complexity is greatly simplified. And by choosing a horizon size that is sufficiently large, the obvious disadvantages of using a myopic, greedy allocation strategy are reduced although not completely eliminated. RHTA is also capable of handling time constraints, loiter constraints [20] and multiple vehicles, including sending multiple vehicles to the same task in the same time window. RHTA has also been extended to include robustness when task information is uncertain. In [21], the value of the various tasks (in this case, the number of targets to be struck) is assumed to be uncertain with a known variance and a robust version of RHTA is developed. Robust RHTA sacrifices high-reward plans in exchange for safe plans, and thus reduces the variation in the reward for different realizations of task rewards [21]. Other work such as [22] takes a similar approach and looks at the value of assigning vehicles to an information-gathering mission to reduce uncertainty in tasks.

Work presented in [23] looks at the problem of assigning vehicles to depots at various locations in a city to respond to traffic incidents. Since the location of these

incidents and their resource demands are unknown, although the stochastic distribution is known. The allocation of vehicles to depots is chosen to maximize a "quality of service" metric, which is defined as the lower bound of the probability that all the resources requested by potential incidents will be satisfied. The problem is formulated an LP with probabilistic chance constraints, which in turn can be written as a deterministic MILP.

Other work that follows the receding horizon paradigm is work by Lauzon et al. [24] and Wohletz [25]. The strategies presented in both these papers optimize a cost function over a finite horizon (the *predictive horizon* in [24]) and execute the resulting strategy for a short time before replanning again. Chandler et al. have proposed formulating the task allocation problem as a network flow optimization problem [26]. Work by Sujit and Beard [27] looks at the problem of creating coalitions of vehicles to accomplish certain missions, and uses a particle swarm optimization technique to solve it.

1.2.5 Auctioning and Consensus

In some planning problems, not only is computational complexity an issue but communication between agents as well. Reliable communications to a central planner might not exist, or a central planner itself might not exist. Various approaches to solving this decentralized problem have been suggested, and among the more promising are auctioning and consensus algorithms. Auctioning as a means for solving assignment problems in a distributed fashion was first proposed by Bertsekas in 1979 [28]. Bertsekas and Castanon then applied auction algorithms to linear network flow problems [29] and to assignment problems [30]. This work was subsequently extended to cases where not all agents could submit auction bids in a synchronous fashion [31]. Sujit, George and Beard [27] describe a decentralized auction algorithm for vehicles with limited sensor and communication ranges to decide what task to perform in a distributed fashion. Auctioning has recently emerged as a mechanism by which to perform multi-robot planning and coordination. Berertton, Gordon, Thrun and Khosla [32] use an auctioning mechanism to decouple the planning problem for a

team of robots each planning independently. The planning problem is first written as a linear program and then decomposed into a smaller planning problems for each individual robot using the Dantzig-Wolfe decomposition. The decomposition introduces additional costs in the objective function of the individual robots. A central planner sets these additional costs, following which individual robots plan separately. The solutions are sent back to a central planner that then changes the costs in a manner that improves the overall team performance. This semi-decentralized way of solving the planning problem is shown to converge to the optimal solution in a reasonable number of iterations. Another auctioning-based consensus algorithm used to solve task planning problems is the Consensus-Based Bundle Allocation (CBBA) algorithm [33]. In CBBA, each agent individually builds a "bundle" of tasks that it would like to perform. The agents then communicate with each other and resolve conflicting claims based on a set of rules that are designed to optimize team reward. CBBA is guaranteed to be within 50% of the optimal solution. Several extensions to CBBA have been proposed, including Asynchronous CBBA (ACBBA) [34] which eliminates the need for all agents to be synchronized. One major advantage of CBBA is the fact that it is decentralized and does not become computationally infeasible as the number of agents increases. Another advantage is that each individual agent solves a much simpler problem, and all agents can solve their respective planning problems in parallel. For these reasons decentralized planners are preferred when solving to optimality is not strictly required.

1.2.6 Markov Decision Processes

A *plan* is a sequence of actions that the vehicle is expected to execute in the future. However, in deciding this sequence, the planner makes certain assumptions about the outcomes of actions in the future. When these outcomes are not realized perfectly, i.e., when there is error, the planner will have to recompute from the new, previously-unexpected state. But it would be ideal if the planner could account for the fact that future actions are uncertain, and come up with contingency plans for all possible outcomes. In this case, planning needs to happen only once initially, and the vehicle

will thereafter have a large set of plans for every possible realization, i.e., it has a *policy*. This is the key idea behind a Markov Decision Process (MDP) [35]. The solution to an MDP is a policy, which is defined as a mapping from states to actions [36]. A policy therefore tells the vehicle what to do in every possible state the vehicle might encounter. MDPs are a very useful tool for planning, but the computational cost for most practical problems is extremely high [36]. A further complication is that the current state might not be known exactly, because of noisy or incomplete sensing. This makes the problem a Partially-Observable Markov Decision Process (POMDP).

There are three main challenges when solving a POMDP - first, POMDPs do not explicitly allow constraints. Constraints can be treated as large negative rewards, but it will be shown later that for finite negative rewards, the constraints cannot be guaranteed to hold. Second, the POMDP solution depends on the accuracy of the transition and observation models that were used in computing the policy. If these models are inaccurate, the POMDP solution cannot be guaranteed to be optimal. In the CBRNE search and rescue problem, we cannot be sure that the transition model is accurate since the rate at which victims are dying is uncertain and is probably changing in uncertain ways. Furthermore, secondary incidents might occur unpredictably, giving rise to new victims. A new policy would have to be computed using the best transition and observation models available. This brings out the third major disadvantage of POMDPs, which is their computational complexity. Recomputing an exact POMDP policy in real-time (using value iteration, for instance) is extremely difficult since the computational complexity grows very rapidly with the size of the problem. Thus we need to turn to approximate methods.

Several approximate methods exist for MDPs and POMDPs, for example [37, 38]. POMDPs can be either discrete or continuous. For discrete POMDPs, factored methods make use of the structure of the problem to direct the search for a policy [39]. Point-based methods [40] [41] have also been proposed for approximate solutions. In point-based methods, only a subset of the total belief space is used in computing the policy. Other methods include belief-space compression using principal components analysis [42] [43], and a Belief Roadmap (BRM) algorithm that relies on factoring

the covariance matrix in a linear Gaussian system [44]. POMDPs can also be solved as linear programs by using the Bellman equation (which must be true for an optimal policy) as a constraint [45]. For continuous POMDPs, extensions of discrete algorithms such as PERSEUS can be used [46]. Some recent work [46] has begun to address the problem of solving POMDPs in the presence of constraints. All these strategies are *offline*, i.e. a policy is computed for all possible states and beliefs ahead of time, before the execution phase.

One disadvantage with approximate offline algorithms is that the computed policies depend on the modeled system dynamics, i.e. on the transition model. In the event that the model is inaccurate, the policies can no longer be guaranteed to be optimal. Therefore, being able to recompute policies online is important. Several online algorithms for solving POMDPs have been developed, for example Real-Time Belief Space Search (RTBSS) [47] and Rollout [48]. Several other algorithms are summarized in a review paper by Ross et al. [49]. Generally speaking, on-line algorithms first obtain an offline approximate solution. Then, the expected value of executing an action in the current belief space is estimated by searching through a finite depth AND-OR tree with actions and observations as the branches and belief states as the nodes. The estimated cost-to-go for each of the leaves of the tree is estimated with the approximate solution obtained previously. In the next section, we discuss the problems with guaranteeing hard constraints in the presence of uncertain dynamics, and in particular when using on-line algorithms to compute plans in such an environment.

1.2.7 Constrained MDPs

Several authors have previously investigated adding constraints to a standard MDP, leading to what is known in the literature as a *constrained MDP* [50, 51]. Work by Chen and Blackenship [52, 53], and by Piunovskiy and Mao [54] demonstrated the application of dynamic programming to solving constrained MDPs. The optimality equations that are satisfied by the optimal policy are derived, existence and uniqueness of solutions are explicitly proven, and a modified dynamic programming operator is introduced and shown to be a contraction, i.e. repeated application of

the operator to a policy is shown to lead to a fixed-point solution which is also the optimal solution. The work is extended by Chen and Feinberg [55, 56] and shown to yield non-randomized policies, i.e. a deterministic action in every state. Dolgov and Durfee extend the constrained MDP literature to include more complex formulations, in particular cases with multiple constrained quantities with multiple discount factors [57, 58]. The problem is formulated as a Mixed-Integer Linear Program, and the proposed algorithm finds both randomized and deterministic policies, although solving for deterministic policies is computationally more difficult [58]. Zadorojniy and Shwartz study the robustness of optimal policies to changes in problem parameters, i.e. whether the a computed policy remains constraint feasible when the transition model (the system dynamics) and the constraint parameters are modified. They show that optimal policies are robust to small changes in the transition model, but not necessarily in the constraint parameters since a change in the constraint parameters can lead to infeasibility [59]. Others have looked at using reinforcement learning to solve constrained MDPs. A summary of those methods is provided by Geibel [60]. These methods include `LinMDP`, a linear programming formulation of the constrained MDP problem; `WeiMDP`, which converts a constrained MDP into an unconstrained MDP by adding the constraint as a reward penalty with a weight than can be chosen by the designer [61]; and `RecMDP`, a recursive Q-Learning algorithm that gives suboptimal solutions [62].

1.3 Proposed Approach

In this work, we formulate the multi-agent constrained planning problem as a Markov Decision Process (MDP). We choose the framework of MDPs for several reasons. First, MDPs naturally incorporate uncertainty and noise and are particularly convenient when the noise is not Gaussian and the system dynamics are not linear. Second, MDPs provide a means by which to encode the mission objectives in the form of a reward model. There are no restrictions on the form and nature of the reward model - the model can be non-linear, continuous or discrete, convex or non-convex. Third,

MDPs can be extended to include the *partially observable* case, and that is one of the areas this work will investigate. Partially Observable Markov Decision Processes (POMDPs) are capable of deciding when to take actions that improve knowledge of the world and when to act upon information that is already available.

The flexibility and power of MDPs comes at the price of computational complexity. MDPs suffer from "the curse of dimensionality" [63], i.e. the computational effort required to solve an MDP grows dramatically as the size of the problem state space increases. This is especially true in the case of multi-agent systems since the size of the problem is directly related to the number of agents in the team. As we have seen previously, much effort has been done in the broader planning community to develop fast, approximate MDP solvers. This work will use and extend some of these solvers. Another shortcoming is that MDPs do not naturally incorporate hard constraints. This work therefore investigates an extension to the standard MDP, called a Constrained MDP (C-MDP). The C-MDP formulation of the planning problem will then be solved using a fast on-line algorithm developed as part of this work. The issues that arise as the size of the team grows will then be investigated, and some approximate techniques will be used to simplify the multi-agent planning problem.

1.4 Contributions

As indicated above, the contributions of this work are to the broad area of planning under uncertainty and constraints, and more specifically to the field of Constrained MDPs. They may be summarized as follows.

Contribution 1: Planning Under Uncertainty and Constraints This work begins by proposing an extension to the standard MDP. The extension is the *constraint model* and provides a means by which to incorporate any general constraints into an MDP, thus turning it into a C-MDP. A fast, computationally efficient on-line algorithm is then presented to solve the C-MDP. This algorithm relies on a finite horizon forward search to optimize the reward and an off-line approximate solution

to estimate constraint feasibility beyond the search horizon. It will be shown that this algorithm achieves good performance in constrained environments. However, the off-line approximate solution becomes a computational bottleneck, and this becomes particularly acute when the environment and the constraint map are not static. The complexity of this off-line approximate solution grows exponentially in the number of agents, and therefore becomes important when we attempt to extend this algorithm to large teams. This directly motivates the third contribution. But before addressing the multi-agent problem, we complete the development of the algorithm by extending it to continuous domains.

Contribution 2: Planning Under Uncertainty and Constraints in Continuous Domains Solving MDPs defined over continuous domains (continuous state and action spaces) is in general a difficult problem. A solution to an MDP is a *policy* - a mapping from states to actions. In other words, a policy prescribes which action to take in every state. In discrete MDPs, this mapping can be expressed (conceptually, if not actually) as a look-up table. In continuous MDPs, representing policies is significantly more challenging. One approach is to express the policy as the gradient of the *value function*, which is the cost-to-go from any state. However, solving for the value function is just as computationally challenging as solving for the policy itself. One method that has been used successfully in the literature [64–66] for finding the value function (which can be applied to both discrete and continuous domains) is *function approximation*. Function approximation finds the value function by assuming it to be a linear combination of a set of *basis functions*. These basis functions, also known as *features*, are picked by the designer to be appropriate for the specific problem of interest. In this work, we extend MDP function approximation techniques for constrained continuous MDPs. We show that when the underlying system is continuous, approximating continuous domains by discretization can yield poor results even for reasonable discretizations. On the other hand, function approximation with the right set of features achieves good performance.

Contribution 3: Planning for Multiple Agents Under Uncertainty and Constraints The size of an MDP for multi-agent teams grows exponentially in the number of agents. This "curse of dimensionality" makes the use of MDPs for teams of agents computationally difficult. However, there are some significant simplifications that can be made under sensible assumptions. One key assumption we make is to assume *transition independence*, i.e. the action of one agent only affects its own dynamics, and not those of other agents. While this is not strictly true (the action of one agent is connected to the actions of other agents at the high-level planning stage) it is a good approximation. However we do need to account for the fact that the agents are still coupled through rewards and constraints. Some rewards may require a joint action by more than one agent, and some constraints might be applicable to the entire team rather than individual agents. Specifically this work investigates the case where there is constraint coupling and proposes a mechanism by which agents can plan their own individual actions while accounting for their impact on the team constraints. In planning for themselves, the agents must make some assumptions about the actions of the other agents. Specifically, they have to assume that the probability of the other agents violating a team constraint (henceforth referred to as *joint constraints*), defined as the *risk*, is fixed. This assumption will introduce some conservatism in the agents' behavior, and in the next contribution we seek to eliminate that conservatism through team communication and consensus.

Contribution 4: Distributed Planning Under Uncertainty and Constraints The final contribution of this work looks at the benefit of having team communication and consensus in the presence of joint constraints. As mentioned previously, agents can plan their own individual actions even in the presence of constraint coupling provided they make some assumptions about the actions of other agents. Specifically, they have to assume the other agents will not take actions that exceed a certain threshold of risk. In this section, we remove that assumption and instead allow the agents to communicate with each other and arrive at a consensus about how much risk each agent can take. This risk negotiation can take place throughout the mission,

so that as the environment changes and some agents have to take more (or less) risk than was originally foreseen, the team can adapt. The properties and complexity of this risk negotiation are discussed, and it will be shown that significantly higher rewards can be expected particularly in highly constrained environments.

The rest of the thesis is organized as follows. Chapter 2 discusses the first contribution, Chapter 3 the second, Chapter 4 the third and Chapter 5 the fourth. Chapter 6 will present some experimental results in both virtual environments with simulated physics (Unreal Tournament) and actual hardware (RAVEN). Chapter 7 concludes with a summary and some suggestions for future work.

Chapter 2

Planning Under Uncertainty and Constraints

2.1 Approach

Markov Decision Processes (MDPs) provide a broad and generalized framework within which to formulate and solve planning problems in uncertain environments. However, the standard MDP formulation does not allow for the incorporation of hard constraints, such as fuel constraints and path constraints [35].

This work investigates an enhancement to the standard MDP by adding constraints, called a *Constrained Markov Decision Process* [50]. This work proposes an online algorithm with finite look-ahead for solving the constrained MDP. 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.

2.2 Literature Review

The problem of planning in the presence of uncertainty and constraints has been addressed in existing literature. Various ways of formulating such problems have been proposed, including Mixed Integer Linear Programming, randomized algorithms and Markov Decision Processes. Some of these approaches have been reviewed in Chapter 1. For instance, Ono and Williams investigated the problem of constraints in the presence of process noise [67, 68] and the planning framework used in their work is a Mixed-Integer Linear Program (MILP). Obstacle avoidance constraints are written as chance constraints, i.e the probability of violating a constraint is required to be below a certain threshold. The probabilistic constraints are then converted into tightened deterministic constraints where the tightening factor is proportional to the bound on the constraint violation probability. A similar constraint-tightening approach was used by Luders and How [69], but with the planning problem formulated as a path planning problem and is solved with Rapidly-Exploring Random Trees (RRTs).

Generally speaking, a standard Markov Decision Process does not explicitly include constraints [35]. However some work has been done to extended the MDP framework to include hard constraints, an extension called *Constrained Markov Decision Processes* [50]. Recent work includes work by Dolgov [57], which investigates resource constraint problems and introduces the resource constraints into a standard MDP. However, Dolgov's work mostly looked at off-line methods [58], whereas in this work we seek fast on-line algorithms to deal with a dynamic environment. Additionally, in this work the resource we consider is risk which is a continuous variable whereas Dolgov et al. do not consider continuous state spaces. Solving MDPs with continuous state spaces is difficult and is best accomplished using function approximation techniques. This approach will be investigated in this work (Chapter 3).

Similar work has been done in the field of Partially Observable Markov Decision Processes (POMDPs). The problem of solving Constrained POMDPs has been addressed by Isom et al. [46], but the proposed methods are again off-line methods that are computationally too expensive for fast planning. In this work, we propose using

online algorithms to solve constrained MDPs and POMDPs due to their speed and scalability.

2.3 Problem Formulation

As stated in the previous chapter, we formulate the planning problem as an MDP. An MDP is formally defined the tuple $< S, A, R, T >$, where S is the state space and A is the set of actions. Throughout this work, unless otherwise specified, we assume that $|S|$ and $|A|$ is countable and finite. $R(\mathbf{s}, \mathbf{a}) : S \times A \rightarrow \Re \quad \forall \mathbf{s} \in S, \mathbf{a} \in A$ is the *reward model*, and gives the reward (in this work, a non-negative real number) for taking action $\mathbf{a}$ in state $\mathbf{s}$. $T(\mathbf{s}'|\mathbf{s}, \mathbf{a}) : S \times A \times S \rightarrow \Re \quad \forall \mathbf{s}', \mathbf{s} \in S, a \in A$ is the *transition model* that gives the probability of transitioning from state $\mathbf{s}$ to state $\mathbf{s}'$ under action $\mathbf{a}$, and hence $\sum_{\mathbf{s}' \in S} T(\mathbf{s}'|\mathbf{s}, \mathbf{a}) = 1 \quad \forall \mathbf{s}, \mathbf{s}' \in S, \mathbf{a} \in A$. A solution to an MDP is a *policy* $\pi(\mathbf{s})$. Formally defined as $\pi(\mathbf{s}) : S \rightarrow A \quad \forall \mathbf{s} \in S$, a policy is a mapping from states to actions - in other words, a policy prescribes as action in every state. An *optimal* policy $\pi^*(\mathbf{s})$ is one that maximizes the expected reward over time, and therefore satisfies the following optimality condition.

$$\pi^* = \arg\max_{\pi} E\left[\sum_{t=0}^{T} \gamma^t R(\mathbf{s}_t, \pi(\mathbf{s}_t))\right] \tag{2.1}$$

In the above cost calculation, γ is the *discount factor* applied to future rewards and is between 0 and 1.

This work seeks to address the problem of solving a *constrained* MDP. A constrained MDP is defined as the tuple $<S, A, R, T, C>$. S, A, R and T are exactly as defined previously. $C(\mathbf{s}, \mathbf{a}) : S \times A \rightarrow \Re \quad \forall \mathbf{s} \in S, \mathbf{a} \in A$ is the *constraint model*, and the value it returns is defined as the *constraint penalty*. In this work, the constraint penalty is defined as follows. If action $\mathbf{a}$ in state $\mathbf{s}$ is disallowed for the system, i.e. violates a constraint, $C(\mathbf{s}, \mathbf{a}) = 1$, and 0 otherwise. Furthermore, states for which $C(\mathbf{s}, \mathbf{a}) = 1 \quad \forall \mathbf{a}$ are always absorbing states. Thus constraint-infeasible states are terminal states.

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

$$\pi* = \arg\max_{\pi} E\left[\sum_{t=t_0}^{T} \gamma^t R(\mathbf{s}_t, \pi(\mathbf{s}_t))\right] \tag{2.2}$$

$$\text{s.t.} \quad E\left[\sum_{t=t_0}^{T} C(\mathbf{s}_t, \pi(\mathbf{s}_t))\right] \leq \alpha \tag{2.3}$$

Since the constraint penalty is always non-negative, $E\left[\sum_{t=t_0}^{T} C(\mathbf{s}_t, \mathbf{a}_t)\right] \leq 0$ if and only if $C(\mathbf{s}_t, \mathbf{a}_t) = 0 \ \forall t$. 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. This probability is henceforth defined as *risk*. Note that in some existing literature (e.g. Dolgov and Durfee 2004, [70], Cavazos-Cadena and Montes-De-Oca 2003, [71] and Geibel 2005 [72]) risk is defined differently - as the probability that the *reward* will lie outside an acceptable bound defined by the designer. However, in this work risk is defined as the probability that a constraint will be violated, the constraints being defined by the constraint model just described.

The standard approach to dealing with constraints in MDP literature [35] [61] is to impose a penalty for violating the constraints. The method is discussed below, and along with a discussion of its shortcomings.

2.3.1 Constraints as Penalties

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

$$\hat{R}(\mathbf{s}_t, \mathbf{a}_t) = R(\mathbf{s}_t, \mathbf{a}_t) - MC(\mathbf{s}_t, \mathbf{a}_t) \tag{2.4}$$

where M is a large positive number. This new reward function $\hat{R}(\mathbf{s}_t, \mathbf{a}_t)$ is used to solve a standard unconstrained MDP, $< S, A, \hat{R}, T >$. Varying the value of M can make the policy more risky or more conservative. However, a counterexample

presented below and shown in Figure 2-1 illustrates that an M value does not always exist for all risk levels.

In the problem shown in Figure 2-1 an agent starts in state s_1 at time 0. 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 state s_4 with probability 0.1. State s_4 is a constrained state ($C(s_4, .) = 1$), and therefore also an absorbing state. The `hop` action is therefore rewarding but risky, while the `walk` action is safe but unrewarding. The objective is to maximize the path reward while keeping risk (probability of entering s_4) below 0.1. By inspection, the optimal policy $\pi*$ corresponding to a risk level $\alpha = 0.1$ is to `hop` in state s_1 but `walk` in state s_2.

In a standard MDP, we would prevent the agent from entering state s_4 by imposing the constraint penalty M on that state. 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 `hop` is chosen in s_2 and `walk` in s_1 - the planner switches between being too conservative (only `walk`s) to too risky (only `hop`s).

The intent of this simple example is to illustrate that incorporating constraints as negative penalties does not always yield good policies, particularly in cases where some risk is required to achieve high performance. In this work we present an algorithm that does not have this shortcoming, but first we provide a more thorough discussion of the notion of risk.

2.3.2 Definition of Risk

In the discussion above, we defined risk as the probability that a constraint will be violated. However, there are two different ways to measure this quantity - defined here as *accumulated* and *memoryless*. *Memoryless risk* is the risk that is expected in the future, at times $t > t_0$ where t_0 is the current time. Since the agent cannot perform any more actions after violating a constraint (constrained states are always absorbing states), it is reasonable to assume that no constraints have been violated

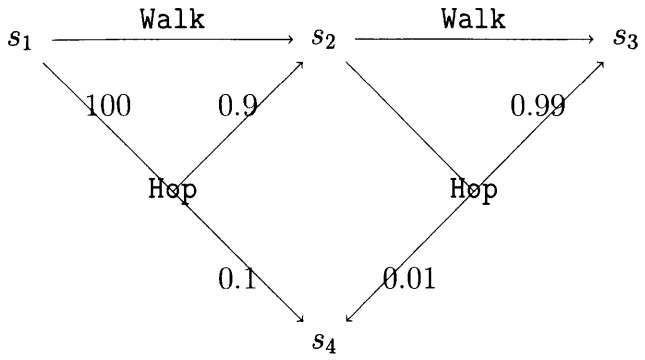

Figure 2-1: A counter example to show that a penalty value corresponding to the optimal policy does not always exist (Courtesy: Alborz Geramifard)

so far. Risk that was taken in the past was not realized and is therefore ignored, and only the risk that is expected in the future is constrained to be less than α. *Accumulated risk* on the other hand is the total risk that has been accumulated by the agent since $t = 0$, i.e. we set $t_0 = 0$ in Equation 2.3. Under this definition, we require that the *total* risk - the risk taken in the past, plus the risk that is expected in the future - to be less than α. Thus the planning problem for the case where we use accumulated risk is given by

$$\pi* = \arg\max_{\pi} E\left[\sum_{t=0}^{T} \gamma^t R(\mathbf{s}_t, \pi(\mathbf{s}_t))\right] \tag{2.5}$$

$$\text{s.t.} \quad E\left[\sum_{t=0}^{T} C(\mathbf{s}_t, \pi(\mathbf{s}_t))\right] \leq \alpha \tag{2.6}$$

Note that the only difference between Equations 2.2-2.3 and Equations 2.5-2.6 is the time over which the risk and reward are computed. In Equations 2.2-2.3 that time is from the current time t_0 to the problem termination time T, whereas in Equations 2.5-2.6 the time is over the entire time 0 to T. We notice that since the policy from time 0 to time $t_0 - 1$ has already been executed, the left side of Equation 2.6 can be broken into two terms as follows.

$$E\left[\sum_{t=0}^{T} C(\mathbf{s}_t, \pi(\mathbf{s}_t))\right] = E\left[\sum_{t=0}^{t_0-1} C(\mathbf{s}_t, \pi(\mathbf{s}_t))\right] + E\left[\sum_{t=t_0}^{T} C(\mathbf{s}_t, \pi(\mathbf{s}_t))\right]$$

$$= V_{C0} + E\left[\sum_{t=t_0}^{T} C(\mathbf{s}_t, \pi(\mathbf{s}_t))\right]$$

Where we define $V_{C0} \equiv E\left[\sum_{t=0}^{t_0-1} C(\mathbf{s}_t, \pi(\mathbf{s}_t))\right]$. A key advantage of using an online algorithm is that since the policy from time 0 to t_0 has already been executed, the actions in states $\mathbf{s}_0, \cdots \mathbf{s}_{t_0}$ are *not* decision variables, and the online MDP solver does *not* have to account for the various values that V_{C0} may take. At any time t_0, V_{C0} is a constant and a parameter that is supplied to the online MDP solver, but does not have to be incorporated as a state variable. Similarly, the past reward does not have to be incorporated as a state variable, since it does not affect the reward-to-go.

When using an offline solver - for example value iteration - running the solver for T iterations (from time 0 to time T) naturally computes the accumulated risk, therefore making it unnecessary to incorporate the past risk in the state. However if the offline solver were used to solve for a policy from t_0 to T, and accumulated risk from times 0 to T were being used as the risk definition, then the past risk (from time 0 to $t_0 - 1$) *does* have to be incorporated as a state. The inverse is also true - if the offline solver were being used to compute a policy from time 0 to T, and memoryless risk were being used as the risk definition, then the past risk again has to be incorporated as a state variable. This highlights one significant advantage of using an online approach - the choice of memoryless risk or accumulated risk can have a major impact on the complexity of an offline algorithm, whereas an online algorithm can incorporate either by simply keeping track of the past risk V_{C0} without having to explicitly condition its policy on all possible values of V_{C0}.

Both methods of measuring risk are equally valid, and choice of accumulated versus memoryless risk is problem-dependent. In problems where a policy is expected to be executed repeatedly, accumulated risk is a more sensible measure, whereas in problems where a policy is expected to executed only once (or in which the the policy is frequently recomputed) memoryless risk might be a better measure. In this thesis, we focus our attention mostly on problems with memoryless risk, although some results and comments for problems with accumulated risk are also provided.

2.4 Proposed Solution

The proposed solution to the constrained MDP defined in Equations 2.2 and 2.3 is to use an on-line forward search to optimize the reward and check for constraint feasibility up to a horizon of length D. Constraint feasibility for times beyond D is ensured by using a risk-to-go estimate that is computed off-line. The solution requires two quantities to be tracked - the reward R and the constraint C. For this reason, we maintain two separate value functions - one associated with the reward $R(\mathbf{s}, \mathbf{a})$ which we call $V_R(\mathbf{s})$ and another associated with the constraint value $C(\mathbf{s}, \mathbf{a})$ which we call $V_C(\mathbf{s})$. The algorithm presented has two components - an off-line component where an approximate solution for V_C is obtained, and an on-line component where the off-line estimate for V_C is refined and V_R is computed over the finite horizon to select the best action.

2.4.1 Off-line Risk Estimate

The overall goal of the planner is to maximize the expected reward obtained while keeping the risk below a threshold, in this case α. In the off-line component of the algorithm, we solve for a policy π_c^* that will *minimize* the risk. If the minimum risk from a certain state is below α, 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 risk bound. During the on-line portion of the algorithm (described in detail the next section) we use this guarantee to ensure constraint feasibility for times beyond the on-line planning horizon.

Therefore, we first obtain the minimum risk, $U_C^*(\mathbf{s})$, by solving the following unconstrained optimization problem:

$$U_C^*(\mathbf{s}) = \min_{\pi} E\left[\sum_{t=0}^{D} C(\mathbf{s}_t, \mathbf{a}_t)\right] \tag{2.7}$$

If, for any state $\mathbf{s}$, $U_C^*(\mathbf{s}) \leq \alpha$, then there exists at least one policy (the optimal policy π_c^*) that guarantees that starting from $\mathbf{s}$, the constraints will never be violated. In

solving the optimization problem shown in Equation 2.7, we may use approximate algorithms for unconstrained MDPs. The only requirement is that the solver provide an upper bound when minimizing. In that case, we know that if the approximate value for $U_C(\mathbf{s})$ returned by the solver remains less than α, we can guarantee that the true risk-to-go $U_C(\mathbf{s})$ will also be less than α. Thus the system will remain constraint-feasible if we use the corresponding approximate policy π_c^*.

2.4.2 On-line Reward Optimization

During the on-line part of the algorithm, we compute the future expected reward $V_R(\mathbf{s})$ and the future expected risk $V_C(\mathbf{s})$ using a forward search. The previously-obtained minimum risk policy is then evaluated at the leaves of the tree to ensure constraint feasibility beyond the planning horizon. The forward search is executed by solving the following set of equations:

$$\pi^*(\mathbf{s}) = \arg\max_{\mathbf{a}\in\mathbf{a}_C} [R(\mathbf{s},\mathbf{a}) + \sum_{\mathbf{s}'} T(\mathbf{s}',\mathbf{s},\mathbf{a}) V_R(\mathbf{s}')] \tag{2.8}$$

$$V_R(\mathbf{s}) = \max_{\mathbf{a}\in\mathbf{a}_C} [R(\mathbf{s},\mathbf{a}) + \sum_{\mathbf{s}'} T(\mathbf{s}',\mathbf{s},\mathbf{a}) V_R(\mathbf{s}')] \tag{2.9}$$

$$\mathbf{a}_C = \{\mathbf{a} : Q_C(\mathbf{s},\mathbf{a}) < \alpha\} \tag{2.10}$$

$$Q_C(\mathbf{s},\mathbf{a}) = C(\mathbf{s},\mathbf{a}) + \sum_{\mathbf{s}'} T(\mathbf{s}',\mathbf{s},\mathbf{a}) V_C(\mathbf{s}') \tag{2.11}$$

$$V_C(\mathbf{s}) = Q_C(\mathbf{s},\pi^*(\mathbf{s})) \tag{2.12}$$

Note that the terms $V_R(\mathbf{s}')$ and $V_C(\mathbf{s}')$ are obtained by solving Equations 2.8 - 2.12 recursively. The recursion is terminated at a depth equal to the desired search horizon, and at that depth values of $V_R(\mathbf{s}') = 0$, $V_C(\mathbf{s}') = U_C^*(\mathbf{s}')$ are used. Algorithm 1 is the complete algorithm, implemented as the subroutine `Expand`.

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 minimum risk U_C^*. For nodes that are not leaf nodes, the algorithm looks through all possible successor states $\mathbf{s}'$ by looping through all possible actions. Any successor state that that does not satisfy the constraints ($V_C(\mathbf{s}') < \alpha$), is not considered. For

Algorithm 1 The Constrained MDP Forward Search Algorithm

Function Expand($\mathbf{s}$,U_C,D)
if $D = 0$ **then**
  $V_C(\mathbf{s}) = U_C(\mathbf{s})$; $V_R(\mathbf{s}) = 0$; $\pi(\mathbf{s}) = \pi_C(\mathbf{s})$
  return $\pi(\mathbf{s})$, $V_R(\mathbf{s})$,$V_C(\mathbf{s})$
else
  for $a \in A$ **do**
    $[V_R(\mathbf{s}'), V_C(\mathbf{s}'), \pi(\mathbf{s}')] = \text{Expand}(\mathbf{s}', U_C, D-1)$
    $Q_R(\mathbf{s}, \mathbf{a}) = \sum_{\mathbf{s}'} V_R(\mathbf{s}')P(\mathbf{s}', \mathbf{s}, \mathbf{a}) + R(\mathbf{s}, \mathbf{a})$
    $Q_C(\mathbf{s}, \mathbf{a}) = \sum_{\mathbf{s}'} V_C(\mathbf{s}')P(\mathbf{s}', \mathbf{s}, \mathbf{a}) + C(\mathbf{s}, \mathbf{a})$
    $\pi'(\mathbf{t}) = \pi(\mathbf{s}) \forall \mathbf{t} \neq \mathbf{s}$
    $\pi'(\mathbf{s}) = \mathbf{a}$
  end for
  $\mathbf{a}_C = [\mathbf{a} : Q_C(\mathbf{s}, \mathbf{a}) < \alpha]$
  $\pi'(\mathbf{s}) = \arg\max_{\mathbf{a} \in \mathbf{a}_C} Q_R(\mathbf{s}, \mathbf{a})$
  $V_R(\mathbf{s}) = Q_R(\mathbf{s}, \pi'(\mathbf{s}))$
  $V_C(\mathbf{s}) = Q_C(\mathbf{s}, \pi'(\mathbf{s}))$
  return $\pi'(\mathbf{s})$, $V_R(\mathbf{s})$,$V_C(\mathbf{s})$
end if

those states that do satisfy $V_C(\mathbf{s}') < \alpha$, the `Expand` routine is called recursively to get the expected reward and the expected constraint penalty for that state. The action that provides the best reward V_R is returned as the best action to execute in the current state.

The inputs to `Expand` are the current state $\mathbf{s}$, the planning horizon length D, and the minimum risk (or its overestimate) U_C. The planning algorithm then begins the cycle of planning and executing. First, the function `Expand` is called on the current state. `Expand` computes the best action to execute as already discussed in the previous paragraph. It is discussed in more detail in the next paragraph. Once the action is executed the state $\mathbf{s}$ is updated and the `Expand` routine is again called on the most current belief. This cycle is repeated until execution is terminated. The complete algorithm is shown in Algorithm 2.

Off-line Risk Estimate as an Admissible Heuristic

The off-line minimum risk estimate serves as the risk-to-go estimate for the on-line forward search. In this section, we show why the minimum risk serves as a good

Algorithm 2 Pseudo-code for an on-line algorithm to solve constrained MDPs

```
1: Function ConstrainedOnlineMDPSolver()
   Static:
   s_c : The current state of the agent
   T :The current search tree
   D : Expansion Depth
   U_C : An upper bound on V_C
   Φ(s, a): System dynamics
2: s_c ← s_0
3: Initialize T to contain only s_c at the root
4: while ExecutionTerminated() do
5:    Expand(s_c, U_C, D)
6:    Execute best action a for s_c
7:    s_c ← Φ(s_c, a)
8:    Update tree T so that s_c is the new root
9: end while
```

heuristic that provides both good reward performance and constraint feasibility.

Assume that search horizon is of length $D \geq 1$, and the current state is $\mathbf{s}_0$. Suppose the optimal policy (one that satisfies Equations 2.2 and 2.3) has a reward value function of $V_R(\mathbf{s})$, and a risk value function of $V_C(\mathbf{s})$. Also, suppose that $V_C(\mathbf{s}_0) < \alpha$, i.e. we start from a feasible state. $U_R(\mathbf{s})$ is the reward-to-go for $V_R^*(\mathbf{s})$ and $U_C(\mathbf{s})$ is the risk-to-go for $V_C^*(\mathbf{s})$. And finally, $U_C^*(\mathbf{s})$ is the minimum risk, i.e the risk value function obtained by solving Equation 2.7.

Suppose we were to use an estimate for $U_R(\mathbf{s})$ as the reward-to-go estimate, and $U_C(\mathbf{s})$ as the risk-to-go estimate. There are several possibilities to consider - first, that we *underestimate* true $U_C(\mathbf{s})$ for all $\mathbf{s} \in S$, and call this underestimate $\underline{U}_C(\mathbf{s})$. In this case, there is a possibility that the on-line forward search will return a policy that is infeasible, by directing the agent into a state from which no further constraint feasible policies exist. This possibility is shown in Figure 2-2. State $\mathbf{s}_2$ is high-reward ($R = 100$) but infeasible ($U_C > \alpha$), but since $\underline{U}_C$ underestimates the risk, an infeasible path is explored. On the other hand, the on-line forward search investigates more paths than are actually feasible and therefore will not miss the actual optimal path - in other words, it is optimistic. This is made especially clear by considering the case where we *overestimate* $U_C(\mathbf{s})$, shown in Figure 2-3. The overestimate is designated at

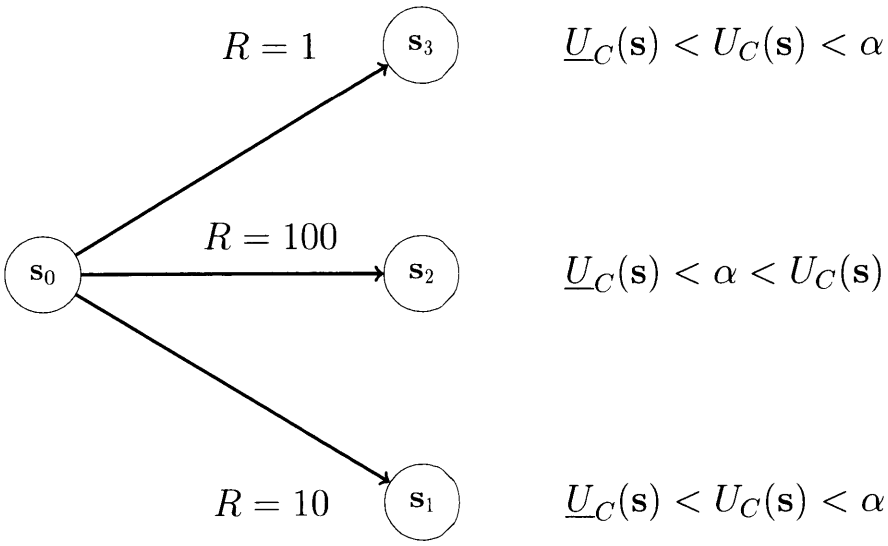

Figure 2-2: Online forward search with an optimistic heuristic, one that underestimates the risk. Infeasible paths are incorrectly assumed to be feasible

$\overline{U}_C(\mathbf{s})$. In this case, the on-line forward search always returns feasible plans but can be poor in reward performance since it rules out paths that are in reality feasible. In Figure 2-3, state $\mathbf{s}_1$ is in fact a high-reward, feasible state ($R = 10$, $U_C < \alpha$) but since the heuristic $\overline{U}_C$ overestimates the risk that state is ruled out in the forward search. In fact, it is possible that *no* feasible paths will be found if $\overline{U}_C$ is a sufficiently high overestimate.

Thus, a good heuristic for $U_C(\mathbf{s})$ is an underestimate that accounts for the fact that in some states, there might be no feasible actions. The minimum risk, $U_C^*(\mathbf{s})$ serves as one such heuristic. If a risk-free action exists in a particular state $\mathbf{s}$, then $U_C^*(\mathbf{s}) = 0$ in that state. If all actions in that state are risky, then $U_C^*(\mathbf{s})$ returns the least risky action. If the least risky action has risk greater than the threshold α, then $U_C^*(\mathbf{s}) > \alpha$ and the on-line forward search eliminates any path that visits state $\mathbf{s}$. At the same time, $U_C^*(\mathbf{s})$ is guaranteed to be an underestimate to $U_C(\mathbf{s})$ (it is the lowest possible risk) and therefore does not pessimistically eliminate plans that are feasible.

For the reward-to-go estimate $U_R(\mathbf{s})$, there are two possibilities - one is that the heuristic is optimistic, i.e. $\underline{U}_R(\mathbf{s}) > U_R(\mathbf{s}) \quad \forall \mathbf{s} \in S$, the second is that it is pessimistic with $\overline{U}_R(\mathbf{s}) < U_R(\mathbf{s}) \quad \forall \mathbf{s} \in S$. However for good reward performance, the heuristic

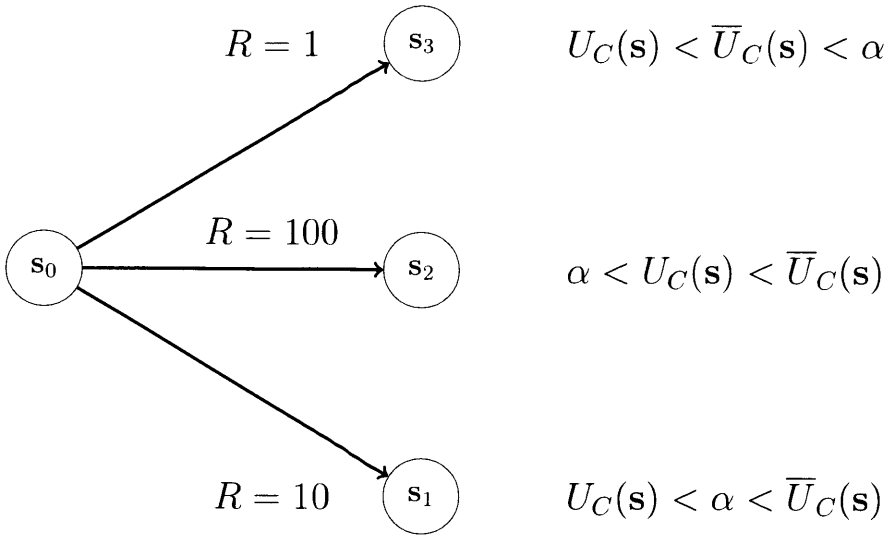

Figure 2-3: Online forward search with a pessimistic heuristic, one that overestimates the risk ($\overline{U}_C > U_C$) - feasible, high-reward paths are incorrectly assumed to be infeasible

also has to be ordered the same as the optimal reward value function, which means that if $U_R(\mathbf{s}_1) > U_R(\mathbf{s}_2) \quad \forall \mathbf{s}_1, \mathbf{s}_2 \in S$, then $\overline{U}_R(\mathbf{s}_1) > \overline{U}_R(\mathbf{s}_2)$ and $\underline{U}_R(\mathbf{s}_1) > \underline{U}_R(\mathbf{s}_2)$. In the absence of this property, being optimistic does not necessarily provide any significant benefit. Finding this ordering essentially requires solving the complete C-MDP problem, and thus we simply set $U_R(\mathbf{s}) = 0 \quad \forall \mathbf{s} \in S$ and avoid incurring any additional computation. The properties of heuristics for the reward and risk, and their impact on the quality of the solution generated are summarized in Table 2.1.

Table 2.1: Impact of heuristic properties on the performance of the On-line C-MDP forward search algorithm

	$U_R > V_R^*$	$U_R < V_R^*$
$\overline{U}_C > U_C$	Feasible, Conservative	Feasible, conservative
U_C^*	Feasible, optimistic	Feasible, pessimistic
$\underline{U}_C < U_C$	Infeasible	Infeasible

2.5 Results

The example problem considered is a robot navigation problem with some constraints. We present three sets of results - first, we show that treating the constrained MDP problem as a standard MDP with constraints treated as penalties does not yield the optimal policy. Second, we show that the on-line approximate solution presented here performs reasonably well when compared to the exact off-line optimal solution. And finally, we show that the definition of risk that is used - memoryless versus accumulated - makes a significant difference in the reward performance.

2.5.1 Constraints as Penalties

The dynamic model for the agent is shown in Figure 2-4. 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 perpendicular directions. In the case shown in Figure 2-4, 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 2-5. 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 cannot execute any further actions once it enters one of these states. The vehicle is not allowed to violate any constraints with a probability of more than 0.05, i.e. the allowed risk $\alpha \leq 0.05$.

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 2-6. 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, modelling the constraints as negative rewards will lead to either very conservative or very risky behavior. We will also show that the proposed on-line algorithm with an off-line 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 modeled 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 2-7. 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.

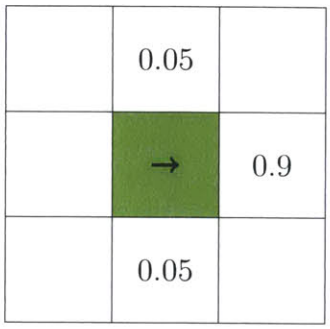

Figure 2-4: Vehicle dynamic model

Figure 2-8 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 2-9. The planner assumes that the reward at location $(4, 4)$ (which is higher than the reward

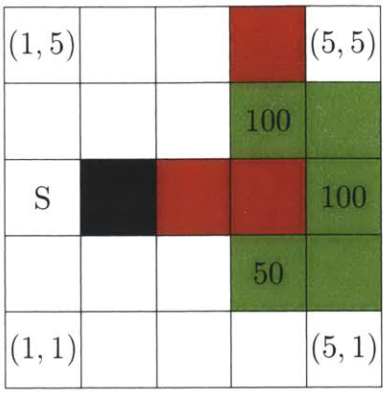

Figure 2-5: The MDP problem set up

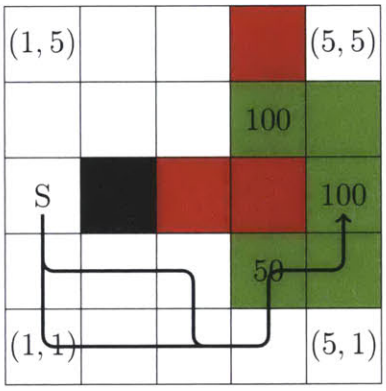

Figure 2-6: Constraint-feasible paths

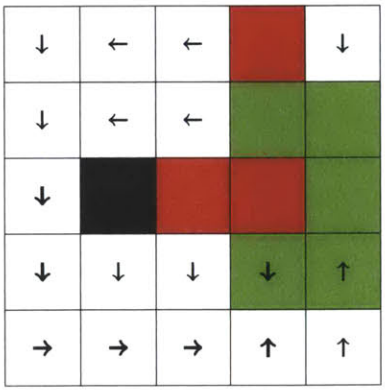

Figure 2-7: The policy computed by MDP value iteration when the constraint is modeled as a high negative reward leads to a very conservative policy that fails to achieve high reward. The nominal path from the start state is shown in bold.

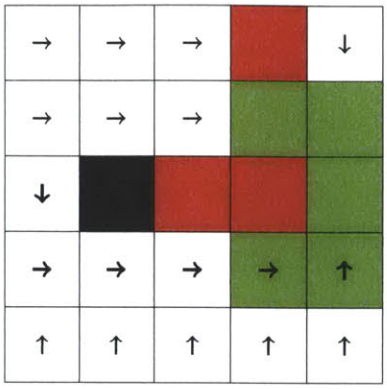

Figure 2-8: The policy computed by MDP value iteration when the constraint penalty is lowered. The nominal path from the start state is shown in bold.

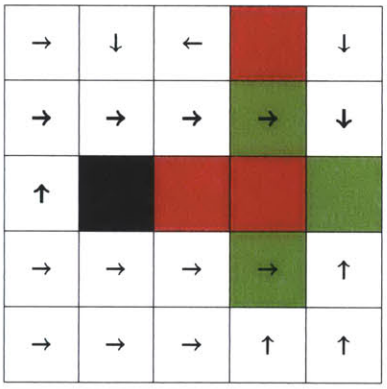

Figure 2-9: The policy computed by MDP value iteration when the constraint is modeled as a low negative reward leads to a policy that violates the safety constraint. The nominal path from the start state is shown in bold.

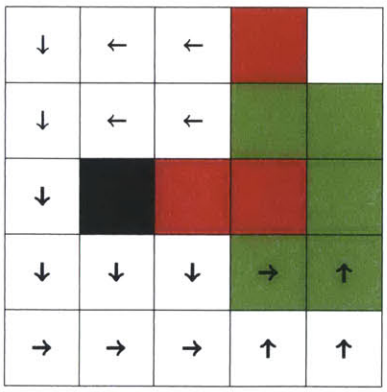

Figure 2-10: The policy computed by the on-line constrained solver. The solver correctly recognizes that going right from location $(4, 2)$ is constraint-feasible and yields high reward. The nominal path from the start state is shown in bold.

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 on-line constrained MDP algorithm presented in this work provides a general way to constrain risk while optimizing performance.

The on-line algorithm presented in this work generates the policy shown in Figure 2-10. The algorithm first generates a conservative off-line 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 2-7. The on-line 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 on-line planner therefore chooses to move right and claim the reward at $(5, 3)$. Thus the off-line approximate solution provides a conservative, constraint-feasible policy while the on-line algorithm adjusts this policy for better performance.

2.5.2 Approximate On-line Solution vs. Optimal Off-line Solution

It is clear that the on-line solution presented here is an approximate solution - the search horizon for the forward search is finite, and the reward-to-go and risk-to-go

used at the end of that horizon are approximate. Thus we should expect that the policies generated on-line will yield a lower reward than those generated by the exact optimal solution. In order to test loss in reward performance, we compare the on-line solution against an exact off-line optimal solution method proposed by Dolgov et al. [58]. The problems upon which the two algorithms were tested were similar to the problem shown in Figure 2-5, but with a larger 10×10 grid size. Rewards and constraints were placed randomly in the environment. Performance was measured as a function of the *constraint density*, which is defined as the fraction of states in the environment that are constrained states (shown in red in Figure 2-5). The number of reward states was kept fixed at 5 out of a total grid size of 100 states. For each value of the constraint density, 40 problems were randomly generated and the two algorithms applied to each. The results are shown in Figure 2-11.

First, we note that the reward for both methods decreases as the constraint density increases, since more rewards become infeasible as the number of constraints in the environment increases. Also as expected, the approximate on-line policy under performs the optimal policy, on average by a factor of 2. The gap between the two seems lower in more highly constrained environments, but this mostly due to the fact that both the optimal policy and the approximate policy both achieve low reward in these domains.

Its important to note that both the optimal and the approximate methods were using *accumulated risk*, i.e. the total risk (including the past risk) is constrained to be less than α. This was done to facilitate comparison with the offline optimal policy. In the next set of results, we show that the definition of risk used makes a significant difference to the reward performance.

2.5.3 Memoryless Risk vs. Accumulated Risk

We expect that using memoryless risk (only constraining future risk) will yield higher reward than using accumulated risk (constraining total risk). In this section, we show that the definition of risk does make a significant difference in the policies computed, and therefore the performance of the system. We again used a problem set-up similar

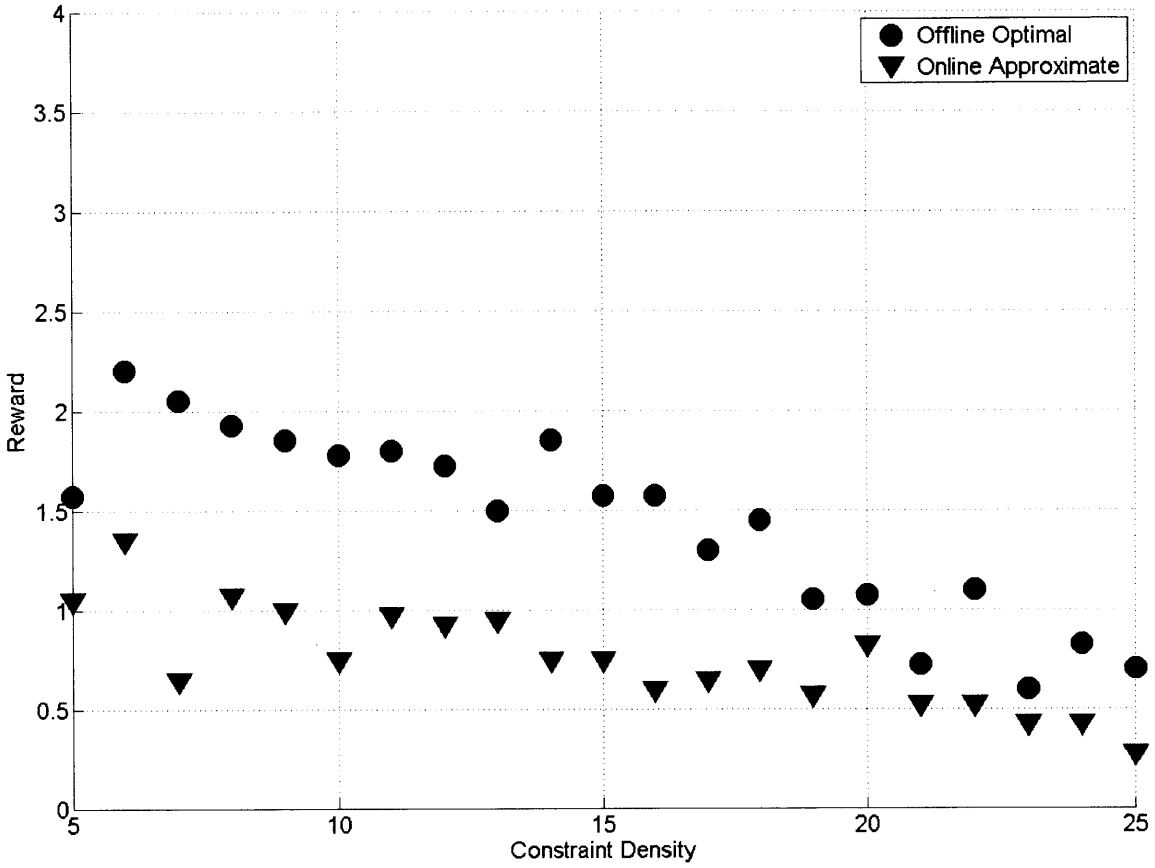

Figure 2-11: Comparison of the approximate online solution with the optimal, off-line solution

to that shown in Figure 2-5. As in the previous set of results, we used a 10×10 grid size and place rewards and constraints at random locations. The number of rewards was kept fixed at 5, and the number of constraints - the constraint density - was varied. For each value of the constraint density, 40 such problems were generated and the on-line approximate method was used to solve each problem with both memoryless risk and accumulated risk. The results are presented in Figure 2-12.

As expected, the reward in the memoryless case was found to be significantly higher than for the accumulated case. Furthermore, the difference is greater for higher constraint densities. This is particularly noticeable when we account for the fact that total reward is in fact lower for both methods when the constraint density is high. Again this is to be expected, since an agent operating in a highly constrained environment is likely to take more risk. By adding all the risk that the agent has

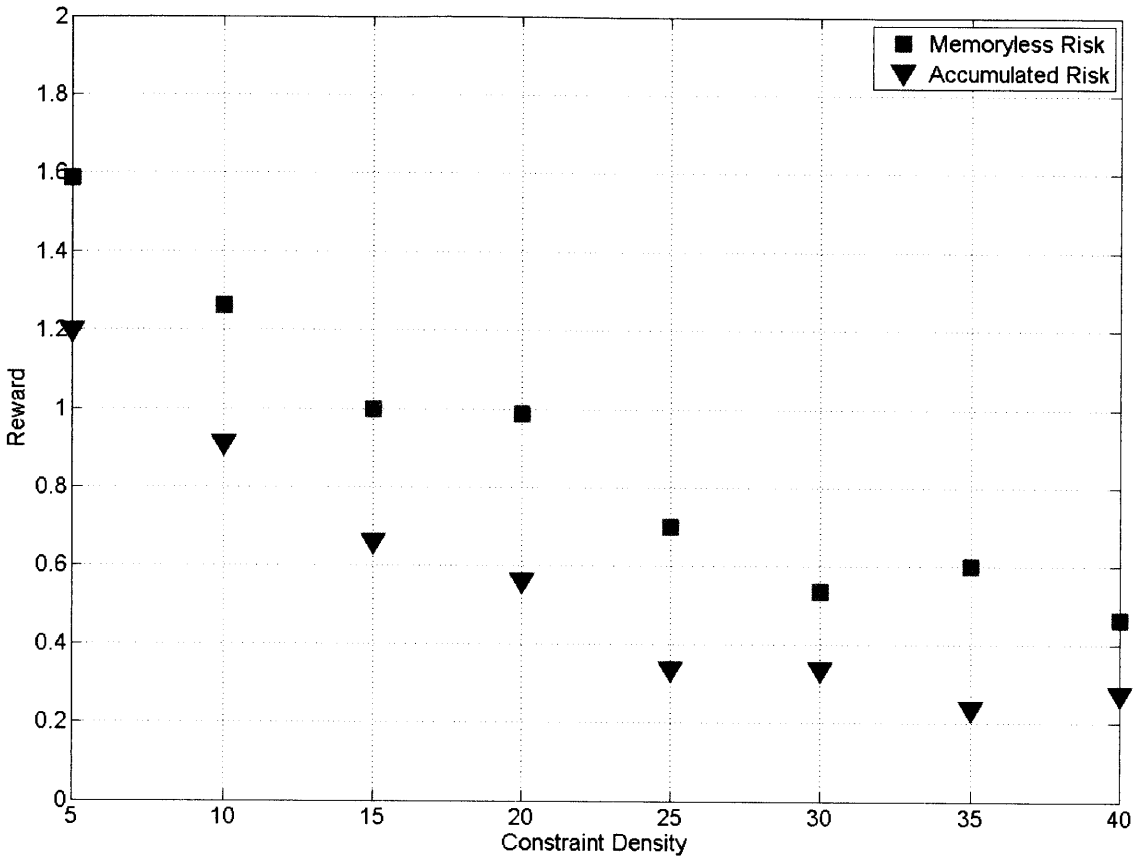

Figure 2-12: Difference in reward performance due to the choice of type of risk

taken in the past - which rises quickly in highly constrained environments – the agent's behavior is much more severely restricted. In fact, comparing Figures 2-11 and 2-12, we see that in the highly constrained cases (constraint density greater than 20) the Memoryless Risk case actually achieves *higher* reward than even the optimal policy with Accumulated Risk. We will observe in Chapter 4 that this effect becomes especially important in the multi-agent case, where risk is accumulated quickly even in less constrained cases due to the presence of many agents. Thus we see that the definition of risk that is used can have a significant impact on the overall performance of the agent. This difference has rarely been considered explicitly in the existing literature, but will be considered in the results and applications to be presented later in this work.

Another variation in the way risk can be measured arises if the agent cannot fully observe its environment - an assumption that we have made thus far by formulating

the problem as an MDP. If the agent cannot observe its environment, i.e., the environment is *partially observable* - then we must also specify whether the risk constraint is to be imposed on the *true state* (which the agent does not know with certainty) or on the *belief state* (which is the probability distribution that gives the likelihood of the agent being in any given state). A constraint on the true state will yield more conservative policies than a constraint on the belief state. While it is not the main focus of this work, we briefly discuss the extension of this work to the partially observable case.

2.6 Extension to POMDPs

An important variation of MDPs are *partially observable* Markov Decision Processes (POMDPs). In a POMDP, knowledge of which state the system is in is uncertain - instead of perfect knowledge of past and current states, we only have a *probability distribution* over all states, defined as the *belief*. As the system executes actions, this probability distribution changes in a manner described by the transition model. In addition, the system receives observations, potentially noisy, about its current state. The relationship between the observations and the current state is given by the *observation model*.

Formally, a POMDP is defined by the tuple $< S, A, Z, R, T, O >$ where S, A, R and T are exactly as defined for an MDP. Z is the set of possible observations, and $O(z, \mathbf{s})$ where $z \in Z, \mathbf{s} \in S$ is the observation model that returns the probability that observation z will be made in state $\mathbf{s}$. The belief is defined as $b(\mathbf{s}) \quad \forall \mathbf{s} \in S$ and gives the probability that the system is in state $\mathbf{s}$. A *solution* to a POMDP is a policy defined over the belief space, $\pi(b)$. The policy determines which action to take at every point in the belief space. The belief is computed from the transition and observation models, and the previous belief $b(\mathbf{s}_{t-1})$, the received observation z_{t-1} and action $\mathbf{a}_{t-1}$. Thus the belief at time t is related to the belief at time $t-1$ by the

following relation.

$$b(\mathbf{s}_t) = \eta O(z_{t-1}|\mathbf{s}_{t-1}, \mathbf{a}_{t-1}) \sum_{\mathbf{s}_{t-1} \in S} T(\mathbf{s}_t|\mathbf{s}_{t-1}, \mathbf{a}_{t-1}) b(\mathbf{s}_{t-1}) \tag{2.13}$$

$$\eta = 1/ \sum_{\mathbf{s}_{t-1} \in S} b(\mathbf{s}_{t-1}) \sum_{\mathbf{s}_t \in S} T(\mathbf{s}_t|\mathbf{s}_{t-1}, \mathbf{a}_{t-1}) P(z_{t-1}|\mathbf{s}_{t-1}, \mathbf{a}_{t-1})$$

An initial belief, $b(\mathbf{s}_0)$ is assumed to be known. Note that to simplify notation, we henceforth define $b_t \equiv b(\mathbf{s}_t)$. As in an MDP, the *optimal* policy is one that maximizes the expected future reward (given by the reward model R), shown below.

$$\pi^* = \arg\max_{\pi} E\left[\sum_{t=0}^{T} \sum_{\mathbf{s}_t \in S} [R(\mathbf{s}_t, \pi(b_t))|b_t]\right] \tag{2.14}$$

The belief b_t is computed using the Equation 2.14, where $\mathbf{a}_{t-1} = \pi(b_{t-1})$ and the initial belief b_0 is assumed to be known. A *constrained* POMDP (C-POMDP) is defined as the tuple $< S, A, Z, R, T, O, C >$, where the constraint model C is defined the same as for MDPs. An optimal policy for a constrained POMDP maximizes the expected future reward subject to a constraint on the expected risk, as given below.

$$\pi^* = \arg\max_{\pi} E\left[\sum_{t=0}^{T} \sum_{\mathbf{s}_t \in S} [R(\mathbf{s}_t, \pi(b_t))|b_t]\right] \tag{2.15}$$

$$\text{s.t.} \quad E\left[\sum_{t=0}^{T} \sum_{\mathbf{s}_t \in S} [C(\mathbf{s}_t, \pi(b_t))|b_t]\right] \leq \alpha \tag{2.16}$$

Several algorithms exist to solve unconstrained POMDPs. Off-line algorithms include POMDP value iteration [73], Point-Based Value Iteration (PBVI) [74] and on-line algorithms include RTBSS, Rollout and others that are summarized in [49]. Off-line algorithms exist for constrained POMDPs [46], and in this work we extend the on-line algorithm for constrained MDPs to constrained POMDPs. As in the MDP case, there are two parts to the algorithm - first, we compute the minimum risk off-line, and use this minimum risk policy to ensure constraint feasibility during the on-line reward optimization. The minimum risk is computed by solving the following the

unconstrained POMDP.

$$U_C(\mathbf{s}) = \min_{\pi} E\left[\sum_{t=0}^{T}\sum_{\mathbf{s}_t \in S} C(\mathbf{s}_t, \pi(b_t))|b_t\right] \tag{2.17}$$

Note that Equation 2.17 is the POMDP equivalent of Equation 2.7. As in the MDP case, the above unconstrained POMDP can be solved using either exact or approximate methods. The only requirement is that any approximate solution be an *overestimate* to the exact $U_C(\mathbf{s})$. For instance, one method that provides an upper bound when minimizing is Point-Based Value Iteration (PBVI). Since the approximate solution is guaranteed to exceed the correct value of $U_C(\mathbf{s})$, if the approximate solution is found to be less than α then the exact solution is also guaranteed to be less than α. Next, we compute the future expected reward and future expected risk using a forward search. The minimum risk U_C is used at the end of the search horizon to ensure constraint feasibility beyond the horizon. The forward search is executed by solving the following set of equations recursively.

$$\pi^*(\mathbf{s}) = \arg\max_{\mathbf{a}\in\mathbf{a}_C}[\sum_{\mathbf{s}\in S} R(\mathbf{s},\mathbf{a})b(\mathbf{s}) + \sum_{z\in Z}\sum_{\mathbf{s}\in S} O(z,\mathbf{s})V_R(\mathbf{s})b'(\mathbf{s},\mathbf{a},z)] \tag{2.18}$$

$$V_R(\mathbf{s}) = \max_{\mathbf{a}\in\mathbf{a}_C}[\sum_{\mathbf{s}\in S} R(\mathbf{s},\mathbf{a})b(\mathbf{s}) + \sum_{z\in Z}\sum_{\mathbf{s}\in S} O(z,\mathbf{s})V_R(\mathbf{s})b'(\mathbf{s},\mathbf{a},z)] \tag{2.19}$$

$$\mathbf{a}_C = \{\mathbf{a} : Q_C(\mathbf{s},\mathbf{a})b(\mathbf{s}) < \alpha\} \tag{2.20}$$

$$Q_C(\mathbf{s},\mathbf{a}) = \sum_{\mathbf{s}\in S} C(\mathbf{s},\mathbf{a})b(\mathbf{s}) + \sum_{z\in Z}\sum_{\mathbf{s}\in S} O(z,\mathbf{s})V_C(\mathbf{s})b'(\mathbf{s},\mathbf{a},z) \tag{2.21}$$

$$V_C(\mathbf{s}) = Q_C(\mathbf{s},\pi^*(\mathbf{s})) \tag{2.22}$$

The main differences between Equations 2.8 - 2.12 for MDPs and Equations 2.18 - 2.22 is the fact that we must now account for imperfect state information (in the form of the belief $b(\mathbf{s})$ and the observations Z. Therefore in Equation 2.18, we have to take the expected future reward over all possible future *belief* states b' (instead of the future states $\mathbf{s}'$, as in Equation 2.8). Future belief states depend on not just the transition model but also the observation model, and are computed as shown in

Equation 2.23.

$$
\begin{aligned}
b'(\mathbf{s}', \mathbf{a}, z) &= \frac{O(z|\mathbf{s}') \sum_{\mathbf{s} \in S} T(\mathbf{s}'|\mathbf{s}, a) b(\mathbf{s})}{P(z|b(\mathbf{s}), \mathbf{a})} \\
P(z|b(\mathbf{s}), \mathbf{a}) &= \sum_{\mathbf{s}' \in S} O(z|\mathbf{s}) \sum_{\mathbf{s} \in S} T(\mathbf{s}'|\mathbf{s}, \mathbf{a}) b(\mathbf{s})
\end{aligned} \tag{2.23}
$$

The complete algorithm is presented as the subroutine `Expand` as shown in Algorithm 3. Note that we abbreviate $\sum_{\mathbf{s} \in S} V_R(b(\mathbf{s}))$ as $V_R(b)$, and similarly $\sum_{\mathbf{s} \in S} V_C(b(\mathbf{s}))$ as $V_C(\mathbf{s})$. 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 off-line constraint penalty approximation U_C (lines 3-4). For nodes that are not leaf nodes, the algorithm generates all possible successor nodes $b'(\mathbf{s})$ 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 Equation 2.19, and the constraint according to Equation 2.22. 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.

Algorithm 4 is the main on-line C-POMDP planning algorithm. The inputs to this algorithm are the current belief state b, 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. Once the action is executed and a new observation is received, the current belief is updated [49] (line 8) again using 2.23, but this time for just the actual observation.

Algorithm 3 The expand routine for solving constrained POMDPs

1: **Function** Expand(b, D)
   Static:
   $U_C(b)$
2: **if** $D = 0$ **then**
3: $V_R(b) \leftarrow 0$
4: $V_C(b) \leftarrow U_C(b)$
5: **else**
6: $i \leftarrow 1$
7: $V_R(b) \leftarrow -\infty$
8: $V_C(b) \leftarrow -\infty$
9: **while** $i \leq |A|$ **do**
10: $\tau(b, a_i, z) = b'(\mathbf{s}|b, a_i, z)$ (future belief given b, a_i, z as in Equation 2.23)
11: $Q_R(b, a_i) \leftarrow R(b, a_i) +$
$\gamma \sum_{z \in Z} P(z|b, a_i)\text{Expand}(\tau(b, a_i, z), D - 1)$
12: $Q_C(b, a_i) \leftarrow C(b, a_i) +$
$\gamma \sum_{z \in Z} P(z|b, a_i)\text{Expand}(\tau(b, a_i, z), D - 1)$
13: **if** $Q_R(b, a_i) = \max(V_R(b), Q_R(b, a_i))$ **and** $Q_R(b, a_i) \leq \alpha$ **then**
14: $V_C(b) = Q_C(b, a_i)$
15: $a* \leftarrow a_i$
16: **end if**
17: $R(b) \leftarrow \max(R(b), R(b, a_i))$
18: $i \leftarrow i + 1$
19: **end while**
20: **end if**
21: **return** $a*, R(b), C(b)$

Notice that the constraint is propagated the same as the reward. The reason this can be done is because the constraint in Equation 2.16 is imposed on the belief, i.e. the *belief* must satisfy the constraint α at all times. The constraint would have been much tighter if it had been imposed on the *state* instead. Had the constraint been imposed on the state, the planning algorithm would have to account for the fact that the observations being received might be too optimistic, i.e. might suggest that the system is farther from violating the constraint than it actually is. Such observations would lead the planner to underestimate the risk of taking an action. One way to ensure that a constraint on the state remains satisfied is to use the `max` operator over the observations in Equation 2.21 instead of the expectation operator, thereby

Algorithm 4 Pseudo-code for an on-line algorithm to solve constrained POMDPs

```
1: Function ConstrainedOnlinePOMDPSolver()
   Static:
   b : 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 ← b_0
3: Initialize T to contain only b_0 at the root
4: while ExecutionTerminated() do
5:    Expand(b, D)
6:    Execute best action a for b
7:    Receive observation z
8:    Update b according to Equation 2.23
9:    Update tree T so that b is the new root
10: end while
```

modifying the Equation to be

$$Q_C(\mathbf{s},\mathbf{a}) = \sum_{\mathbf{s}\in S} C(\mathbf{s},\mathbf{a})b(\mathbf{s}) + \max_{z\in Z} \sum_{\mathbf{s}\in S} O(z,\mathbf{s})V_C(\mathbf{s})b'(\mathbf{s},\mathbf{a},z) \qquad (2.24)$$

Such a modification would ensure that the constraint will continue to be satisfied no matter what observation we receive after executing the planned action. Note that this adds additional conservatism, since we are now requiring that *all* future belief states satisfy the constraint, and not just the *expected value* of the future belief states.

2.7 Summary

This chapter proposed a fast on-line algorithm for solving constrained MDPs and POMDPs. The algorithm relies on an off-line approximate solution to estimate the risk beyond the search horizon. It was shown that this algorithm provides constraint-feasible plans while giving higher reward than simply treating constraints as reward penalties. In this chapter, we considered the case where there is a single agent operating in a static environment. In a later section we applied the methods to a team of two agents. However, there are several significant obstacles to fully extending this

work to multi-agent teams and dynamic environments. First, the size of the state space and the action space grow exponentially in the number of agents. An on-line search becomes computationally expensive as the branching factor of the search tree becomes large, in this case $|A|^N$ where N is the number of agents. Second, the above approach assumes that the constraint map of the environment is fixed. But this may not be the case, particularly in a dynamic environment. The ability to recompute $U_C(\mathbf{s})$ quickly becomes important. For both these reasons, it is desirable to have a solution scheme that scales reasonably as the number of agents grows. In the later chapters, we will investigate ways to overcome this scaling problem. But first, we complete the discussion for the single agent case by looking at the case of an agent operating in continuous domains.

Chapter 3

Planning Under Uncertainty and Constraints in Continuous Domains

So far we have dealt exclusively with discrete MDPs and discrete state spaces. Existing literature on constrained MDPs has also focused on discrete MDPs. For example, Dolgov El al. have investigated resource constraint problems, where the resource (e.g. the number of robots) is a discrete quantity [57, 58]. Other work includes offline methods for solving discrete constrained POMDPs [46]. The previous chapter proposed an online algorithm for solving constrained MDPs, but was only applied to discrete problems.

The size of the state space for realistic domains is often large and possibly infinite. This property often limits the applicability of discrete representations in two ways: (1) representing the value function in such domains require large storage capacity, and (2) the computational complexity involved in calculating the solution is not sustainable (e.g. complexity of exact policy evaluation is cubic in the size of the state space). One approach to deal with continuous state spaces is to write the value function as a weighted sum of known continuous functions, known as *basis functions* or *features*. The problem of representing the value function then becomes a matter of finding he weights associated with each basis function - generally a much more tractable

problem.

Function approximators have elevated the applicability of existing MDP solvers to large state spaces [75–78]. The main virtue of function approximators is their ability to generalize learned values among similar states by mapping each state to a low dimensional state space and representing the corresponding value as a function of the mapped point. Linear function approximation, in particular, has been largely used within the community due to its simplicity, low computational complexity [79], and analytical result [80]. The set of bases for linear function approximation can been defined randomly [81], hand picked by the domain expert [82], or learned adaptively [83, 84]. As the main focus of this thesis is to extend constrained MDP solvers to continuous domains, we focus our attention on the hand picked function approximators and leave the extension of our work to adaptive function approximators for future work.

3.1 Proposed Solution

As we have seen the previous chapter, an MDP is defined by the tuple $< S, A, R, T >$ where $\mathbf{s} \in S$ is the state space, $\mathbf{a} \in A$ is the set of actions, $R(\mathbf{s}, \mathbf{a}) : S \times A \rightarrow \Re$ is a *reward model* that maps states and actions to rewards, and $T(\mathbf{s}', \mathbf{a}, \mathbf{s}) : S \times A \times S \rightarrow \Re$ is the *transition model* that gives the probability that state $\mathbf{s}'$ is reached when action $\mathbf{a}$ is taken in state $\mathbf{s}$. The stochastic dynamics of the system are captured in the transition model T, and the mission objectives are encoded in the reward model R. A *solution* to an MDP is a *policy* $\pi : S \rightarrow A$ that maps states to actions. In other words, a policy informs us about which action to take given a state. The *optimal policy* π^* is defined as one that satisfies the optimality condition given in Equation 2.1 and reproduced below:

$$\pi^* = \arg\max_{\pi} E\left[\sum_{t=0}^{T} \gamma^t R(\mathbf{s}_t, \pi(\mathbf{s}_t))\right],$$

where $\gamma \in [0, 1]$ is a discount factor that time-discounts rewards. S can be discrete or continuous, although in this work we focus on continuous state spaces.

Previously, we extended the standard MDP framework by defining a *constrained MDP* as the tuple $< S, A, R, T, C >$, where S, A, R, and T are the same as the MDP formulation, and $C(\mathbf{s}) : S \rightarrow \{0, 1\}$ is a *constraint model* that identifies if being at state $\mathbf{s}$ would violate a constraint. The optimal policy π^*, that solves a constrained MDP was defined as the policy that satisfies Equations 2.2 and 2.3, which are reproduced below:

$$\pi* = \arg\max_{\pi} E\left[\sum_{t=0}^{T} \gamma^t R(\mathbf{s}_t, \pi(\mathbf{s}_t))\right]$$

$$\text{s.t. } E\left[\sum_{t=0}^{T} C(\mathbf{s}_t, \pi(\mathbf{s}_t))\right] \leq \alpha$$

We define risk as the probability that a constraint on the system will be violated. α is therefore the tolerable risk threshold. In order to assure that the risk is bounded by 1.0 for all states, we assume that all violating states are terminal states as well.

The standard MDP framework does not provide a mechanism to explicitly capture risk because there is no natural means by which hard constraints can be incorporated. Typically, hard constraints are treated as penalties in the reward model. As penalties for violating constraints are increased, the resulting policy will be more conservative. By adding $-\infty$ as the penalty for any constraint violation, MDPs can easily capture scenarios where no tolerance is acceptable. However, in realistic domains accomplishing a mission objective often requires some risk tolerance. For example, a rescue robot looking for civilians in a disastrous situation might face fire explosions along the way. If no tolerance is allocated for the mission, the solution might command the robot to stand still for the whole mission horizon. Hence, we assume some degree of risk is acceptable. Unfortunately, we showed in the previous chapter that in many situations the choice of penalty value that achieves the desired balance is not clear, and tuning that penalty value might require more *a priori* knowledge of the optimal policy than can reasonably be expected. Furthermore, there might not even

be a value for which this balance is achieved - it was shown that in some situations a planner that models both rewards and risks in the reward model can switch abruptly between being too conservative or being too risky. Therefore, we proposed a solution that explicitly keeps track of both reward and risk. We now extend this work to continuous domains.

3.1.1 Solving Continuous MDPs: Function Approximation

In a continuous state space, the computation of expected values can be accomplished by taking integrals over expected future distributions of the $\mathbf{s}_t$. However, this can be tedious and difficult to carry out for all but the simplest cases, such as linear systems with Gaussian noise. Since we are interested in solving more general problems, we seek approximation techniques.

Function approximation has been a popular choice for solving large unconstrained continuous MDPs [85, 86]. In our work, we focus on linear function approximation where the value function is written as linear combination of a set of *basis functions* $\phi_i(\mathbf{s})$, which we refer to as *features*. Thus the value function is represented as

$$V(\mathbf{s}) = \sum_{i=0}^{M} \theta_i \phi_i(\mathbf{s}) \tag{3.1}$$

The $\boldsymbol{\theta}_{M\times 1}$ vector is the *weight* vector associated with all features. Since the features are known (they are picked by the designer) the problem of determining $V(\mathbf{s})$ is transformed into the problem of determining the weight vector. If the value function is known at a set of sample states $\mathbf{s}_1, \mathbf{s}_2, \cdots \mathbf{s}_N$, and $N > M$, we can use linear regression to solve for $\boldsymbol{\theta}$:

$$\begin{aligned} \boldsymbol{\theta} &= (\boldsymbol{\Phi}^{\mathbf{T}}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^{\mathbf{T}}\mathbf{V} \\ \mathbf{V} &= [V(\mathbf{s}_1), V(\mathbf{s}_2), \cdots, V(\mathbf{s}_N)] \end{aligned} \tag{3.2}$$

$$\mathbf{\Phi} = \begin{bmatrix} \phi_1(\mathbf{s}_1) & \phi_2(\mathbf{s}_1) & \cdots & \phi_M(\mathbf{s}_1) \\ \phi_1(\mathbf{s}_2) & \phi_2(\mathbf{s}_2) & \cdots & \phi_M(\mathbf{s}_2) \\ \vdots & \vdots & \vdots & \vdots \\ \phi_1(\mathbf{s}_N) & \phi_2(\mathbf{s}_N) & \cdots & \phi_M(\mathbf{s}_N) \end{bmatrix}_{N \times M}$$

The matrix $(\mathbf{\Phi^T \Phi})$ is invertible under the condition that there is at least one feature $\phi_i(\mathbf{s})$ that is non-zero for every state $\mathbf{s} \in S$, and all features are *linearly independent*, i.e the relation $a_i\phi_i(\mathbf{s}) + a_j\phi_j(\mathbf{s}) + a_k\phi_k(\mathbf{s}) = 0$ has no solution (except the trivial solution $a_i = a_j = a_k = 0$) valid for all $\mathbf{s} \in S$, for any i, j, k. The problem, of course, is that $\mathbf{V}$ is unknown. However, we do know that the value function must satisfy the Bellman condition for optimality, given by

$$V(\mathbf{s}) = \max_{\pi(\mathbf{s})} \left[R(\mathbf{s}, \pi(\mathbf{s})) + \sum_{\mathbf{s}'} T(\mathbf{s}', \pi(\mathbf{s}), \mathbf{s}) V(\mathbf{s}') \right] \tag{3.3}$$

We use this condition to iteratively generate the new weight vector. We begin by initializing $\boldsymbol{\theta}$ to some vector. Substituting $\mathbf{V}$ in Equation 3.3 using Equation 3.1 yields the following:

$$V(\mathbf{s}) = \max_{\pi(\mathbf{s})} \left[R(\mathbf{s}, \pi(\mathbf{s})) + \sum_{\mathbf{s}'} T(\mathbf{s}', \pi(\mathbf{s}), \mathbf{s}) \sum_{i=0}^{M} \theta_i \phi_i(\mathbf{s}') \right] \tag{3.4}$$

Thus with an existing weight vector $\boldsymbol{\theta}$, we can compute an updated value function $V(\mathbf{s})$. A new $\boldsymbol{\theta}$ is then obtained by applying Equation 3.2 to the updated $V(\mathbf{s})$. The process is repeated until there is no further improvement in $V(\mathbf{s})$. This process is called *fitted value iteration* [36, 87–89]. In this work, we adapt fitted value iteration and apply them to continuous constrained MDPs. This process requires two major changes. First, we need to maintain *two* value functions, one associated with the reward and the other associated with the risk. It is also desirable, while not essential, to maintain two sets of features as the value function and the risk function might have different complexities each captured best with an individual set of features. Second, the Bellman condition in Equation 3.3 needs to be modified so that the `max` operator is applied only over those policies π that satisfy the bound on the risk.

We call the value function associated with the reward $V_R(\mathbf{s})$ and the value function associated with the risk as $V_C(\mathbf{s})$. The features for describing $V_R(\mathbf{s})$ are $\phi_{Ri}(\mathbf{s})$ whereas the features describing $V_C(\mathbf{s})$ are $\phi_{Ci}(\mathbf{s})$. The value functions can then be written as

$$V_R(\mathbf{s}) = \sum_{i=0}^{M} \theta_{Ri}\phi_{Ri}(\mathbf{s}) \tag{3.5}$$

$$V_C(\mathbf{s}) = \sum_{i=0}^{M} \theta_{Ci}\phi_{Ci}(\mathbf{s}) \tag{3.6}$$

The Bellman condition is modified as follows

$$V_R(\mathbf{s}) = \max_{\pi(\mathbf{s})\in a_C} \left[R(\mathbf{s}, \pi(\mathbf{s})) + \sum_{\mathbf{s}'} T(\mathbf{s}', \pi(\mathbf{s}), \mathbf{s}) V_R(\mathbf{s}') \right] \tag{3.7}$$

$$Q_C(\mathbf{s}, a) = \left[C(\mathbf{s}, a) + \sum_{\mathbf{s}'} T(\mathbf{s}', a, \mathbf{s}) V_C(\mathbf{s}') \right] \tag{3.8}$$

Where a_C is the set of constraint-feasible actions, given by

$$a_C = \{a : Q_C(\mathbf{s}, a) < \alpha\} \tag{3.9}$$

Finally, we obtain $V_C(\mathbf{s})$ as $V_C(\mathbf{s}) = Q_C(\mathbf{s}, \pi(\mathbf{s}))$. The complete algorithm is shown in Algorithm 5.

Note while the risk features ϕ_C and the reward features ϕ_R are independent, a poor choice of ϕ_C will affect the reward performance. For example, if ϕ_C are chosen such that all features have a value of 0 over the constraint-infeasible parts of the state space, then the weights θ_C will always remain 0 and the risk will always be estimated as 0. This can lead to policies that are too risky. Thus the choice of features is an important design criterion. In the next section, we show some results that emphasize this point.

3.2 Results

In this section, we solve a continuous constrained MDP with the function approximation technique presented above. We compare the solution against the performance

Algorithm 5 Fitted Constrained Value Iteration

Input: $S, A, R, T, C, S_0, \mathbf{\Phi}_R, \mathbf{\Phi}_C$
Output: $\boldsymbol{\theta}_R, \boldsymbol{\theta}_C$
Initialize $\theta_R, \theta_C, \Delta \neq 0$
while $\Delta \neq 0$ **do**
  $V_R \leftarrow \mathbf{\Phi}_R \theta_R$
  $V_C \leftarrow \mathbf{\Phi}_C \theta_C$
  for $\mathbf{s} \in S_0$ **do**
    for $a \in A$ **do**
      $Q_C(\mathbf{s}, a) \leftarrow \left[C(\mathbf{s}) + \sum_{\mathbf{s}'} T(\mathbf{s}', a, \mathbf{s}) V_C(\mathbf{s}')\right]$
    end for
    $a_C \leftarrow \{a : Q_C(\mathbf{s}, a) < \alpha\}$
    $V_R(\mathbf{s}) \leftarrow \max_{a \in a_C} \left[R(\mathbf{s}, a) + \sum_{\mathbf{s}'} T(\mathbf{s}', a, \mathbf{s}) V_R(\mathbf{s}')\right]$
    $a^* \leftarrow arg\max_{a \in a_C} \left[R(\mathbf{s}, a) + \sum_{\mathbf{s}'} T(\mathbf{s}', a, \mathbf{s}) V_R(\mathbf{s}')\right]$
    $V_C(\mathbf{s}) \leftarrow Q_C(\mathbf{s}, a^*)$
    $\boldsymbol{\theta}_R{}' \leftarrow (\mathbf{\Phi}_R^T \mathbf{\Phi}_R)^{-1} \mathbf{\Phi}_R^T V_R$
    $\theta_C{}' \leftarrow (\mathbf{\Phi}_C^T \mathbf{\Phi}_C)^{-1} \mathbf{\Phi}_C^T V_C$
    $\Delta \leftarrow \max\{||\theta_C - \theta_C{}'||_2, ||\theta_R - \theta_R{}'||_2\}$
    $\theta_C \leftarrow \theta_C{}'$
    $\theta_R \leftarrow \theta_R{}'$
  end for
end while

of a discrete constrained MDP solver on a discretized version of the problem. The problem studied is a robot navigation problem in a space that has some regions that are dangerous and must be avoided with probability $p > 0.15$ - therefore $\alpha = 0.15$. The definition of risk used is accumulated risk, i.e. the sum of the past and future risk. The goal is to reach a reward region that is located adjacent to a constrained region. Note that under this setup, achieving a reward entails taking some risk, i.e. approaching the danger zones. The problem layout is shown in Figure 3-1. The agent is assumed to be operating in a continuous state space $\mathbf{s} \in S$ and is allowed discrete actions $\mathbf{a}$. The outcome of an action is stochastic and is given by a Gaussian distribution centered at $\mathbf{s} + \mathbf{a}$:

$$P(\mathbf{s}'|\mathbf{s}, \mathbf{a}) = N(\mathbf{s}', \mathbf{s} + \mathbf{a}, \sigma) \tag{3.10}$$

The standard deviation σ determines the amount of noise in the system. However when there are obstacles present, the noise is no longer Gaussian - we account for

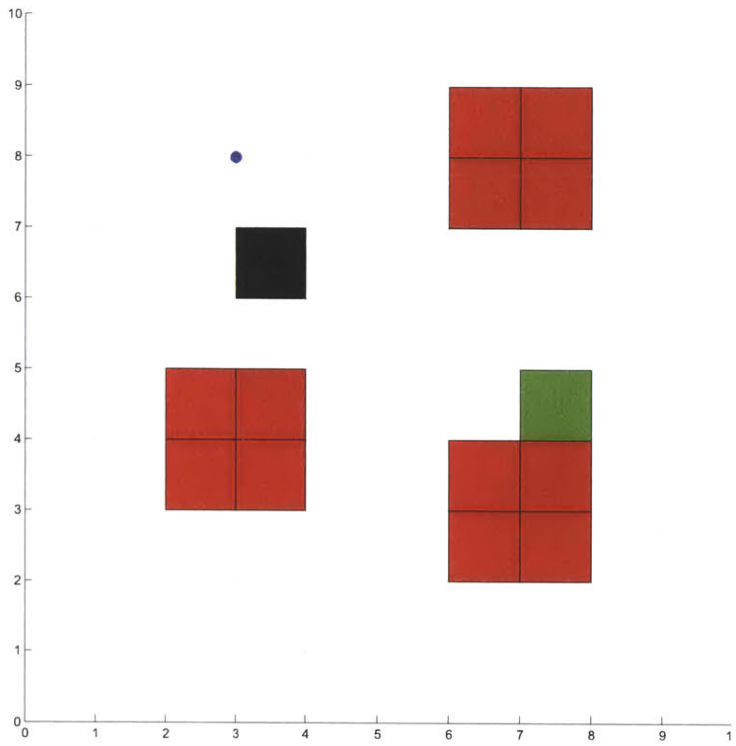

Figure 3-1: Problem setup showing constrained areas (red) and reward area (green).

the fact that passing through the obstacle has zero probability and re-normalize the distribution accordingly. The performance of the proposed algorithm for various values of σ in the presence of obstacles will be investigated in this section. We use *Radial Basis Functions* (RBFs) as features and apply Algorithm 5. A generic radial basis function is defined as a function whose value at a point $\mathbf{x}$ depends only on the distance from some other point $\mathbf{c}$, i.e. $\phi(\mathbf{x}) = \phi(||\mathbf{x}-\mathbf{c}||)$. The norm $||\mathbf{x}-\mathbf{c}||$ a metric that defines a unique distance between two points in the space that contains $\mathbf{x}$ and $\mathbf{c}$. The norm is typically defined as the Euclidean distance, although other distance functions may also be used. The specific radial basis function used in this work is given as

$$\phi(\mathbf{s}_i) = e^{-(\mathbf{s}_i-\mathbf{s_0}).(\mathbf{s}_i-\mathbf{s_0})/\sigma} \tag{3.11}$$

Graphically, Figure 3-2 shows the geometric form of an RBF. An RBF is radially symmetric and has the same form as a Gaussian function. $\mathbf{s_0} \in S$ is a reference

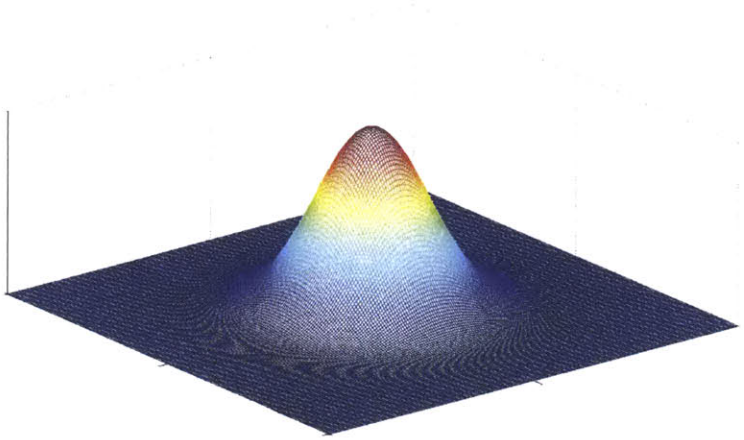

Figure 3-2: A two-dimensional radial basis function

location where the RBF is centered, $\mathbf{s}_i \in S$ is the point at which the RBF is evaluated (the sample states $\mathbf{s}_1, \mathbf{s}_2, \cdots \mathbf{s}_N$) and σ is a parameter that determines the shape of the RBF.

The performance of this algorithm will be compared against a discrete CMDP using a uniform discretization scheme shown as dashed grids in Figure 3-1. The discrete transition model is obtained by integrating the continuous transition model (Equation 3.10) over each grid. A total of 100 cells (10×10) were used in the discretization.

As mentioned above, we solve the problem by using radial basis functions (RBFs) as features. The reward value function V_R is represented by a single RBF centered at the reward region, and the constraint value function V_C is represented by four RBFs at each constrained region (for a total of twelve features). Algorithm 5 is then applied to these features and the resulting value functions are plotted in Figure 3-3. Figure 3-3 shows the approximated reward value function V_R with a representative trajectory. The path towards the high-reward region is not a straight line as one would expect, due to the constrained regions. The high-risk regions (risk greater than 0.15) are highlighted as dark blue. The path taken by the agent curves to avoid the high-risk areas. Although the trajectory shown successfully avoids the constrained regions, this is not a requirement - since we allow a probability of 0.15 for constraint violation, we expect about 15% of the trajectories to be constraint-infeasible.

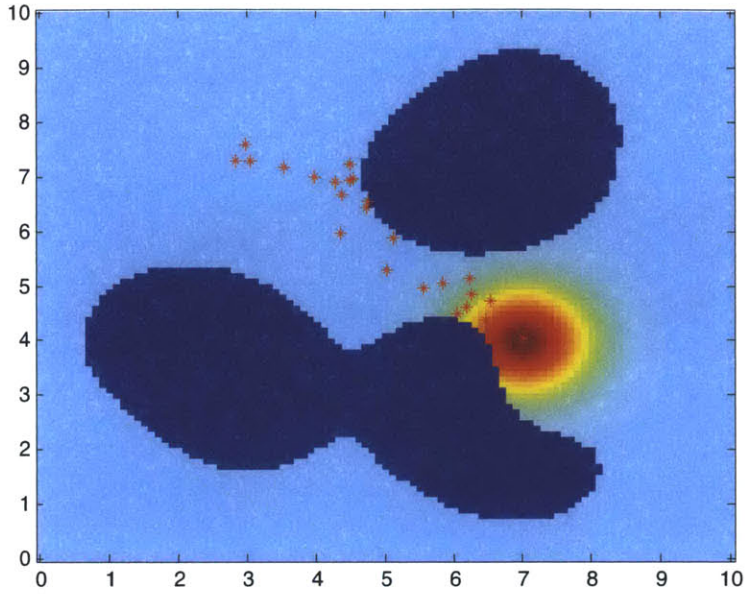

Figure 3-3: A representative trajectory showing the approximated reward value function. Dark areas are regions where under the current policy, risk is greater than 0.15 as measured by the approximated risk value function. The shades represent contours of the reward value function.

This is shown in Figure 3-4. First, we compute V_R and V_C for various values of σ. The resulting policy is then executed 100 times, and the number of trajectories that violate constraints are counted. The RBF function approximation technique stays below the bound of $\alpha < 0.15$ except for high values of σ. At these high noise levels, there are in fact *no* trajectories that can guarantee constraint feasibility with $\alpha < 0.15$. In contrast, uniformly discretizing the state space is too conservative for low σ, and then completely fails to find constraint-feasible policies as σ increases. This is also reflected in the reward shown in Figure 3-5 - the reward obtained by uniform discretization drops drastically as σ increases whereas RBF function approximation performs better, showing only a slow degradation as noise increases.

3.3 Impact of Features

Selection of the right set of features is important to achieve good performance with function approximation. Although there is only one reward in the problem, the use

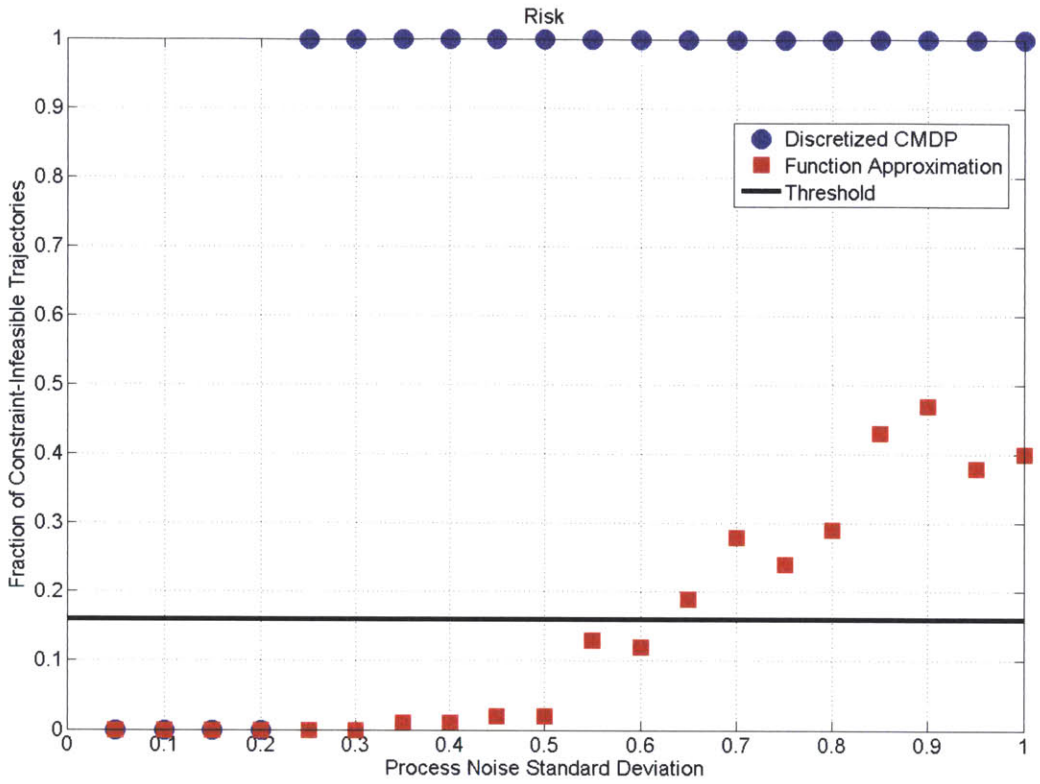

Figure 3-4: The empirically measured risk: RBF vs. Discretization

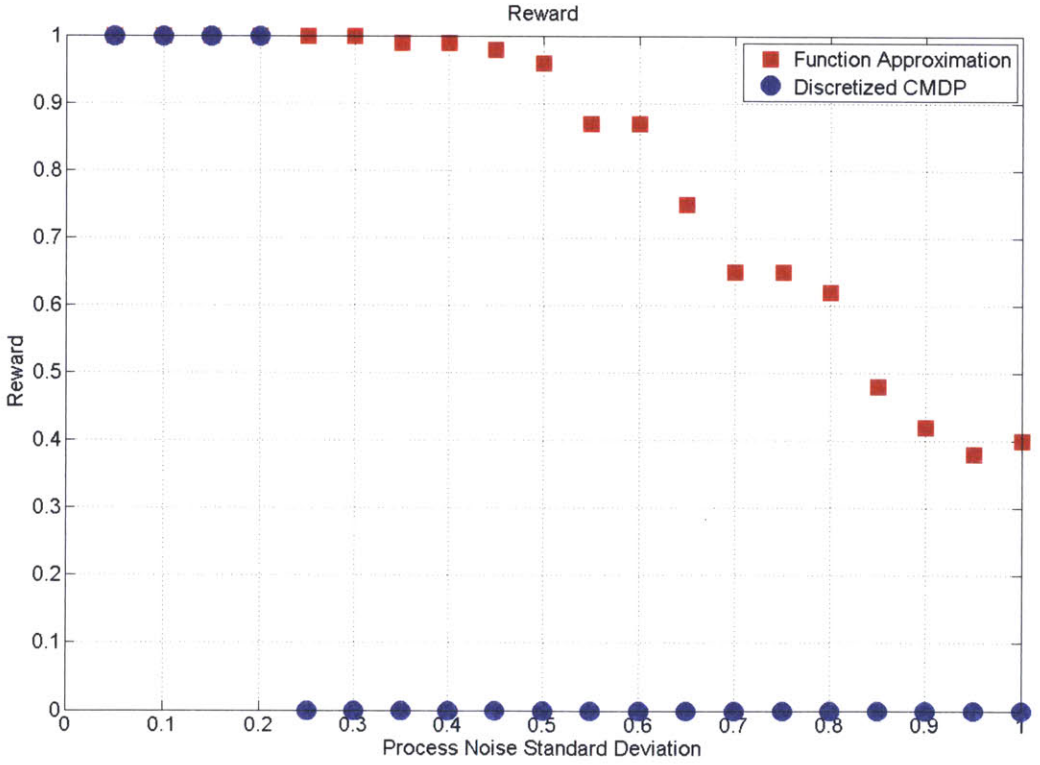

Figure 3-5: The empirically measured reward: RBF vs. Discretization

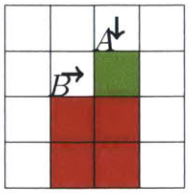

Figure 3-6: Presence of constraints must be accounted for in estimating reward

of only one RBF to represent the reward value function does not capture all the complexity in the problem. This is illustrated with an example shown in Figure 3-6. Suppose two agents A and B are equally distant from a reward and each take an action to reach the reward as shown in Figure 3-6. Also assume that the probability of B entering the constrained region under the action shown is less than α, i.e. the action shown is feasible. The expected reward for agent A can be shown to be close to 1 (it is not exactly 1 since the probability of entering the constrained region, although small, is not zero). However, the expected reward for agent B will be significantly lower than 1, since there is a relatively high probability of the agent entering the constrained region after which the agent cannot perform any further actions (the constrained states are absorbing states). Therefore, despite the risk bound being satisfied for both agents, and despite both agents being the same distance from the reward, the expected reward is not equal. In other words, constrained regions directly affect not just the risk, but the expected reward as well. If the reward value function (which is just the expected future reward) is approximated with a single RBF, this complexity is lost. The expected future reward for both A and B will be the same since an RBF has transverse symmetry and only encodes radial distance information. For small values of σ, this impact will not be very great. For very high values of σ, the action will be identified as infeasible and the underlying policy itself will be changed. But for moderate values of σ, the action will still be identified as feasible, but will affect the expected reward.

The overall effect will therefore be as follows. For low values of σ, the effect of ignoring the constrained regions is insignificant. However, as σ increases but remains small, the expected reward drops. As σ increases even more, the underlying policy

switches to be "safer" and therefore the expected reward (which is not discounted) again increases. As σ continues to increase, the expected reward again drops since the probability of violating a constraint (and hence being unable to gather any more rewards) also increases. This can be accounted for by adding RBFs to the reward value function V_R that are centered at the constraints. These features would capture such behavior and would allow for the direct effect of constraints on the reward to be captured. The results shown in Figure 3-5 use these RBFs associated with the constraints.

3.4 Computational Complexity

The complexity of each while loop iteration in Algorithm 5 is $O(|S_0|^2+M^3+|S_0||A|L)$ where the first term is due to $\mathbf{\Phi}^T\mathbf{\Phi}$, the second term corresponds to matrix inversions, and the last term corresponds to calculating Q_C values. The L parameter specifies the MDP's degree of locality based on sampled points, highlighting the maximum branching factor. Formally $L = \max_{\mathbf{s}\in S_0, a\in A} |\{s'|T(s,a,s') \neq 0\}|$. For most realistic domains $L, |A| \ll |S_0|$, hence the complexity can be approximated in a simpler form $O(|S_0|^2 + M^3)$.

Discretizing the problem is essentially the same as using M features, where M is the number of grid cells. Consequently, $\phi_i()$ maps all states within grid i to 1 and rest of states to 0. Thus using $M = 100$ requires inverting two 100×100 matrices (one for the reward and one for the risk), whereas the proposed function approximation technique requires inverting a 13×13 matrix (for the reward) and a 12×12 matrix (for the risk). By using a relatively small number of RBFs, we decrease the M parameter, making the problem computationally more tractable.

There is yet another computational advantage to using Algorithm 5 with RBFs as features. Since RBFs are defined throughout the entire state space, knowledge gained in one part of the state space, by design, generalizes over the entire state space. Hence, the number of sample points $|S_0|$ does not need to be very high. However with discretization, knowledge gained over one grid is only generalized for

states within that gird. RBFs by potentially requiring less samples to capture the risk and value functions over the whole state space, reduce the quadratic term of the computational complexity.

3.5 Online C-MDP Algorithm vs. Function Approximation

This chapter developed a function approximation method that potentially provides a fast solution to constrained MDPs. However speed comes at the price of accurate representation - an increase in speed requires a more approximate solution (essentially fewer features ϕ_i). As indicated in the previous section, the complexity of each iteration in the algorithm, for small $|A|$ and L, is $O(|S_0|^2 + M^3)$. In contrast, value iteration has a complexity of $O(|S||A|L)$ [35]. Online C-MDP algorithms, on the other hand, scale as $O((L|A|)^D)$ where D is the horizon length [49]. Clearly, by choosing a set of values for L, D and M, the performance of each of these three algorithms can be compared. In these comparisons, only one agent is used and S is set to 3200 (a 10×10 world with 5 reward states) and $|A| = 4$ corresponding to actions in four compass directions. Given the transition model discussed in Chapter 2, $L = 4$. Also $D = 4$ is chosen for the online C-MDP algorithm. For the function approximation algorithm, the number of features M is set to the sum of the reward density and the constraint density, which in this case we allow to vary from 5 to 25. Since the reward density is kept fixed at 5, the maximum number of features used is 30. The number of sample states is set to be $|S_0| = 100$. With this choice of parameters, the three algorithms have comparable computational complexity. The online C-MDP algorithm has complexity of $O((L|A|)^D) = O((16 \times 16)^4) = O(10^4)$, the function approximation algorithm $O(|S_0|^2 + M^3) = O(100^2 + 30^3) = O(10^4)$, and value iteration has complexity $O(|S||A|L) = O(3200 \times 4 \times 4) = O(10^4)$.

In Figure 3-7, we see how the three algorithms perform in terms of the reward. Note that this is the same figure as in Figure 2-11, but with the data for the function

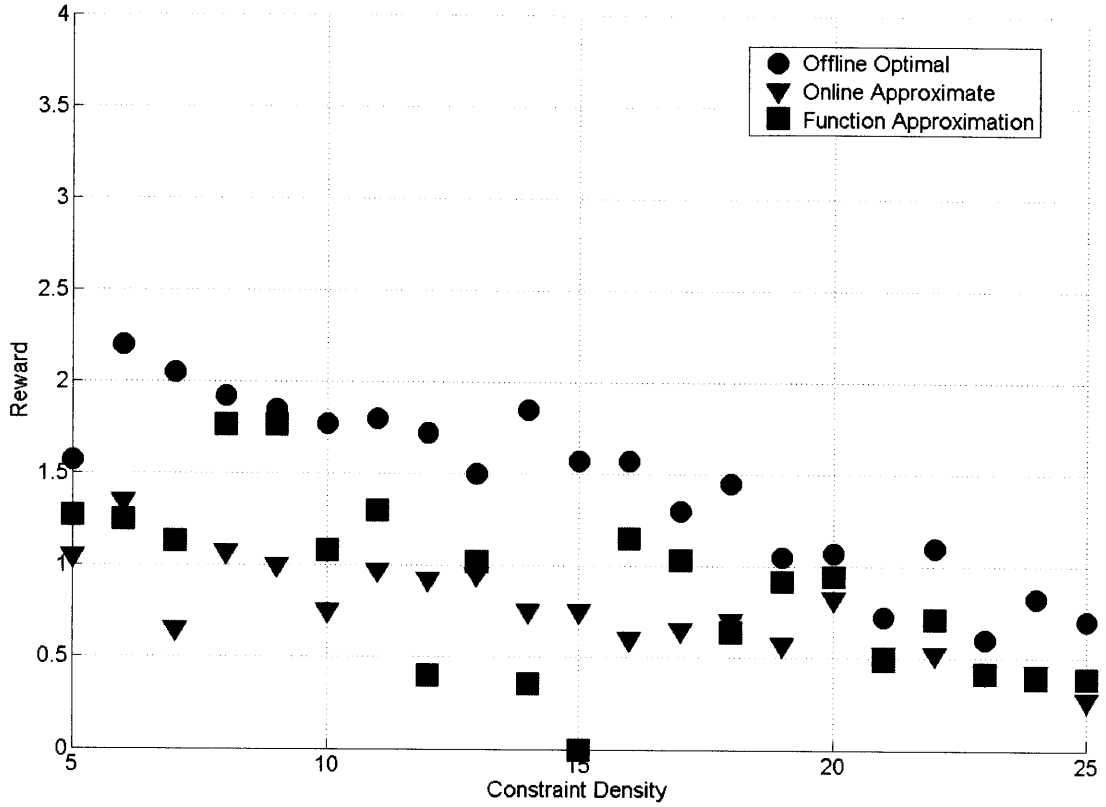

Figure 3-7: Comparison of the online C-MDP algorithm, function approximation and the offline optimal solution

approximation method added. Function approximation with the choice of features used generates policies that achieve high reward, in many cases outperforming the Online C-MDP algorithm. A more dense set of features would come close to matching the performance of the optimal solution, but computation times would correspondingly increase.

Note that although the size of the problem was carefully chosen to make computational complexity comparable, the algorithms scale very differently to larger problem sizes. For instance had the size of the state space $|S|$ been chosen to be bigger (with more reward states, or with a larger grid world) value iteration would become slower. Function approximation will also require more features and more sample states to be effective. However under the condition that the transition model and action space do not change significantly, the online algorithm would be unaffected. For these var-

ious reasons, the Online C-MDP algorithm is considered for extension to multi-agent planning, which is the topic of the next Chapter.

3.6 Summary

Function approximation has been used to solve continuous MDPs quickly and efficiently. By an appropriate choice of basis functions, called *features*, good approximations to the optimal value function can be computed. In this work, we extend function approximation techniques to constrained continuous MDPs. Crucially, in constrained MDPs, there are *two* value functions to be approximated, one for the reward and one for the constraints. We show that the proposed algorithm is computationally more efficient and generates higher-performing policies than uniform discretization. We also show that function approximation is significantly faster in terms of computation time than exact value iteration. It also performs better than the Online C-MDP algorithm from Chapter 2, but at the price of greater computational complexity.

Chapter 4

Planning Under Uncertainty and Constraints for Multiple Agents

In this chapter, we extend the on-line constrained MDP solution technique to teams of multiple agents. A straightforward extension of the work from Chapter 2 to multi-agent systems would involve formulating a large MDP whose state space is the joint state space of individual agents' state spaces, and whose action space would be a joint action space consisting of all the individual agents' actions. The state space of this joint constrained MDP would therefore include the states of all N agents, and would be of size $\prod_{i=1}^{N} |S_i|$, where $|S_i|$ is the size of each individual agent is state space. Note that this expression simplifies to $|S_i|^N$ if all agents have the same state space size $|S_i|$. The action space would be all possible joint actions, and would be of size $\prod_{i=1}^{N} |A_i|$. The reward model R and the constraint model C would be defined over this joint state space and action space, allowing for joint rewards and joint constraints to be incorporated. Theoretically, the on-line constrained MDP solution technique from the previous chapters can be applied to this larger problem without any significant changes.

The drawback of formulating the multi-agent problem as a single constrained MDP is computational complexity. As the number of agents N increases, the size of the state space and the action space grow exponentially. This is undesirable for two reasons. First, the off-line minimum risk computation (designated as U_C in

the previous chapter) becomes difficult as the MDP size increases. This becomes a particularly severe limitation if the constraint map is dynamic, i.e. the constraint model C varies as the mission progresses. For the types of scenarios being considered, this is a real possibility. Second, the speed of on-line reward optimization is directly related to the branching factor of the forward search tree, which in turn is the size of the action space. As the size of the action space grows exponentially, the on-line reward optimization also slows down considerably. Thus a naive application of the proposed solution to multi-agent teams quickly becomes infeasible.

The rest of this thesis deals with the issue of solving constrained MDPs for multi-agent teams. We begin by looking at ways to separate the planning problem for each agent, so that each agent can solve a much simpler planning problem individually and in parallel with other agents. In order to make such a separation, we use the notion of *transition independence*, i.e. the actions of one agent affect only its own dynamics and not those of other agents. Note that although the agents are in fact independent in their transition models (their dynamics) they are in fact coupled at the policy level. This occurs due to reward and constraint coupling. Others, specifically Becker, Zilberstein and Goldman [90] investigate using transition independence to handle reward coupling in the unconstrained case. In this work, we mostly focus on constraint coupling. We also discuss the consequences of the transition independence assumption and situations where the assumption breaks down, and address the issue again in Chapter 6.

4.1 Approach

We approach the problem by first noting that the agents are *transition independent*. Transition independence is defined as follows. If the state vector can be decomposed into N sub-vectors $\mathbf{s} = [\mathbf{s}_1^T, \cdots \mathbf{s}_i^T, \cdots \mathbf{s}_N^T]^T$ and the action vector into sub-vectors $\mathbf{a} = [\mathbf{a}_1^T, \mathbf{a}_2^T, \cdots \mathbf{a}_i^T, \cdots \mathbf{a}_N^T]^T$ such that $p(\mathbf{s}_i'|\mathbf{s}_i, \mathbf{a}_i, \mathbf{a}_k) = p(\mathbf{s}_i'|\mathbf{s}_i, \mathbf{a}_i)$ for all $k \neq i$, the MDP $< S, A, T, R >$ is said to be transition independent [91] [90]. In this case, $\mathbf{s}_i$ is the state of agent i, and $\mathbf{a}_i$ is the action of agent i. Therefore the multi-agent

system under consideration is transition independent due to the fact that the agent dynamics are decoupled. Each agent's individual transition model will henceforth be denoted $T_i(\mathbf{s}'_i, \mathbf{a}_i, \mathbf{s}_i)$. The only coupling between the agents occurs in rewards and constraints. Becker, Zilberstein and Goldman [90] investigate reward coupling in the unconstrained case. Wu and Durfee [91] also solve the problem with reward coupling, but with a linear programming formulation. Yin, Rajan and Tambe address the problem of solving continuous MDPs with transition independence, and again consider reward coupling [92]. This work addresses a different problem, that of transition independence with coupled *constraints*.

The constraint coupling between the agents is represented as follows. We define an *event* e_i as the tuple $< \mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i >$. Agent i is said to have experienced an event e_i when there is a transition from $\mathbf{s}_i \in S_i$ to $\mathbf{s}'_i \in S_i$ under the action $\mathbf{a}_i \in A_i$. Then, we define a *joint event* as the tuple $< e_1, e_2, \cdots e_N, JC >$ where e_i is an event associated with agent i and JC is the *joint constraint penalty* that is awarded to every agent when events $e_1, \cdots e_N$ all occur. Note that this is in addition to the penalty awarded by the constraint model $C_i(\mathbf{s}_i, \mathbf{a})$. We further define $p_{ij}(\pi_i)$ as the probability that event e_i in the joint event j will occur when agent i follows policy π_i. With these definitions, we can write the constrained optimization problem from Eq. 2.2 and Eq. 2.3 as

$$\pi_i^* = \arg\max_{\pi} E\left[\sum_{t=0}^{T} \gamma^t R_i(\mathbf{s}_{it}, \pi_i(\mathbf{s}_{it}))\right] \tag{4.1}$$

$$\text{s.t.} \quad E\left[\sum_{t=0}^{T} C_i(\mathbf{s}_{it}, \pi_i(\mathbf{s}_{it}))\right] + \sum_{j=0}^{E} JC_j \prod_{k=0}^{N} p_{kj}(\pi_k) \leq \alpha \tag{4.2}$$

Knowledge of the other agents' policies are summarized in $p_{kj}(\pi_k)$. The additional term added to the left side of the constraint (Equation 4.2) is the expected *joint constraint penalty*. Since one MDP must be solved for each agent, we have to solve N MDPs of size $S \times A$, rather than one single MDP of size $S^N \times A^N$. However, the policy for any agent depends on $p_{kj}(\pi_k)$ of all the other agents, which in turn depend on their policies. In the next section, we present an iterative scheme for finding the solutions for all agents given this mutual dependency.

The technique to be described below has two components - first, we compute a "library" of policies that give the minimum risk for chosen values of $p_{ij}(\pi_i)$. This library is computed off-line. The agents then communicate with each other and decide which one of the policies from their libraries they will use as the minimum risk-to-go policy. This policy, and its corresponding value function, provide the minimum risk $U_C(\mathbf{s})$ that is then used in the on-line reward optimization to estimate the risk-to-go. However, since this on-line reward optimization modifies the agents' policies π_i, at each step the agents must recompute their new $p_{ij}(\pi_i)$. In general the joint policy iteration must be repeated, but we use a heuristic to avoid doing so. A more detailed description is given below.

4.1.1 Off-line Library Computation and Joint Policy Iteration

First we need to compute the minimum constraint penalty $U_C(\mathbf{s})$. Since we have multiple agents, each with potentially different constraints, each agent i maintains its own individual $U_{Ci}(\mathbf{s}_i)$. By definition, $U_{Ci}(\mathbf{s})$ satisfies

$$U_{Ci}(\mathbf{s}_i) = \min_{\pi_i} E\left[\sum_{t=0}^{T} C_i(\mathbf{s}_{it}, \pi_i(\mathbf{s}_{it}))\right] + \sum_{j=0}^{E} JC_j \prod_{k=0}^{N} p_{kj}(\pi_k) \tag{4.3}$$

In order to compute $U_{Ci}(\mathbf{s}_i)$ as shown above, it is essential to know the $p_{kj}(\pi_k)$, i.e. the probability that the *other* agents k will perform their respective actions in event j. Since these are not known, the above optimization problem is solved *off-line* by *assuming* some values for $p_{kj}(\pi_k)$. The $U_{Ci}(\mathbf{s})$ corresponding to those values of $p_{kj}(\pi_k)$ are stored in a *library* $\mathbf{U_{Ci}}$. Once this library is created, the agent simply needs to look up the appropriate $U_{Ci}(\mathbf{s}_i) \in \mathbf{U_{Ci}}$ when the other agents decide on their actual $p_{kj}(\pi_k)$ values. If the $p_{kj}(\pi_k)$ received do not correspond to any of the sample points, interpolation is used to find the appropriate $U_{Ci}(\mathbf{s}_i)$. It is important to note here that the size of the library is crucial - for example, if a very dense set of sample $p_{kj}(\pi_k)$ are used in computing the library, the library will be large but will also be more complete

than for a smaller sample. In general, there are a maximum of $|S_i||A_i|$ policies per agent. Thus the worst-case number of policies that need to be maintained in the library are $N\prod_{i=0}^{N}|S_i||A_i|$. Given that many of the agents have similar dynamics (e.g. ground vehicles, Dubins vehicles, helicopters), and since many policies may have the same $p_{ij}(\pi_i)$, in practice the total number of policies in the library is less than in the worst-case. It was found during Monte Carlo simulations (whose results are presented in the next section) that the number of policies that need to be maintained to span the entire space of $p_{kj}(\pi_k) \in [0,1]\ \forall j,k$ is finite and typically several orders of magnitude smaller than $N\prod_{i=0}^{N}|S_i||A_i|$. This is particularly useful in situations where the constraint model $C(\mathbf{s},\mathbf{a})$ are not fixed - new U_{Ci} can be computed more quickly and efficiently than if the problem were formulated as a single, centralized MDP.

Once every agent computes its library, each agent picks one policy out of this library and the $p_{ij}(\pi_i)$ (probability of its own event e_i) corresponding to that policy is communicated to all other agents. Since all the other agents perform the same procedure, agent i also receives values for $p_{kj}(\pi_k)$ from all other agents $k \neq i$. Agent i then picks the policy in the library that corresponds to the received set of $p_{kj}(\pi_k)$. This policy yields a new $p_{ij}(\pi_i)$, and the process is repeated until the all agents converge on a policy. This is called the *joint policy iteration* and is shown in Algorithm 6. Once this process is completed (and it is known to converge, see [90]), the agents each have a policy π^*_{ci} that minimizes the risk-to-go. The value function corresponding to this policy, $U_{Ci}(\mathbf{s}_i)$, is then used as the risk-to-go estimate during the on-line search. This process is described in the following section.

4.1.2 Online Search

During the on-line search (Algorithm 7), every agent maintains two quantities - the expected future reward $V_R(\mathbf{s}_i)$ and the expected future risk $V_C(\mathbf{s}_i)$. Both values are initialized to be zero. The Q values for every state $\mathbf{s}_i$ and action $\mathbf{a}_i$ are computed

using the definition of expected reward:

$$Q_R(\mathbf{s}_i, a_i) = R(\mathbf{s}_i, a_i) + \sum_{\mathbf{s}'_i} T(\mathbf{s}'_i, \mathbf{s}_i, a_i) V_R(\mathbf{s}'_i) \quad (4.4)$$

$$Q_C(\mathbf{s}_i, a_i) = C(\mathbf{s}_i, a_i) + \sum_{\mathbf{s}'_i} T(\mathbf{s}'_i, \mathbf{s}_i, a_i) V_C(\mathbf{s}'_i) \quad (4.5)$$

$$\pi_i(\mathbf{s}_i) = \arg\max_{a_i \in a_{Ci}} Q_R(\mathbf{s}_i, a_i) \quad (4.6)$$

$$a_{Ci} = \{a_i : Q_C(\mathbf{s}_i, a_i) < \alpha\}$$

$$V_R(\mathbf{s}_i) = Q_R(\mathbf{s}_i, \pi_i(\mathbf{s}_i)) \quad (4.7)$$

$$V_C(\mathbf{s}_i) = Q_C(\mathbf{s}_i, \pi_i(\mathbf{s}_i)) \quad (4.8)$$

$V_R(\mathbf{s}'_i)$ and $V_C(\mathbf{s}'_i)$ are obtained by calling Equations 4.4–4.8 recursively. At the end of the search horizon (the end of the recursion) we use $V_R(\mathbf{s}'_i) = 0$, $V_C(\mathbf{s}'_i) = U_{Ci}(\mathbf{s}'_i)$ and $\pi_i(\mathbf{s}_i) = \pi_{Ci}(\mathbf{s}_i)$.

In the process of optimizing the reward, the policy $\pi_i(\mathbf{s}_i)$ (Equation 4.6) is modified. In general, this will lead to a change in the value of $p_{ij}(\pi_i)$, which would then require all the agents to repeat the joint policy iteration. Since doing so at each node in the forward expansion can be computationally expensive, we use a heuristic instead. Each time the policy is modified (Equation 4.6), a new p_{ij} is computed for the new policy π'_i. If $p_{ij}(\pi'_i) > p_{ij}(\pi_i)$, the new policy π'_i is discarded – in other words, the agent is not allowed to take more risk than was announced to the rest of the team. Since the joint constraint penalty $\sum_{t=0}^{T} JC_j \prod_{k=0}^{N} p_{kj}(\pi_k)$ is proportional to $p_{kj}(\pi_k)$, the risk-to-go estimate $U_{Ci}(\mathbf{s}_i)$ that was computed previously becomes an overestimate and remains valid (as discussed in Chapter 2). Thus we avoid having to repeat the joint policy iteration, and can ensure that the joint constraint is not violated despite a unilateral change in policy. The disadvantage is the additional conservatism introduced - if an agent finds an opportunity to increase reward by taking more risk that opportunity cannot be exploited. Despite this conservatism, the main computational advantage of the proposed method is significant - we only need to solve $O(N)$ MDPs of size $S \times A$, instead of one MDP of size $S^N \times A^N$. Furthermore, in practice, the proposed method achieves good performance. These results are presented next.

Algorithm 6 Joint policy iteration for computing the minimum constraint-penalty-to-go

Function JointPolicyIteration($\mathbf{U_{Ci}}$)
Initialize $p_{kj} = 1 \forall k$, $p'_{kj} \neq p_{kj}$
while $p'_{kj} \neq p_{kj}$ **do**
  $\pi_{Ci} = \mathbf{U_{Ci}}(p_{kj})$
  $p'_{ij} = \text{ComputeP}(\pi_{Ci})$
  if $p'_{ij} \neq p_{ij}$ **then**
    Broadcast p'_{ij}
    $p_{ij} = p'_{ij}$
  end if
  Receive p'_{kj}
end while
return π_{Ci}

Algorithm 7 Decentralized Constrained MDP Algorithm

Function Expand($\mathbf{s}_i$,U_{Ci},D)
if $D = 0$ **then**
  $V_C(\mathbf{s}_i) = U_{Ci}(\mathbf{s}_i)$; $V_R(\mathbf{s}_i) = 0$; $\pi_i(\mathbf{s}_i) = \pi_{Ci}\mathbf{s}_i)$
  return $\pi_i(\mathbf{s}_i)$, $V_R(\mathbf{s}_i)$,$V_C(\mathbf{s}_i)$
else
  for $a_i \in A_i$ **do**
    $[V_R(\mathbf{s'}_i), V_C(\mathbf{s'}_i), \pi_i(\mathbf{s'}_i)] = \text{Expand}(\mathbf{s}'_i, U_{Ci}, D-1)$
    $Q_R(\mathbf{s}_i, \mathbf{a}_i) = \sum_{\mathbf{s}'_i} V_R(\mathbf{s'}_i) P(\mathbf{s}'_i, \mathbf{s}_i, \mathbf{a}_i) + R(\mathbf{s}_i, \mathbf{a}_i)$
    $Q_C(\mathbf{s}_i, \mathbf{a}_i) = \sum_{\mathbf{s}'_i} V_C(\mathbf{s'}_i) P(\mathbf{s'}_i, \mathbf{s}_i, \mathbf{a}_i) + C(\mathbf{s}_i, \mathbf{a}_i)$
    $\pi'_i(\mathbf{t}_i) = \pi_i(\mathbf{s}_i) \quad \forall \mathbf{t}_i \neq \mathbf{s}_i$
    $\pi'_i(\mathbf{s}_i) = \mathbf{a}_i$
    $p_{ij}(\mathbf{a}_i) = \text{ComputeP}(\pi'_i) \quad \forall j$
  end for
  $a_{Ci} = [\mathbf{a}_i : Q_C(\mathbf{s}_i, \mathbf{a}_i) < \alpha \quad \textbf{AND} \quad p_{ij}(\mathbf{a}_i) < p_{ij}]$
  $\pi'_i(\mathbf{s}_i) = \arg\max_{\mathbf{a}_i \in \mathbf{a}_{Ci}} Q_R(\mathbf{s}_i, \mathbf{a}_i)$
  $V_R(\mathbf{s}_i) = Q_R(\mathbf{s}_i, \pi'_i(\mathbf{s}_i))$
  $V_C(\mathbf{s}_i) = Q_C(\mathbf{s}_i, \pi'_i(\mathbf{s}_i))$
  return $\pi'_i(\mathbf{s}_i)$, $V_R(\mathbf{s}_i)$,$V_C(\mathbf{s}_i)$
end if

4.2 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. We deal with the fully-observable MDP case. The environment is shown in Figure 4-1. It is characterized by an inner "Courtyard" area (shown by the dotted outline). The Courtyard has several danger zones (shown in red) that contain potential threats, but these danger zones also contain victims that must be approached, assessed and rescued. 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 2-4. The victims are treated as rewards and 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 4-1. In general $R_A \neq R_B$ since there may be a preference as to which agent visits which victim. For instance, Agent A can be viewed as a human first responder and Agent B an autonomous vehicle, in which case $R_A > R_B$.

Agents A and B are not allowed to *both* be in the danger zone with a probability of more than 0.1, i.e the constraint is that only one agent is allowed into the danger zone, and the risk allowed (the maximum constraint violation probability) is $\alpha \leq 0.1$. Also, the rewards R_A and R_B become 0 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 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 nominal paths are shown in Figure 4-2. The highest reward is obtained by having Agents A and B both enter the danger zone to acquire reward, with Agent A gathering more rewards than Agent B since $R_A > R_B$. However, it can be easily seen that this nominal path violates the joint constraint with $\alpha > 0.1$, since it requires both Agents to enter the danger zone with probability 1.

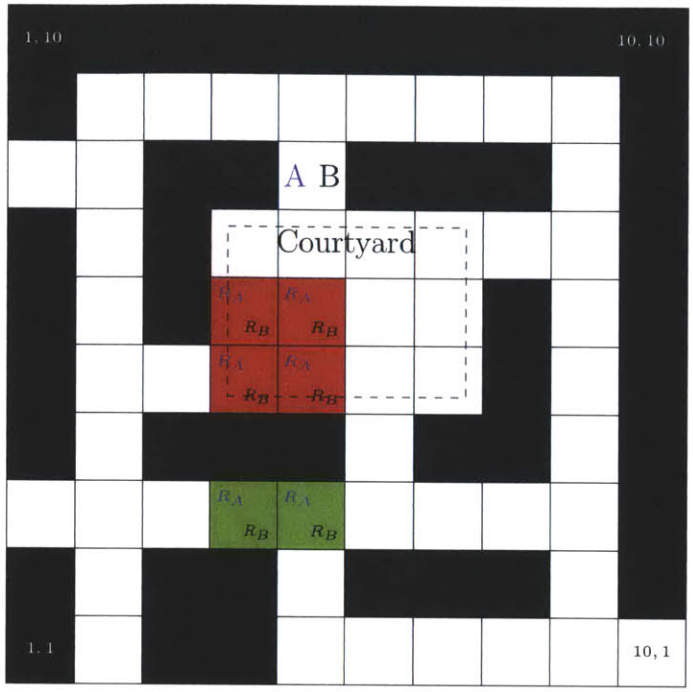

Figure 4-1: The MDP problem set up, showing the inner "Courtyard" (dashed line) which contains danger zones (red). Rewards R_A and R_B exist both within the danger zone and outside (shown in green)

Therefore the highest-reward nominal paths are 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 4-3.

The highest-reward, constraint feasible paths for the agents are for Agent B to gather all the rewards located inside the danger zone, and for Agent A to travel through the Courtyard area (but never entering the danger zones) to reach the rewards on the other side of the courtyard. The risk associated with these paths is non-zero, since there is a finite probability that Agent B, in the course of travelling through the Courtyard, will accidentally enter the danger zone. In the next section, we will show that modelling 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 on-line algorithm with an off-line constraint-feasible approximate

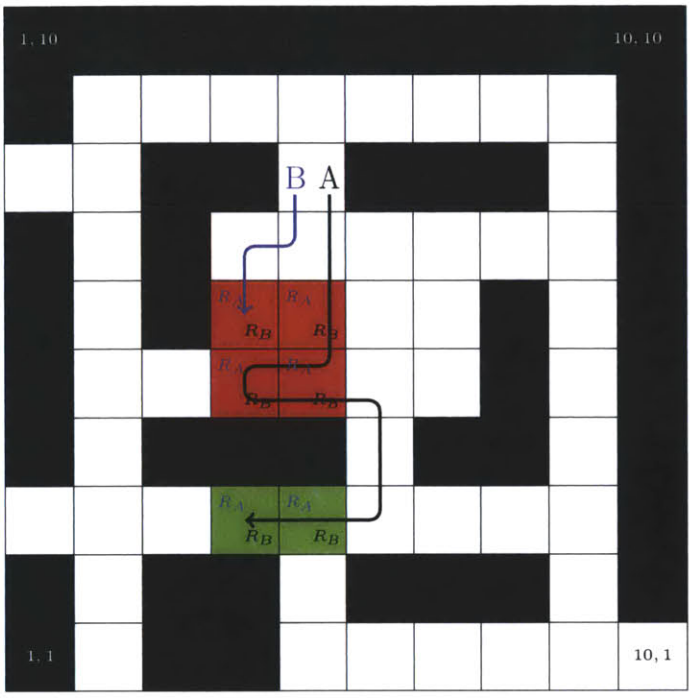

Figure 4-2: The highest-reward nominal paths through the rewards are constraint-infeasible

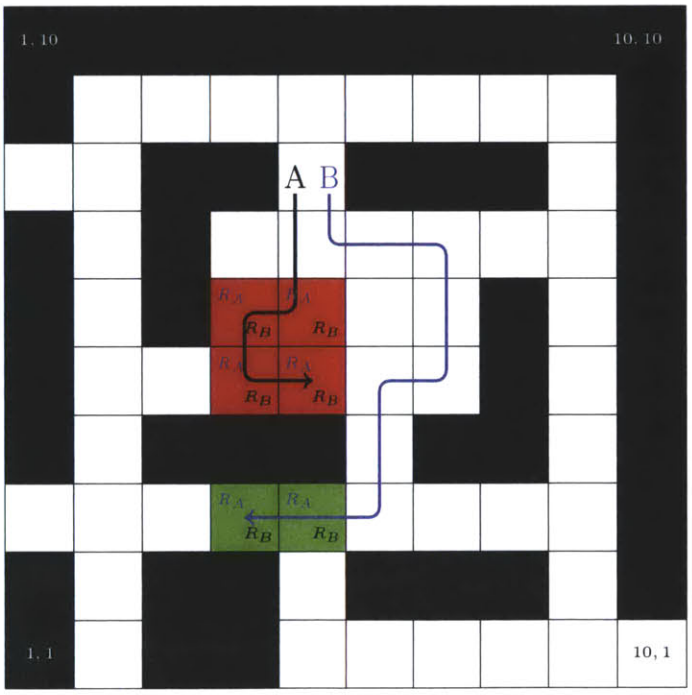

Figure 4-3: Constraint-feasible nominal paths

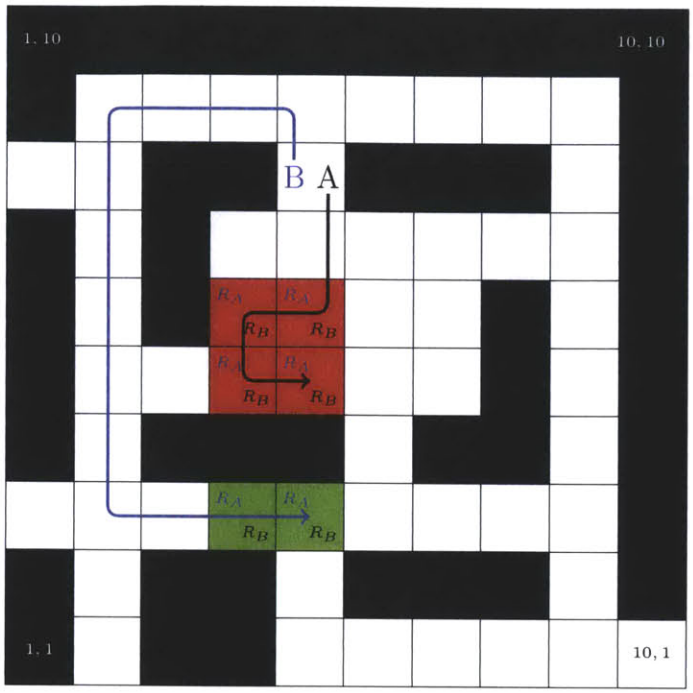

Figure 4-4: Nominal paths computed by value iteration when the constraint is a high negative reward

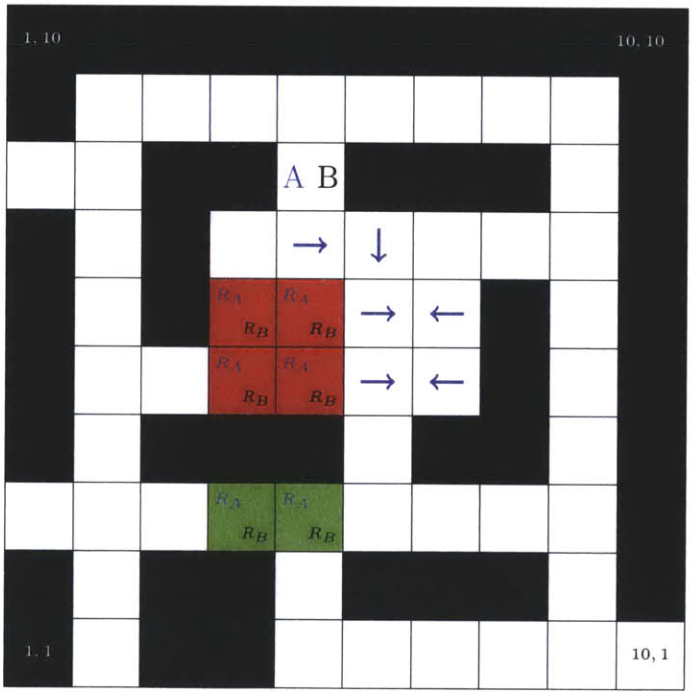

Figure 4-5: Policy (arrows) computed by MDP value iteration when the constraint penalty is lowered - note that under this policy, Agent B's nominal path never leaves the Courtyard area

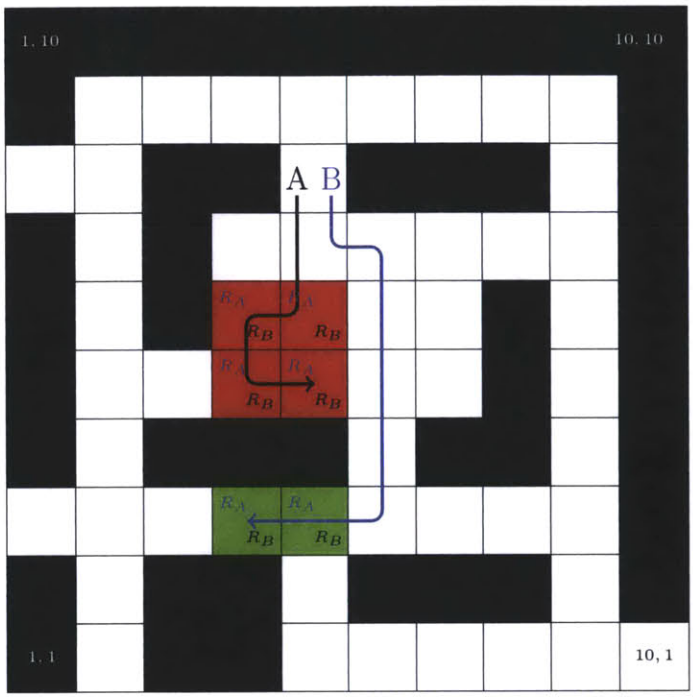

Figure 4-6: Policy (arrows) computed by value iteration when the constraint is a low negative reward

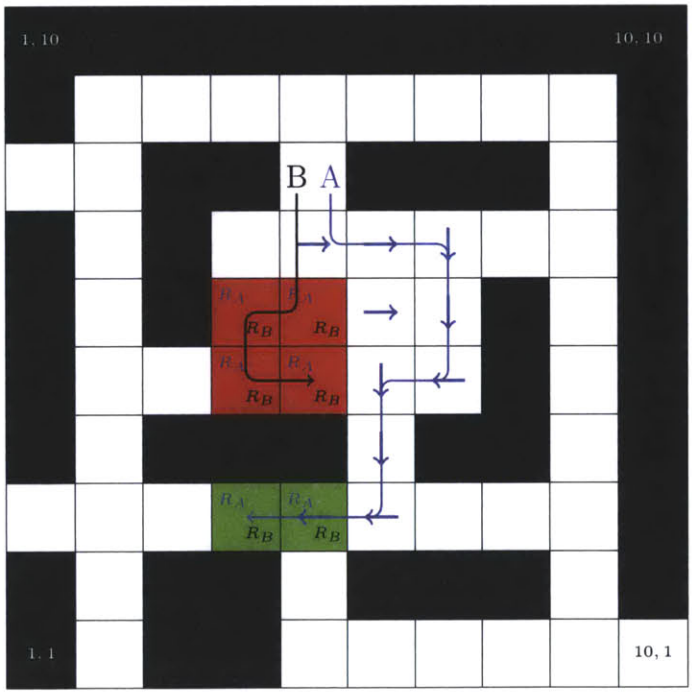

Figure 4-7: Policy (arrows) and nominal paths (lines) computed by the online constrained solver

solution achieves high performance in problems such as this example that require operating close to constraint boundaries.

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 4-4. First thing to note is that the planner does not send 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, by not having a mechanism by which to measure risk, fails to see that the risk associated with moving through the courtyard is an acceptable amount of risk, and acts too conservatively.

Figure 4-5 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 4-6. 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.

Constrained On-line Algorithm Solution

The on-line algorithm presented in this work generates the policy shown in Figure 4-7. The on-line 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 off-line 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 on-line 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 off-line approximate solution provides a conservative, constraint-feasible policy while the on-line algorithm adjusts this policy for better performance. The off-line exact value iteration algorithm also generates the same policy, but requires much more computational time (approximately 10 minutes, compared to approximately 5 seconds for the online algorithm).

4.3 Results

This section presents simulation results that compare the performance of the decentralized Constrained MDP algorithm presented here with that of a standard MDP and an off-line centralized Constrained MDP solver. The problem layout is characterized by three quantities - the *constraint density*, the *reward density* and the *obstacle density*. The constraint density is the fraction of states that are constrained states, the reward density is the fraction of states that are reward states, and the obstacle density is the fraction of states that are blocked by obstacles. The goal is to maximize the rewards gathered while keeping the risk at $\alpha < 0.15$. The transition model is shown in Figure 4-8 – the probability of executing the intended action is 0.9. In the first set of results, we compare an off-line, centralized constrained MDP solver

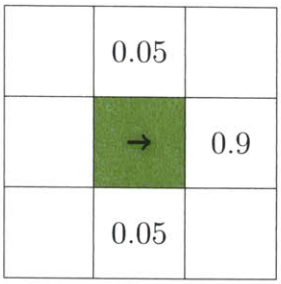

Figure 4-8: Vehicle dynamic model

with the decentralized, on-line solver developed in the previous sections. We use a team of five agents in these results. In the second set of results, we compare the decentralized on-line solver with a standard MDP formulation and show that even when we use only future risk, the decentralized on-line solver generates policies that are significantly less risky than those generated by the standard MDP, particularly in highly constrained cases. To enable a comparison with centralized MDP solutions, the problem size is restricted to two agents.

4.3.1 Off-line Centralized vs. On-line Decentralized

As in Chapter 2, we investigated the difference in performance, as measured by the reward, between the off-line, centralized constrained MDP planner (which gives the optimal solution) and the on-line, decentralized approximate planner. Since the centralized planner gives the optimal policy, we expect that its performance will be better than that of the decentralized planner. As discussed above, the decentralized planner makes some conservative assumptions and uses a finite-horizon forward search, and as a result does not give the optimal policy. This is measured quantitatively in the robot navigation problem described in the previous paragraph. For each value of the constraint density, 40 problems each with randomly-generated constraint and reward locations were created and both the centralized and decentralized planners were applied. The results are shown in Figure 4-9 and confirm our expectations.

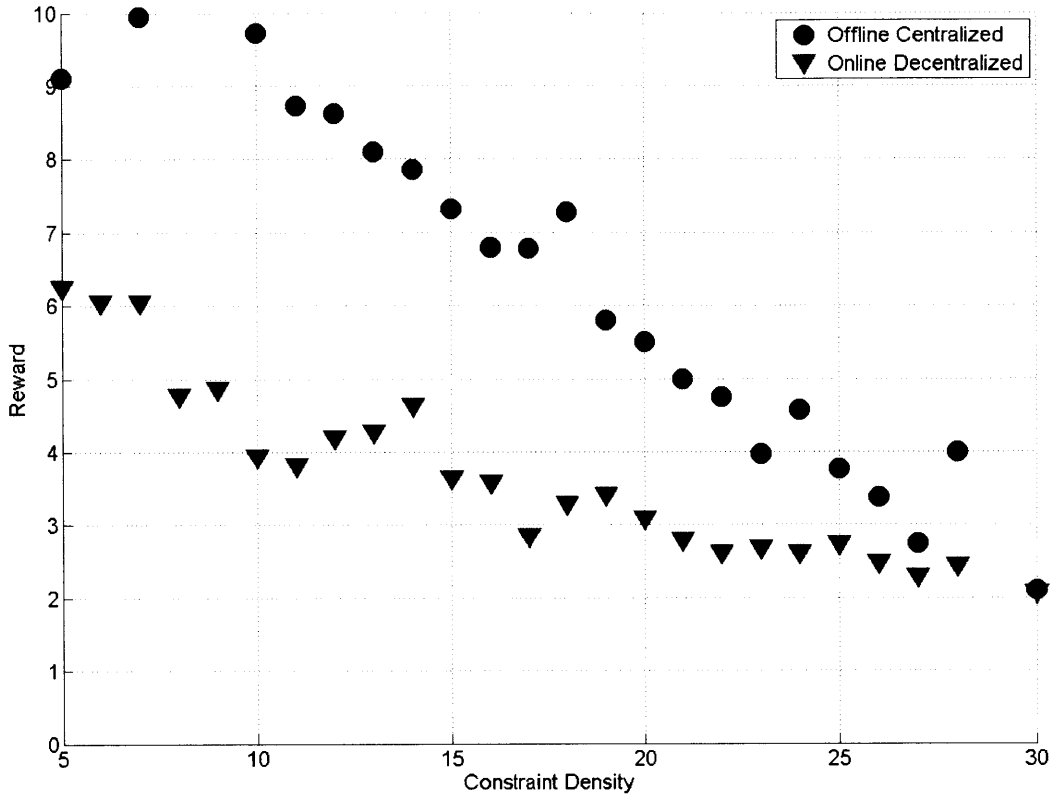

Figure 4-9: A comparison of the off-line centralized solver and the on-line decentralized solver for five agents

4.3.2 Comparison of Risk

We also investigate the difference between the two definitions of risk, memoryless and accumulated. We expect that the reward for accumulated risk will be lower because the agents' choice of actions will become more and more restricted as risky actions are taken during the course of the mission. Figure 4-10 confirms these expectations. Note that the reward for the memoryless planner is substantially higher than that of the accumulated risk planner, particularly for constraint densities greater than 15. This is qualitatively different than in the single agent case (see Figures 2-11 and 2-12), where the two forms of risk did not differ as much, particularly when compared to the centralized optimal planner. Therefore in the multi-agent case, the choice of risk has a bigger impact on the team performance. This can be explained by the

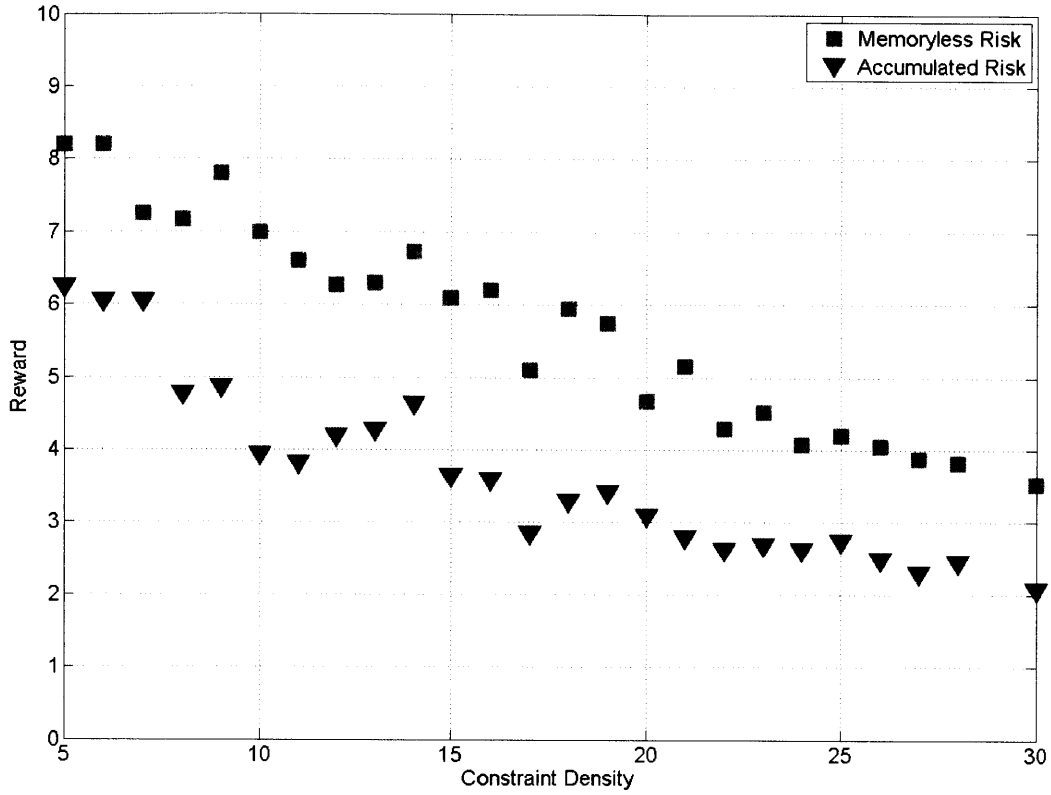

Figure 4-10: A comparison of accumulated risk and memoryless risk for five agents

fact that multiple agents operating in a space of the same size (both the single and multi-agent cases used a 10×10 grid) will encounter more constrained states and therefore have to take more risk than a single agent. Thus using accumulated risk, in the multi-agent case, will constrain the agents much more severely and sooner in the mission than in the single agent case. In fact, if the constraints are placed close to the reward (instead of being placed at random locations) the decentralized planner with memoryless risk might even get a *higher* reward than the optimal centralized planner with accumulated risk. Such a case is presented in the next set of results.

4.3.3 Performance in High Risk, High Reward Environments

The purpose of this section is two-fold - first to show that a standard MDP planner generates policies that are far too risky for the amount of reward they provide, and

second to show that in these high-risk, high-reward environments the choice of risk makes a much bigger difference in performance than in the randomly-generated environments we have investigated thus far. In the following set of results, we apply a standard MDP planner (with constraints treated as reward penalties), the off-line centralized planner with accumulated risk, and the on-line decentralized planner with memoryless risk to problems where high reward necessarily entails operating close to the constraint boundary, i.e taking significant risk. The problems are similar to the robot navigation problems that have been presented thus far, with the exception that the constrained states are placed adjacent to rewards. Thus even low constraint densities require the agents to take substantial risk to achieve high rewards.

The first solution technique we use is *MDP Value Iteration* with constraints treated as penalties. The shortcoming of this method is that as the penalty is increased, the MDP solution becomes more conservative, taking less risk at the cost of a lower reward, whereas when the penalty is lowered, the solution becomes too risky. Furthermore, this trade-off is determined by the constraint density – a low constraint density would have very little effect on the MDP solution, whereas a high constraint density would have a significant impact. The second solution technique is one that explicitly accounts for risk - centralized, *Constrained MDP value iteration.* This method solves the following optimization problem:

$$\begin{aligned}
\pi^*(\mathbf{s}) &= \arg\max_{\mathbf{a}\in\mathbf{a}_C} \left[R(\mathbf{s},\mathbf{a}) + \sum_{\mathbf{s}'} T(\mathbf{s}',\mathbf{s},\mathbf{a}) V_R(\mathbf{s}') \right] \\
V_R(\mathbf{s}) &= \max_{\mathbf{a}\in\mathbf{a}_C} \left[R(\mathbf{s},\mathbf{a}) + \sum_{\mathbf{s}'} T(\mathbf{s}',\mathbf{s},\mathbf{a}) V_R(\mathbf{s}') \right] \\
\mathbf{a}_C &= \{\mathbf{a} : Q_C(\mathbf{s},\mathbf{a}) < \alpha\} \\
Q_C(\mathbf{s},\mathbf{a}) &= C(\mathbf{s},\mathbf{a}) + \sum_{\mathbf{s}'} T(\mathbf{s}',\mathbf{s},\mathbf{a}) V_C(\mathbf{s}') \\
V_C(\mathbf{s}) &= Q_C(\mathbf{s},\pi^*(\mathbf{s}))
\end{aligned}$$

The main shortcoming of this method is that the plan is made off-line and does not account for the fact that in most cases, risk is not realized. This method effectively uses *accumulated risk*, making the planner more conservative than the *Decentralized*

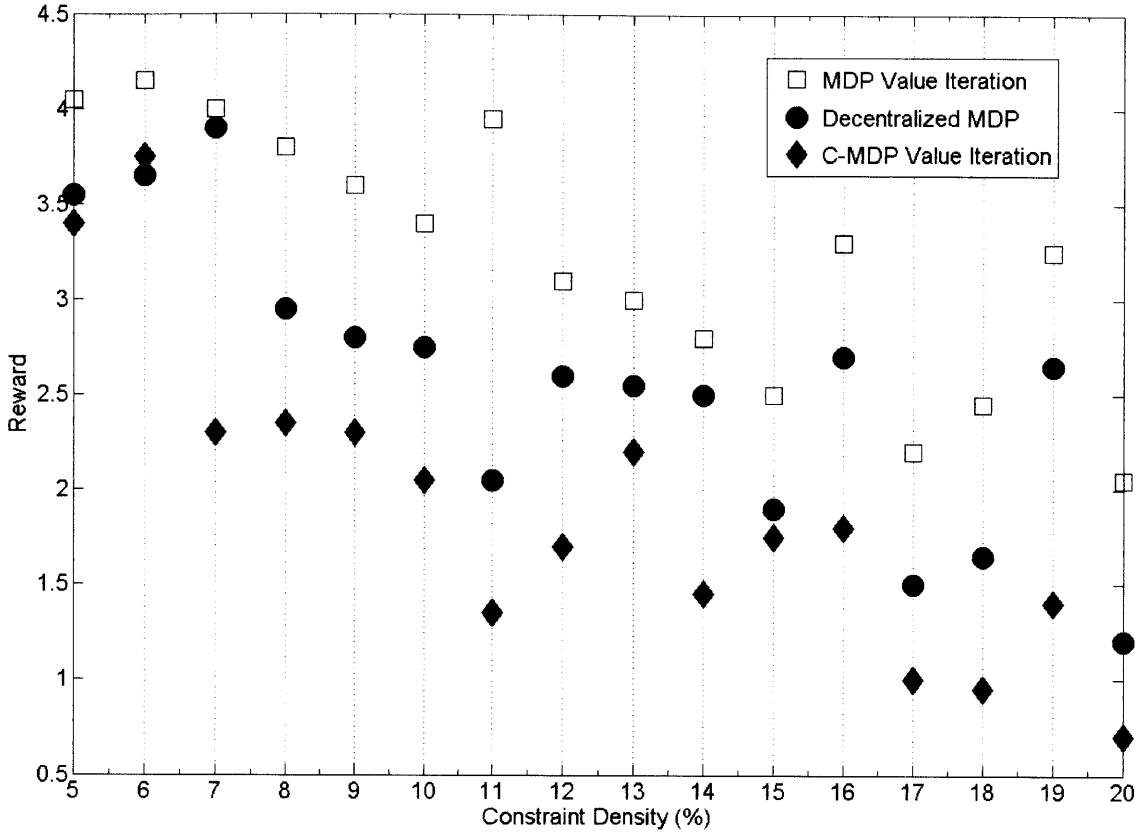

Figure 4-11: Average reward for three planning methods

MDP solution, which uses memoryless risk and is the third solution technique in the comparison.

In order to compare these three methods, we generated problem layouts as described above for varying constraint densities while keeping the reward and obstacle densities fixed. The constraint density is relatively high compared to the reward density - in this case, we use a reward density of 0.05 whereas the constraint density is varied from 0.05 to 0.2. The relatively high constraint density ensures that the risk in the environment is non-trivial, as would be expected in a realistic CBRNE operation [1] [2]. For each value of the constraint density, 40 such random problems were generated. The plans computed by each of the three methods were executed and average reward for each method computed. The fraction of runs in which a constraint was violated was also computed, and this value was compared against the specified risk of

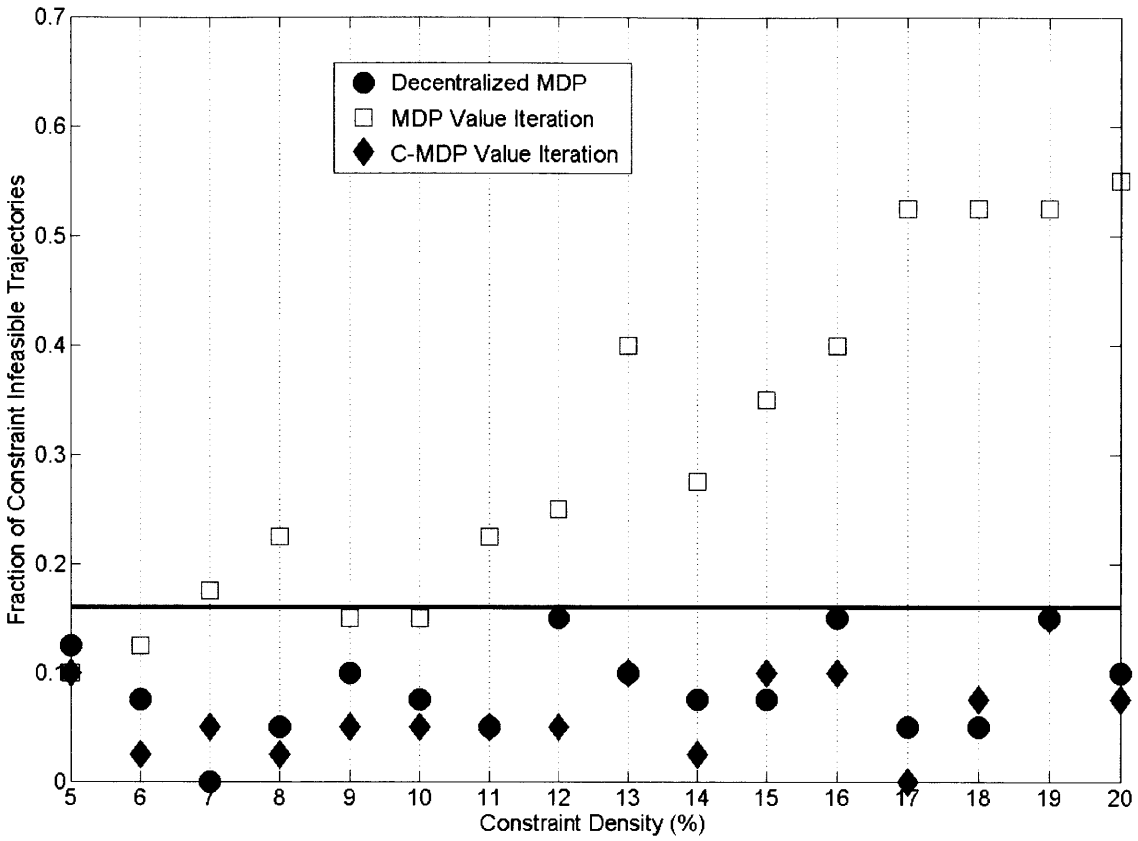

Figure 4-12: Empirically observed risk for three planning methods

$\alpha < 0.15$. The results for the average reward and the risk are shown in Figures 4-11 and 4-12 respectively. First, we note that the reward for all three techniques decreases as the constraint density increases. This is because rewards become less accessible as the number of constrained states increases. In Figure 4-12, we see that unless risk is explicitly accounted for and kept bounded (which the standard MDP does not do), the risk correspondingly increases. Figures 4-11 and 4-12 show that the reward is highest for the standard MDP planner but the risk associated is extremely high. The decentralized planner achieves approximately 25% lower reward than standard MDP planner for any given value of the constraint density, but the risk remains bounded below the threshold of $\alpha \leq 0.15$. The off-line centralized planner also achieves the risk levels specified, but the reward is much lower than the decentralized planner due to the fact that risk taken in the past, even risk that was not realized, is still accounted for in planning. The results show that the decentralized planner obtains good reward

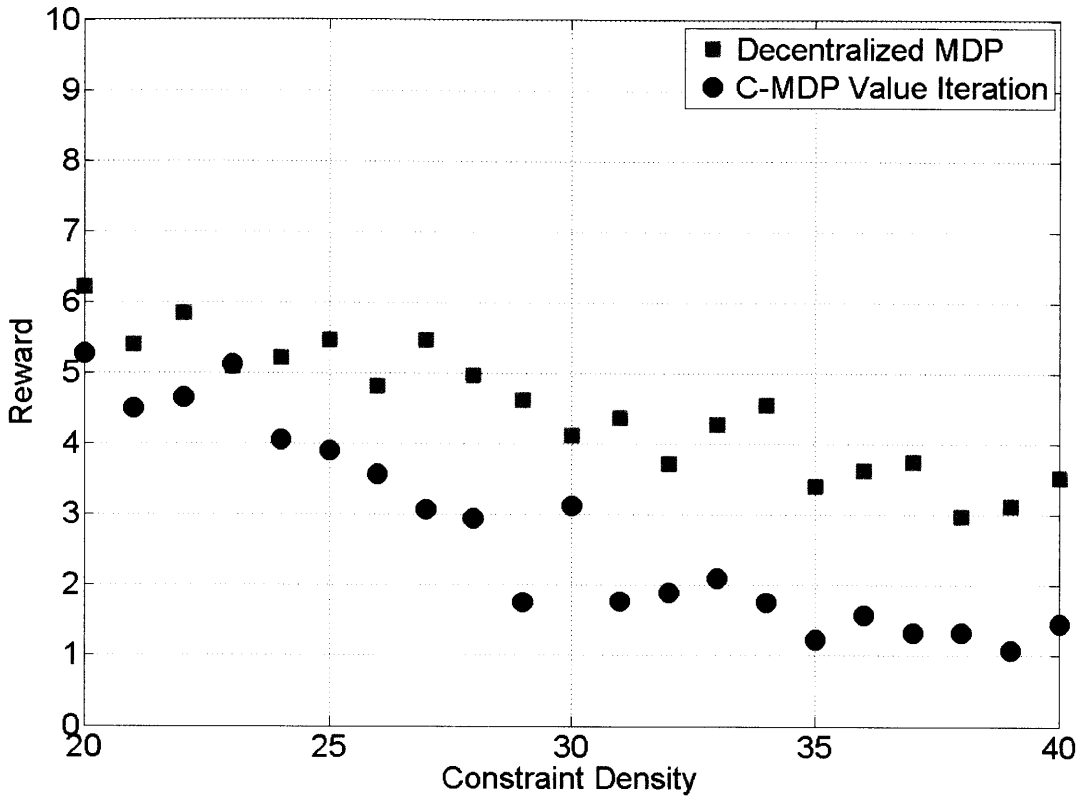

Figure 4-13: Average reward for a team of five agents

while keeping the risk bounded. Finally, in Figure 4-13, we show that similar trends hold for larger teams, in this case five agents. The standard MDP planner is not shown due to the computational difficulty of solving an MDP for that size team.

4.4 Summary

In this chapter, the on-line constrained MDP solver that was proposed previously was extended to multi-agent teams. The exponential growth in the size of the MDP problem with increasing team size was handled by assuming *transition independence*, i.e. the actions of one agent affect only its own dynamics and not those of other agents. Under this assumption, the problem can be decoupled into N separate and significantly smaller MDPs. To handle reward and constraint coupling, we make some assumptions about the actions of the other agents. We specifically look at constraint

coupling, and assume that the agents all observe a pre-allocated limit on the amount of risk they take. This simplifying assumption is called the *Max Risk heuristic.* This heuristic introduces some conservatism, but was still shown to give good performance in simulated problems. In the next chapter we again look at the problem of planning for multi-agent teams, but remove the conservative Max Risk heuristic. In order to preserve constraint feasibility, we instead introduce a consensus-based risk negotiation strategy.

Chapter 5

Distributed Planning Under Uncertainty and Constraints

The work presented previously for planning for multi-agent teams involved using the transition independence of the agents to break a single, large joint planning problem into several smaller individual planning problems. The computational benefit of doing is clear, since the solution technique scales easily to large teams. However, in decentralizing the planner, some assumptions about the actions of the other agents had to be made to ensure that overall team safety is not compromised. Specifically, agents were not allowed to take more risk than was decided by team consensus at the beginning of policy execution. This restriction introduces some conservatism to the planner. In this chapter, we investigate a method for overcoming this conservatism. This is achieved by allowing the agents to take greater risk under the condition that the rest of the team is informed and consents to the agent's proposed increase in risk. The mechanism by which such consensus is obtained is presented in this chapter. We begin by reviewing other work that has looked into the problem of distributing risk among agents in a team. Then a consensus-based planner that extends the work from the previous chapter is presented. This planner allows agents to renegotiate the risk distribution during the execution of the mission. And finally, the results of executing this algorithm in a risky environment are presented.

5.1 Literature Review

The problem of planning in the presence of risk and uncertainty has been investigated in the past. Ono and Williams [68], and Blackmore and Williams [93–99] have looked at the problem of avoiding obstacles with a certain guaranteed probability. The obstacles are treated as constraints, and the probability of colliding with an obstacle is treated as the amount of "risk" the agent is allowed to take. The agent has uncertain dynamics and the underlying planner used is a Mixed-Integer Linear Programming (MILP) solver. Ono and Williams in particular investigate the problem of planning for multi-agent teams with joint constraints. Specifically, the problem they seek to solve is the following.

$$\min_{\mathbf{U}^{1:I}} \sum_{i=1}^{I} J^i(\mathbf{U}^i) \tag{5.1}$$

$$\text{s.t.} \quad \mathbf{x}_{t+1}^i = \mathbf{A}^i \mathbf{x}_t^i + \mathbf{B}^i \mathbf{u}_t^i + \mathbf{w}_t^i \tag{5.2}$$

$$\mathbf{u}_{min}^i \leq \mathbf{u}_t^i \leq \mathbf{u}_{max}^i \tag{5.3}$$

$$P\left[\cap_{i=1}^{I} \cap_{n=1}^{N_i} h_n^{iT} \mathbf{X}^i \leq g_n^i\right] \geq 1 - S \tag{5.4}$$

Where $\mathbf{x}_t^i$ is the state of agent i at time t, $\mathbf{u}_t^i$ is the action (or control) applied by agent i at time t, J^i is the reward function of agent i, $\mathbf{A}^i$ and $\mathbf{B}^i$ collectively define the dynamics of agent i, $\mathbf{u}_{min}^i$ and $\mathbf{u}_{max}^i$ define the range of possible actions (or controls) and h_n^i and g_n^i define the constraints on agent i, with $n \in (1, N_i)$ where N_i is the total number of constraints on agent i. $\mathbf{w}_t^i$ is zero-mean Gaussian noise that adds uncertainty into the dynamics of the agent. S is the probability with which each agent is required to satisfy its constraints. In other words, Ono and Williams require *all* agents to satisfy joint constraints in order for the entire multi-agent team to be constraint-feasible. A violation of a joint constraint by any agent is taken as a system failure. In the formulation used in this work, an agent i can violate a constraint j with probability p_{ij} greater than α, but the system is said to have failed only if the *joint constraint probability* of $\prod_k p_{kj}$ is greater than α. In other words, a subset of all agents $I_k \in I$ is allowed to violate a constraint provided agents not in I_k follow

"safe" policies such that the joint constraint is still satisfied. In the language of Ono and Williams, the problem we seek to solve in this work is given by

$$\min_{\mathbf{U}^{1:I}} \sum_{i=1}^{I} J^i(\mathbf{U}^i) \tag{5.5}$$

$$\text{s.t.} \quad \mathbf{x}_{t+1}^i = \mathbf{A}^i \mathbf{x}_t^i + \mathbf{B}^i \mathbf{u}_t^i + \mathbf{w}_t^i \tag{5.6}$$

$$\mathbf{u}_{min}^i \leq \mathbf{u}_t^i \leq \mathbf{u}_{max}^i \tag{5.7}$$

$$P\left[\cup_{I_k \in I} \cap_{i=1}^{I_k} \cap_{n=1}^{N_i} h_n^{iT} \mathbf{X}^i \leq g_n^i\right] \geq 1 - S \tag{5.8}$$

In the formulation shown above, the union operator ($\cup$) is non-convex, thus making it difficult to provide optimality guarantees in a manner similar to Ono and Williams. Other literature addressing the problem of planning under constraints includes Luders and How [69] who also have addressed the problem of avoiding obstacles with finite probability, but by using Rapidly-exploring Random Trees (RRT) as the underlying planner due its better scalability. All the work cited above uses approximation techniques that rely on process noise in the system being Gaussian white noise. Geibel [72] proposes a method for reinforcement learning in the presence of risk and constraints, and use a definition of risk that is very similar to the formulation used later in this work. However their main focus is learning, whereas in this work we focus on fast on-line planning. Work discussed in [100] investigates the problem of consensus in the presence of constraints, but that work addresses the problem of consensus when there are constraints on each agent's estimate of the consensus value. In this work, we are interested in the problem of consensus on a value whose *total* value is constrained and that value is the total team risk.

Previously the multi-agent constrained planning problem was formulated as a constrained Markov Decision Process (MDPs) since MDPs provide a natural framework in which to capture system uncertainty. MDP formulations are also easily extended to the *partially observable* case [101]. A major drawback of MDPs is the issue of scalability - as the size of the state space increases, MDPs become very challenging to solve. Particularly in the multi-agent case, the state space and action space

of the MDP grows exponentially in the number of agents. In the previous chapter we proposed and demonstrated a method for dealing with this scaling problem. When the agents are *transition independent*, i.e. the actions of one agent only affect the dynamics of that agent, we can decompose the single, large MDP into a number of significantly smaller MDPs, all of which can be solved in parallel and with far less computational effort than solving the original MDP. Formally, transition independence is defined as follows. If the state vector can be decomposed into N subvectors $\mathbf{s} = [\mathbf{s}_1^T, \cdots \mathbf{s}_i^T, \cdots \mathbf{s}_N^T]^T$ and the action vector into sub-vectors $\mathbf{a} = [\mathbf{a}_1^T, \mathbf{a}_2^T, \cdots \mathbf{a}_i^T, \cdots \mathbf{a}_N^T]^T$ such that $p(\mathbf{s}_i'|\mathbf{s}_i, \mathbf{a}_i, \mathbf{a}_k) = p(\mathbf{s}_i'|\mathbf{s}_i, \mathbf{a}_i)$ for all $k \neq i$, the MDP $< S, A, T, R >$ is said to be transition independent. In this case, $\mathbf{s}_i$ is the state of agent i, and $\mathbf{a}_i$ is the action of agent i. Therefore the multi-agent system under consideration is transition independent due to the fact that the agent dynamics are decoupled. Each agent's individual transition model will henceforth be denoted $T_i(\mathbf{s}_i', \mathbf{a}_i, \mathbf{s}_i)$. The only coupling between the agents occurs in rewards and constraints. Existing literature (e.g. [90]) has investigated reward coupling in the unconstrained case, and in this work we account for constraint coupling.

The constraint coupling between the agents is represented as follows. We define an *event* e_i as the tuple $< S_i, A_i, S_i >$. Agent i is said to have experienced an event e_i when there is a transition from $\mathbf{s}_i \in S_i$ to $\mathbf{s}_i' \in S_i$ under the action $\mathbf{a}_i \in A_i$. Then, we define a *joint event* as the tuple $< e_1, e_2, \cdots e_N, JC >$ where e_i is an event associated with agent i and JC is the *joint constraint penalty* that is awarded to every agent when events $e_1, \cdots e_N$ all occur. Note that this is in addition to the penalty awarded by the constraint model $C_i(\mathbf{s}_i, \mathbf{a})$. We further define $p_{ij}(\pi_i)$ as the probability that event e_i in the joint event j will occur when agent i follows policy π_i. With these definitions, we can write the constrained optimization problem from Eq. 2.2 and Eq. 2.3 as

$$\pi_i^* = \arg\max_{\pi} E\left[\sum_{t=0}^{T} \gamma^t R_i(\mathbf{s}_{it}, \pi_i(\mathbf{s}_{it}))\right] \tag{5.9}$$

$$\text{s.t. } E\left[\sum_{t=0}^{T} C_i(\mathbf{s}_{it}, \pi_i(\mathbf{s}_{it}))\right] + \sum_{j=0}^{E} JC_j \prod_{k=0}^{N} p_{kj}(\pi_k) \leq \alpha \tag{5.10}$$

Knowledge of the other agents' policies are summarized in $p_{kj}(\pi_k)$. The additional term added to the left side of the constraint (Equation 4.2) is the expected *joint constraint penalty.* Since one MDP must be solved for each agent, we have to solve N MDPs of size $S \times A$, rather than one single MDP of size $S^N \times A^N$. However, the policy for any agent depends on $p_{kj}(\pi_k)$ of all the other agents, which in turn depend on their policies. Thus each agent picks an initial policy, and the $p_{ij}(\pi_i)$ (probability of its own event e_i) corresponding to that policy is communicated to all other agents. Since all the other agents perform the same procedure, agent i also receives values for $p_{kj}(\pi_k)$ from all other agents $k \neq i$. Agent i then picks a policy that is optimal for the received set of $p_{kj}(\pi_k)$. This policy yields a new $p_{ij}(\pi_i)$, and the process is repeated until the all agents converge on a policy. This is called the *joint policy iteration* and is shown in Algorithm 6 in Chapter 4. Once this process is completed, the agents each have a policy and its corresponding risk. In the previous chapter, we introduced an approximation - called the *Max Risk heuristic* - that assumes that the other agents never exceed this amount of risk. The conservatism introduced by the heuristic reduces team performance, and in this chapter we present a method by which risk can be redistributed during plan execution. But first, we show with the help of an example the potential impact of not redistributing risk.

5.2 Example

In this example we see the disadvantage of being able to take on more risk during mission execution. We have two agents, both of which start to the left side of a long and narrow corridor. Red indicates constrained areas, and green indicates rewards. The agents each have a planning horizon of length $T = 4$. The *events* in this problem are transitions into constrained states. The additional penalty for both agents transitioning into constrained states is set to be $JV = 1$. The vehicle model used is the same as shown in Figure 4-8. The probability of taking a step in the intended direction is 0.9 and the probability of moving in a direction perpendicular is 0.1. Risk for each agent is constrained to be less than 0.05 ($\alpha = 0.05$), and we use memoryless

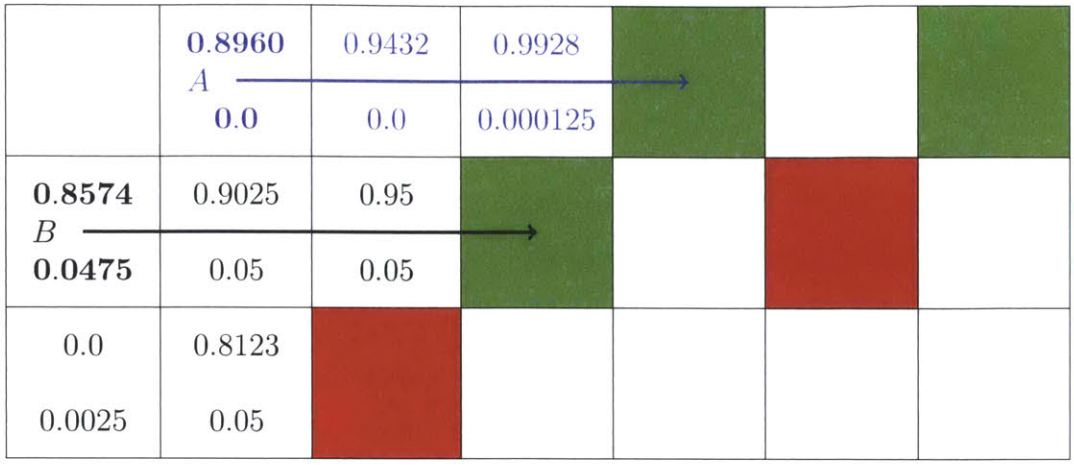

Figure 5-1: Agents A and B both plan their actions up to a horizon of length $T = 4$. Initially neither agent sees the rightmost reward. Agent A plans a zero-risk path, whereas Agent B takes some risk to reach its reward sooner.

risk.

Initially, the two agents each decide upon plans shown in Figure 5-1. In each state, the number at the top represents the expected reward under the policy shown, and the number in the bottom shows the expected risk. Agent B takes a starting risk of 0.0475 (which is the risk associated with the highest-reward constraint-feasible policy) while Agent A takes 0 risk. Under a fixed allocation strategy, the agents are henceforth constrained to keep their risk values below these initial allocations. However as both agents begin executing their respective policies and move towards their rewards, a second reward appears within Agent A's planning horizon. This reward requires Agent A taking higher risk than previously announced, specifically a risk of 0.0475. Under a fixed risk allocation, Agent A is not allowed this higher risk since it has no means by which to communicate with Agent B and ensure that Agent B's policy does not become infeasible. Note that in fact Agent B's current plan would not become infeasible - its total risk will now be $0.0475 + 0.05 \times 0.05 \times 1 = 0.05$, where the additional term $0.05 \times 0.05 \times 1$ represents the joint risk. Thus a fixed risk allocation is unnecessarily conservative - even in this simple example, allowing the agents to communicate and arrive at a *consensus* on taking greater risk can significantly improve reward.

The underlying features of the problem that make reward redistribution impor-

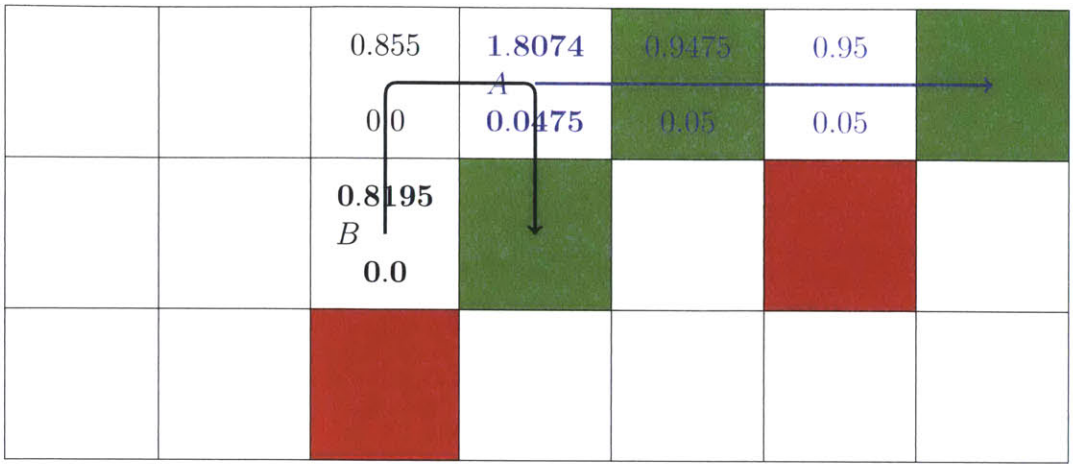

Figure 5-2: After the agents have both executed two time steps, Agent A sees that it can achieve twice the reward by taking more risk. Agent A bids on more risk, and Agent B identifies that it can give up some of its risk to Agent A while still increasing the overall team reward.

tant are found frequently in the types of problems we are considering. From the perspective of the agents, the environment is dynamic - either because the rewards and constraints themselves are changing, or - as was the case in this example - because new rewards appear within the horizon of the planner as the mission is executed. In the risky environments that this work addresses, both those conditions are expected to be common, hence the importance of being able to renegotiate risk during mission execution.

5.3 Proposed Solution

As we have seen, allowing an agent to modify its policy and take on more risk as the mission progresses has clear advantages. However, if any one agent i unilaterally changes its policy, the policies of all other agents might need to change to accommodate the additional risk that agent i might be taking. This in turn changes the risk of all other agents k, requiring agent i to recompute its own policy. Clearly if all agents are changing their policies simultaneously, the outcome may be unpredictable and may not converge to a feasible solution. Therefore we design an algorithm that lets only one agent modify its policy at a time, and select this agent in a way that

maximizes the benefit to the entire team (as indicated by the total reward).

The algorithm works in two stages. In the first stage, all agents initialize their $p_{kj}(\pi_k)$'s. Using these $p_{kj}(\pi_k)$'s, every agent solves Equations 4.1 and 4.2. This yields the reward improvement each agent k expects, given by ΔR_k. The agents all broadcast their ΔR_k to all other agents. Each agent now compares the ΔR_ks that is has received from the other agents, and if its own ΔR_i is the highest, it broadcasts its $p_{ij}(\pi_i)$ value. The agent thus essentially "wins" the right to keep its optimal policy with risk $p_{ij}(\pi_i)$.

In the second stage, the other agents need to recompute their policies for the new $p_{ij}(\pi_i)$, which we call $p'_{ij}(\pi_i)$. All agents $k \neq i$ thus compute their new policies, and broadcast their new ΔR_k. With this information, every agent (including i) computes the new team reward $\sum \Delta R_k$. If the new team reward is greater than the previous team reward by less than ϵ, where $\epsilon \geq 0$, $p'_{ij}(\pi_i)$ is rejected and $p_{ij}(\pi_i)$ is restored to its previous value. The agent with the next highest ΔR_k is allowed to broadcast its $p_{kj}(\pi_k)$, and the second stage is repeated. If a $p'_{ij}(\pi_i)$ is not rejected, both the first and second stage are repeated with the new set of $p_{kj}(\pi_k)$'s. The complete algorithm is shown in Algorithm 8.

Graphically, we can visualize the progress of the algorithm as follows. For simplicity, assume there are only two agents and one joint event, so the probabilities associated with each event is given by p_{11} (probability that Agent 1 will experience event 1) and p_{21} (probability that Agent 2 will experience event 1). The event probabilities are determined by the agents' policies π_1 and π_2, and this dependence is made explicit by writing the event probabilities as $p_{11}(\pi_1)$ and $p_{21}(\pi_2)$. Since each pair of policies (π_1, π_2) is associated with a unique $p_{11}(\pi_1)$ and $p_{21}(\pi_2)$, we can represent each pair of policies as a point in the space spanned by $p_{11} \in (0, 1)$ and $p_{21} \in (0, 1)$, as shown in Figure 5-3. Note that only policies that lie below the line defined by $p_{11}p_{12} = \alpha$ are constraint-feasible.

Suppose the agents are initially assigned event probabilities of $\hat{p}_{11}$ and $\hat{p}_{21}$. With a fixed risk allocation, agents are allowed to explore only those policies in the region $0 \leq p_{11} \leq \hat{p}_{11} \cap 0 \leq p_{21} \leq \hat{p}_{21}$, shown as the shaded region in Figure 5-4. If the optimal

Algorithm 8 The two-stage Risk Auctioning Algorithm

Initialize $\rho_1, \rho_2, \cdots \rho_N$, $R_0 = -\epsilon$, $R = 0$
while $R > R_0$ **do**
  Stage 1
  Solve MDP(i) for ρ_i unconstrained
  Send ΔR_i to all $j \neq i$, **Receive** ΔR_j from all $j \neq i$
  Stage 2
  $J = \{1, 2, \cdots N\}$, **StageComplete** = false
  while NOT StageComplete AND $J \neq \emptyset$ **do**
    $k = \arg\max_{j \in J} \Delta R_j$
    $\rho_{0k} = \rho_k$
    $\rho_k = \rho'_k$
    Solve R'_i = MDP(i) for bounded ρ_i
    Send R'_i to all $j \neq i$
    Receive R'_j from all $j \neq i$
    if $\sum R'_j > R + \epsilon$ **then**
      $\rho_k = \rho_{0k}$
      $R_0 = R$
      $R = \sum R'_j$
      StageComplete = true
    else
      $J = J \backslash k$
    end if
  end while
end while

policies (optimal defined as maximum team reward) are located outside this region as shown in Figure 5-4, a fixed risk allocation strategy will not find these policies.

We can now follow the progress of the risk negotiation algorithm in the (p_{11}, p_{21}) space. First, both agents compute their individual optimal reward while keeping the risk for the other agent fixed. Thus Agent 1 searches along the blue line shown in Figure 5-5 and Agent 2 along the vertical red line. Both agents find their individual optimal policies π'_1 and π'_2 along these lines and communicate the reward and risk associated these policies to each other. The agent whose policy yields the highest individual reward improvement wins its bid. Assume that Agent 1 wins the bid, thus keeping its $p_{11}(\pi'_1)$. In Stage 2 of the algorithm, Agent 2 has to recompute its optimal reward given the new $p_{11}(\pi'_1)$. Agent 2 thus searches along the line shown in Figure 5-6 and finds its individual optimal policy, giving rise to a new $p_{21}(\pi''_2)$. If the total

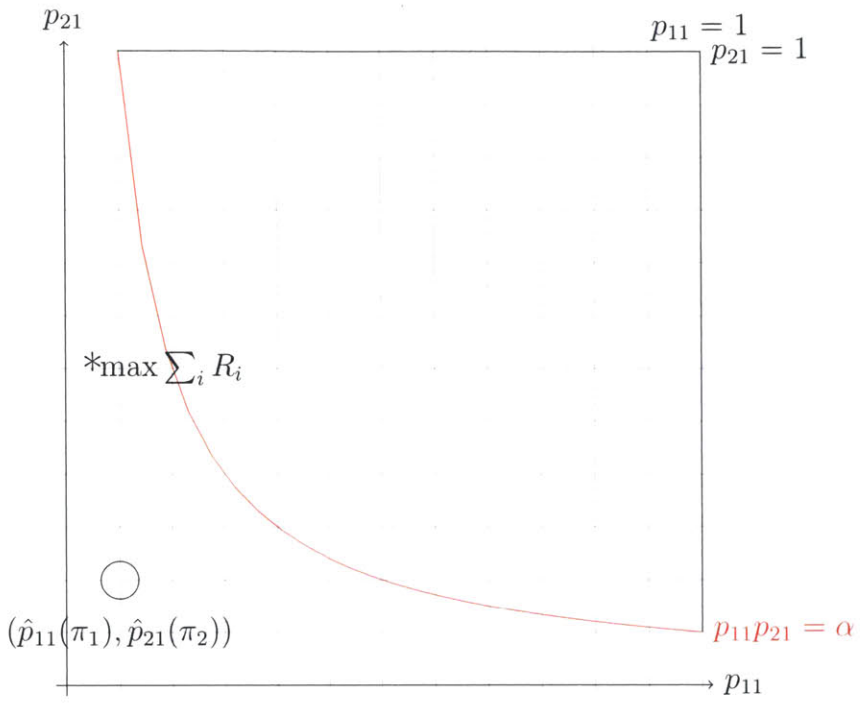

Figure 5-3: Every set of policies $(\pi_1, \pi_2 \cdots \pi_N)$ is associated with a single unique point in the space $(p_{11}, p_{21} \cdots p_{NE})$. Shown here (circle) is the case where $N = 2$ and $E = 1$. Also shown (by *) are the policies associated with the optimal team reward.

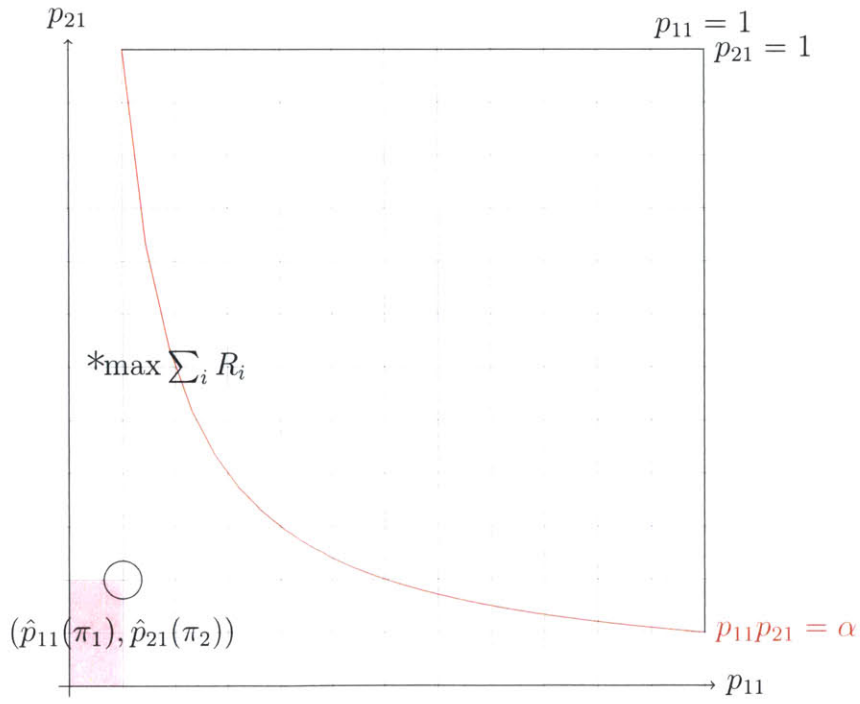

Figure 5-4: The set of policies that agents are allowed to explore is restricted to lie in the shaded region with a fixed risk allocation. The * indicates the policies that yield the optimal team reward.

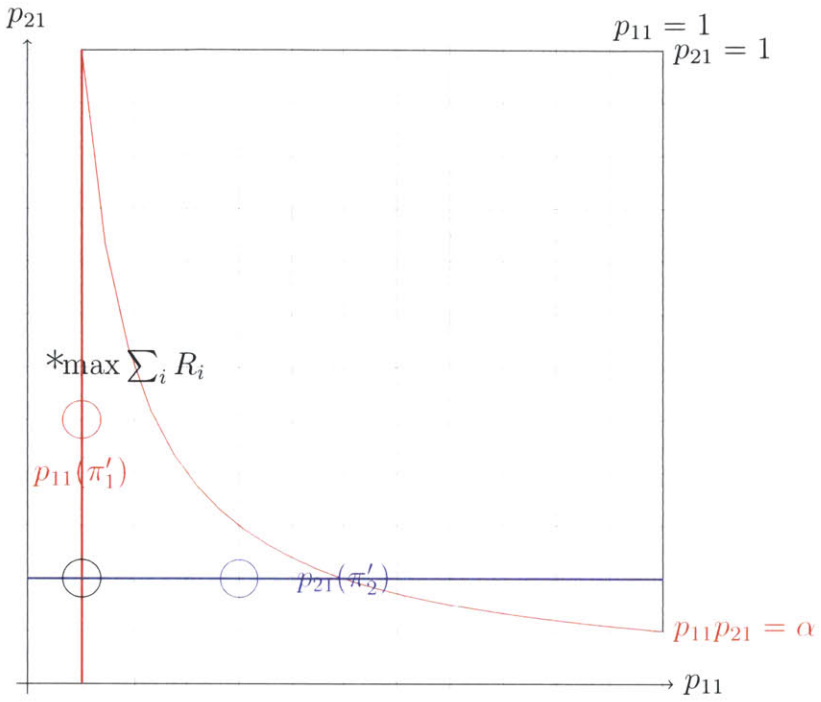

Figure 5-5: In Stage 1 of the algorithm, both agents keep the other agents' risk fixed and find their individually optimal policy. That policy is π_1' for Agent 1 and π_2' for Agent 2. Shown are the risks associated with each policy.

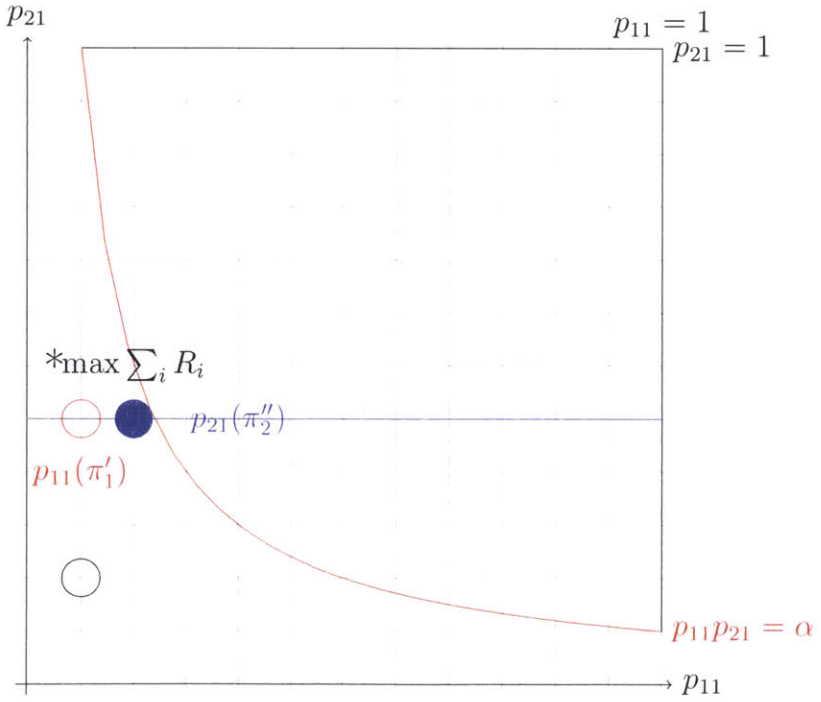

Figure 5-6: Agent 1 wins the right to keep its new policy π_1', and Agent 2 recomputes its new policy π_2'' to account for Agent 1's new policy. The risks associated with the new policies are shown by the solid circle.

team reward at this new $(p_{11}(\pi_1'), p_{21}(\pi_2''))$ is less than the previous optimal team reward $(p_{11}(\pi_1), p_{21}(\pi_2))$, π_1' is rejected. Stage 2 is then repeated, but with keeping π_2' fixed and Agent 1 computing a new policy π_1''. If π_2' is also rejected, the algorithm terminates.

However, if one of the bids is accepted - say Agent 1's bid - there is now a new set of $p_{11}(\pi_1'), p_{21}(\pi_2'')$, and both Agents repeat Stage 1 starting with these policies. Agent 2 does not change its policy (since it is already the optimal individual policy given $p_{11}(\pi_1')$), but it is possible that Agent 1 will find a new optimal individual policy π_1'' along the line $p_{21}(\pi_2'')$, shown in Figure 5-7. If π_1'' is accepted, Agent 2 will have to recompute its policy, and the process continues. There are three ways in which the process can terminate:

1. The process terminates when no agent finds a better individual policy, i.e. the algorithm terminates during Stage 1.

2. The process terminates when all agents' bids are rejected because none of the bids increase team score, i.e. the algorithm terminates during Stage 2.

3. The algorithm terminates when every agent finishes exploring all possible policies, i.e. the algorithm finds the globally optimal team reward.

We can easily verify that the algorithm does not get stuck in cycles, and this is primarily due to Stage 2 - we check to see if the new policy actually improves the total team reward. Suppose, as in the example, the agents start with policies π_1 and π_2, and the team reward associated with the policies is $R(\pi_1, \pi_2)$. Also suppose that Agent 1 wins the bid to change its policy to π_1', in the first iteration, and in the next iteration Agent 2 wins the bid to change its policy to π_2''. By construction of the algorithm, this means that $R(\pi_1', \pi_2'') > R(\pi_1', \pi_2) > R(\pi_1, \pi_2)$. For a cycle to occur, Agent 1 *and* Agent 2 must both win bids to return to policies π_1 and π_2. However, this is possible only if $R(\pi_1', \pi_2'') < R(\pi_1, \pi_2)$ - a contradiction of the previous statement that $R(\pi_1', \pi_2'') > R(\pi_1', \pi_2) > R(\pi_1, \pi_2)$. Thus the check on the team reward ensures that policies that were previously explored are not explored again.

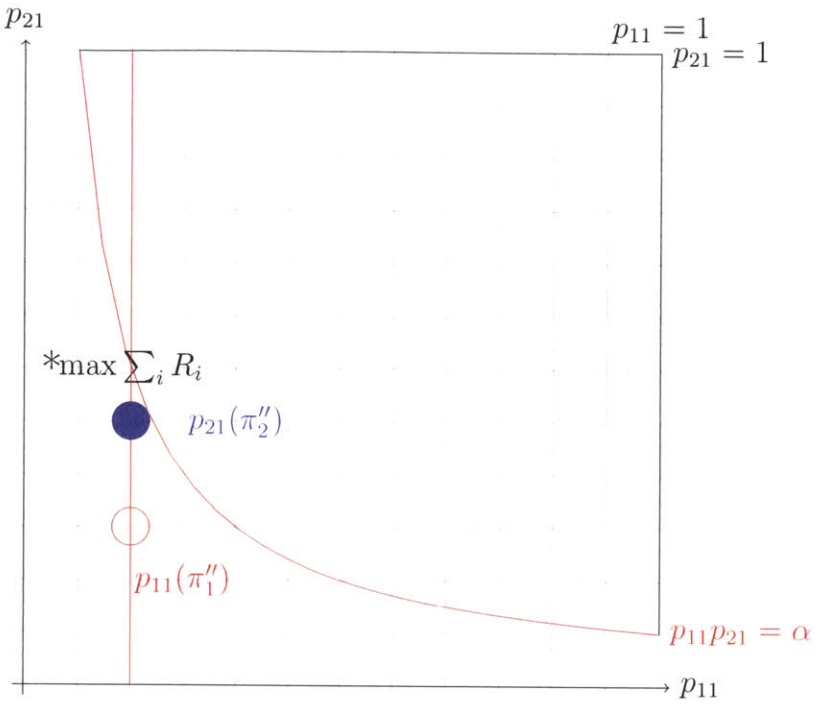

Figure 5-7: In the next iteration, Agent 1 again computes a new individually optimal policy π_1'' while keeping $p_{21}(\pi_2'')$ fixed, and this results in a new risk $p_{11}(\pi_1'')$. If the team reward with this new policy is greater than the team reward with π_1' and π_2'', this new policy is kept and Agent 2 would have to recompute the policy for with the new $p_{11}(\pi_1'')$. In the *worst* case, this process terminates after $|S|^2|A|^2$ iterations (i.e. after all possible policies are explored). We can guarantee that cycles do not exist due to the check in Stage 2 to ensure that the team reward always improves.

This allows us to bound the maximum number of iterations before the algorithm terminates. The total number of policies available to Agent i is given by $|S_i||A_i|$. Assuming $|S_i|$ and $|A_i|$ are countable and finite, the number of policies is finite. Thus the total number of policies the algorithm explores before terminating is *at most* $\prod_{i=1}^{N} |S_i||A_i|$. However in reality the algorithm only explores feasible policies - the total number of feasible policies is only a subset of all policies. The size of this subset depends on a number of factors - the transition model, the constraint model, and α and is difficult to estimate in general. Furthermore, the existence of constrained states from which no further actions are possible also reduces the total number of policies, from $|S_i||A_i|$ to $(1-\rho_C)|S_i||A_i|$ where ρ_C is the constraint density - the fraction of states that are infeasible. Taking this into account, we arrive at

$\prod_{i=1}^{N}(1-\rho_C)|S_i||A_i|$ as a loose upper bound on the number of iterations required for the algorithm to terminate. Note that in particular if $\epsilon > 0$, the algorithm may not explore all possible solutions, but will terminate once it finds solution that cannot be improved by more than ϵ.

By construction, the algorithm terminates when no agent can find a higher-reward policy that preserves feasibility of all agents' policies and improves the overall team performance. Based on this, the following shows that **the algorithm converges to a *Pareto optimal* solution**. A Pareto optimal solution is one in which no agent can improve its own reward without reducing the reward of at least one other agent [102]. We show this first for the case $\epsilon = 0$, where ϵ is the minimum improvement in team reward for an agent's bid to be accepted.

Suppose the algorithm has terminated at a set of policies $\pi_1, \cdots \pi_i, \cdots \pi_N$, that have rewards given by $R_1(\pi_1), \cdots R_i(\pi_i), \cdots R_N(\pi_N)$ and event probabilities given by $P(e_1|\pi_1), \cdots, P(e_i|\pi_i), \cdots P(e_N|\pi_N)$. Suppose these policies are *not* Pareto optimal such that one agent, Agent i, can feasibly change its policy to π_i' and increase its own reward to $R_i(\pi_i') > R_i(\pi_i)$ without requiring any of the other agents $k \neq i$ to get lower rewards. In such a case, running the risk consensus algorithm for one more iteration results in Agent i computing the new policy π_i' and bidding for it in Stage 1. Since we assumed that only Agent i can feasibly achieve a higher reward while keeping other agents' policies fixed, Agent i is the only agent with a non-zero reward improvement. Therefore Agent i wins Stage 1. In Stage 2, agents $k \neq i$ recompute their policies to account for Agent i's new policy. Since the agents' original set of policies were not Pareto optimal, all $k \neq i$ agents will be able to achieve at least the same reward performance even with Agent i's new policy. Furthermore, since the reward of $k \neq i$ agents get no worse and Agent i's reward has increased, the overall team reward also increases. Thus Agent i's new policy is accepted, contradicting the supposition that the algorithm had terminated at a non-Pareto optimal solution. Thus we show that for $\epsilon = 0$ the risk consensus algorithm does not terminate unless at a Pareto optimal solution.

We can show this to be true in a more general case. Suppose the algorithm

has terminated at a set of policies $\pi_1, \ldots \pi_i, \ldots \pi_N$, that have rewards given by $R_1(\pi_1), \ldots R_i(\pi_i), \ldots R_N(\pi_N)$ and event probabilities given by $P(e_1|\pi_1), \ldots, P(e_i|\pi_i)$, $\ldots P(e_N|\pi_N)$. Suppose these policies are *not* Pareto optimal such that M agents, Agents 1 to M where $M \leq N$, can feasibly change their policies to $\pi'_1, \ldots \pi'_M$ and increase their own rewards to $R_i(\pi'_i) > R_i(\pi_i) \quad \forall i \in (1, \ldots, M)$ without requiring any of the other agents $k \in (M+1, \ldots N)$ to get lower rewards. In such a case, running the risk consensus algorithm for one more iteration results in Agents 1 to M computing the new policies $\pi'_1, \ldots \pi'_M$ and bidding for these policies in Stage 1. One of these agents, say Agent 1, has the highest reward improvement $(R(\pi'_1) - R(\pi_1) > R(\pi'_i) - R(\pi_i) \quad \forall i \in (2, \ldots M))$. Agent 1 thus wins Stage 1. In Stage 2, all agents $k \neq 1$ must compute new policies $\pi''_2, \ldots \pi''_N$ to account for Agent 1's new policy. Since the original set of policies were not Pareto optimal, a change in Agent 1's policy does not reduce the reward of any of the other agents $(R(\pi''_i) \geq R(\pi_i) \quad \forall i \in (2, \ldots N))$. As Agent 1's reward has increased and all other agents' rewards have not decreased, the overall team performance has improved and Agent 1's new policy π'_1 is accepted. This contradicts the supposition that the algorithm had terminated, hence again showing that the risk consensus algorithm does not terminate at a non-Pareto optimal solution. Intuitively, the algorithm terminates when any gain by one subset of agents requires offsetting losses to some other subset of agents. For the case where $\epsilon = 0$, the algorithm terminates when the gains are exactly offset by the losses.

For the case $\epsilon > 0$, we must consider the possibility that the algorithm may terminate without converging to the Pareto optimal solution. Consider the general case described above. If there is even a single agent which can improve its reward by more than ϵ without the other agents reducing reward, the algorithm does not terminate. But suppose all M agents that can improve their reward can do so by less than ϵ. In that case, all M agents may have their bids rejected, leading to an early termination of the algorithm. Thus the algorithm will converge to solution whose team reward is within $M\epsilon$ of a Pareto optimal solution. Since $M = N$ in the worst case, we see that in the worst case the algorithm will terminate with a team reward

that is within $N\epsilon$ of a Pareto optimal solution.

Another relevant notion of optimality is *Hicks optimality* [103], which is defined as a solution where no agent can improve its own performance without degrading the *total* team performance. However we *cannot* guarantee that the risk consensus algorithm will converge to a Hicks optimal solution. We show this by using the following counter-example. Using the same notation as before, suppose that during the optimization, the agents currently have a set of policies $\pi_1, \ldots \pi_i, \ldots \pi_N$, that have rewards given by $R_1(\pi_1), \ldots R_i(\pi_i), \ldots R_N(\pi_N)$ and event probabilities given by $P(e_1|\pi_1)$, ..., $P(e_i|\pi_i)$,... $P(e_N|\pi_N)$. Assume that these policies do not constitute a Hicks optimal solution. In other words, a new solution exists in which one agent, Agent i, can improve its own reward by adopting a new policy π_i' if in addition one other agent, Agent j, adopts a policy π_j' that has a lower reward. Finally, assume that the policies of all other agents remain the same and that

$$|R_i(\pi_i') - R_i(\pi_i)| > |R_j(\pi_j') - R_j(\pi_j)|$$

so that the overall team reward improves by agents i and j changing their policies to π_i' and π_j' respectively. In Stage 1 of the algorithm, when agent i initially computes its potential reward improvement, policy π_i' is *not guaranteed* to be identified as a higher-reward policy. This is because π_i' yields higher reward, but only when Agent j follows policy π_j'. But in computing which policy to bid on in Stage 1, Agent i is constrained to assume whatever policy (π_j in this case) Agent j had *previously* decided upon. When Agent j follows policy π_j, there is no guarantee that π_i' will yield a reward improvement, or even that it will be feasible. Thus in Stage 1 of the consensus algorithm Agent i is not guaranteed to identify π_i' as a higher-reward policy than π_i, and is not guaranteed to bid for that policy. Thus it is possible that the algorithm may terminate at this solution. However π_i and π_j are not a Hicks optimal solution, so the algorithm will have terminated at a non-Hicks optimal solution. Furthermore, the algorithm will not find any other Hicks optimal solution because it has terminated. Intuitively, the reason the algorithm cannot be guaranteed

to find the Hicks optimal solution is because both Agent i and j would need to modify their policies *simultaneously*. Since the algorithm does not allow that, it is possible that it will terminate with a solution that is not Hicks optimal. We observe that every Hicks optimal solution is a Pareto optimal solution, but not every Pareto optimal solution is a Hicks optimal solution. Hicks optimality is a "stronger" optimality condition (stronger from the perspective of the team reward) and the risk consensus algorithm does not achieve those stronger conditions.

Since in Stage 2 of the algorithm we specify that the leader's new risk value can be accepted only when there is an overall team performance improvement, it is clear that the team performance never gets worse. Also, since the total reward available in the environment is finite, the algorithm will eventually converge to a finite value which *at most* will be the maximum reward that can be obtained by the team if no constraints were present.

Since the algorithm requires agents to communicate, the performance critically depends on the connectivity between the agents. The communication structure is given by a *graph* $G = (V, E)$ that is defined by a set of nodes V and edges E. The nodes in this case are the agents, and the existence of an edge between any two nodes (agents) implies that those agents can communicate with each other. The *diameter* of the graph D_G is the longest path between any two nodes in the graph. A *fully connected* graph (where every node is connected to every other node) has diameter 1. A *strongly connected* network is one in which a path exists from any node to any other node, although there may not be a direct link between each node. Given these definitions, and assuming the communication network between the agents to be strongly connected, each iteration of the algorithm requires $2D_G$ communication steps. This can be seen by observing that the first stage of the algorithm is essentially a leader election problem, with the message being their expected performance gain ΔR_k. The election of a leader takes D_G steps [104]. Next, the agents all compute their new reward, using the new risk value for the "leader", and their new performance is transmitted to all agents they are connected to. It takes D_G time steps for all agents to receive updates about all the agents' new performance values, and the decision to

accept or reject the leader's new risk value can be taken only after this stage. Thus, the communication time to arrive at a decision in each iteration is $2D_G$ steps.

Clearly, the worst-case limits for the algorithm are not very strict. We have already seen that the complexity of solving an individual agents' planning problem is of size $O(S^2AE)$, and thus when the number of joint events E is very large, the complexity of solving individual planning problems is significant. Furthermore, the number of event probabilities that need to be communicated also grows. In addition, if there are a large number of events, it is very likely that a small change in the policy of one agent will require other agents to also change their policies, since more events indicate greater coupling between the agents. In other words, a large number of events increases the likelihood that the number of policies explored will approach the worst-case upper bound. Intuitively, the larger the number of events, the greater the coupling between the agents and therefore the greater the deviation from the assumption of transition independence. Thus with many events, solving several decentralized planning problems approaches the complexity of solving the complete centralized planning problem. Assuming transition independence allowed us to avoid the "curse of dimensionality" associated with the centralized planning problem, but as that assumption becomes less valid the curse of dimensionality again returns, although in a slightly different form. The increase in the number of events, the increase in the number of iterations required to terminate the risk negotiation, and the increase in communications requirements are therefore all a manifestation of the curse of dimensionality in domains where transition independence is a poor assumption.

5.4 Results

We compare the performance of the Risk Negotiation algorithm with a fixed risk allocation. The types of problems considered are shown in Figure 5-8. The world is discretized into a grid with reward (green) and constrained states (red). The location of the rewards is chosen at random, and the dangers are randomly placed within a

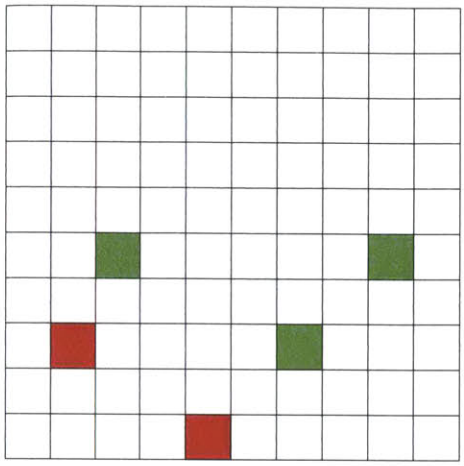

Figure 5-8: Grid world example with rewards (green) and constraints (red)

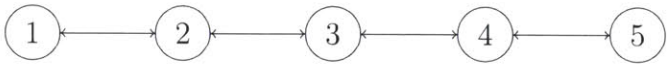

Figure 5-9: The communications architecture for a team of five agents

finite distance from the rewards. This reflects the fact that in a CBRNE scenario mission performance typically involves going into danger areas. Entering a danger area causes the agent to fail. The number of states that are constrained, defined as the *constraint density*, is varied, and the impact on performance and risk is discussed here. We use team sizes of two, five and ten agents. The size of the state space of the full problem, formulated with joint state and action spaces, varies from 1.024×10^7 for the two-agent case to 1.126×10^{35} for the ten-agent case. The agents are assumed to have limited communication capability and can therefore communicate with only two other agents, forming a line network. An example of such a network for a team of five agents is shown in Figure 5-9. The diameter of a line network $D_G = N$ where N is the number of agents. For the five and ten agent teams, the constraint imposed is that no more than three agents should fail. For the two-agent team, we require that no more than one agent fail. The allowed risk α is set to 0.15. Note that these constraints are *joint constraints* since they are imposed on the team and not on any individual agent. In this case, there are no individual constraints.

In Figures 5-10, 5-11 and 5-12 we compare the performance of the risk negoti-

ation algorithm with a fixed risk allocation for teams of two, five and ten agents respectively. For very low constraint densities, the risk negotiation algorithm does not achieve significantly higher reward than a fixed risk allocation, since the risk required for good reward performance is low. As the constraint density increases, using a fixed risk allocation becomes increasingly conservative leading to a significant drop in performance. The risk negotiation algorithm also achieves lower reward - the more constraints in the environment, given the same risk bound, the lower the maximum reward that can be obtained. However the reduction in performance is less pronounced than for a fixed risk allocation since the risk negotiation algorithm is less conservative. For very high constraint densities both methods give low reward, and this reward is close to zero. This is to be expected, since a highly constrained environment is one in which the agents cannot access most reward states easily.

As mentioned previously, the risk negotiation algorithm is less conservative than a fixed risk allocation. Thus we expect that the risk taken by the risk negotiation algorithm will be higher than that taken by fixed allocation. This is seen clearly in Figure 5-13, which shows the risk - specifically, the *accumulated risk* - the total risk taken by the team during the entire mission - for both methods. Clearly, the risk negotiation algorithm accumulates more risk. Note that at any given time during the mission, the *future* risk is constrained by $\alpha = 0.15$, therefore we expect that the accumulated risk (past + future risk) over the course of the mission can be *higher* than α.

The policies computed by the Risk Negotiation algorithm meet the risk requirements only under the condition that the events that constitute a joint event remain independent. In this case, an event corresponds to a single agent entering the red danger zone and failing. Thus by assuming independence of events, we are effectively assuming that the probability of an agent failing is independent of the probability of another agent failing. While this may be an accurate assumption for the problems considered here, it is not necessarily true in general. If the agents' failures were in fact correlated (due to unmodeled environmental factors, for instance), the Risk Negotiation algorithm would underestimate or overestimate risk, leading to policies that

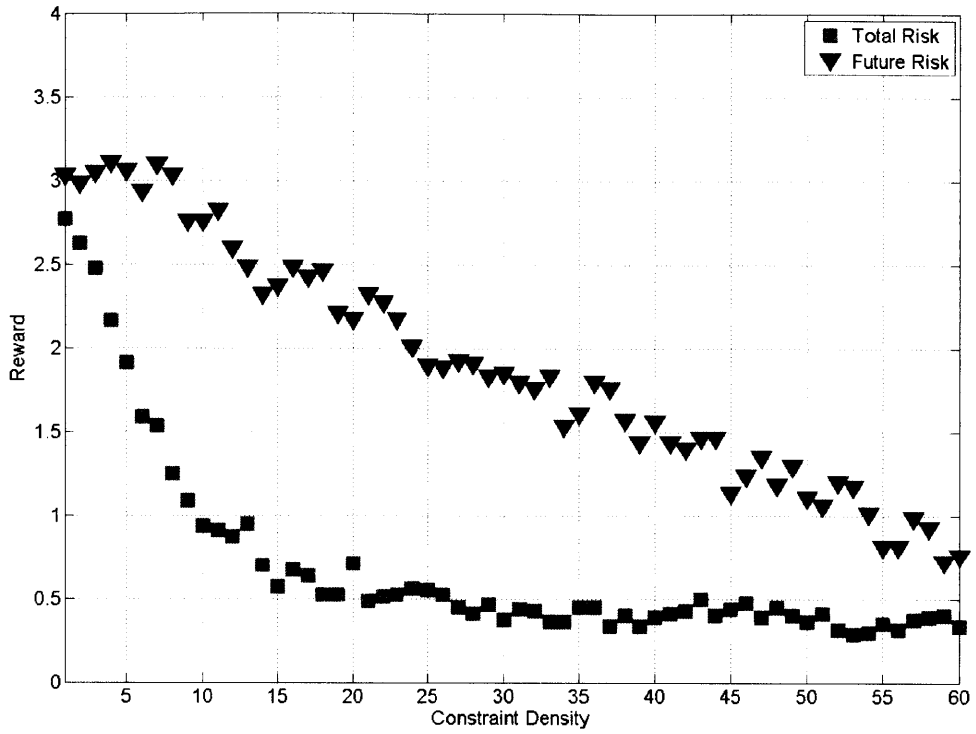

Figure 5-10: Comparison of the performance of the risk negotiation algorithm with a fixed risk allocation for two agents

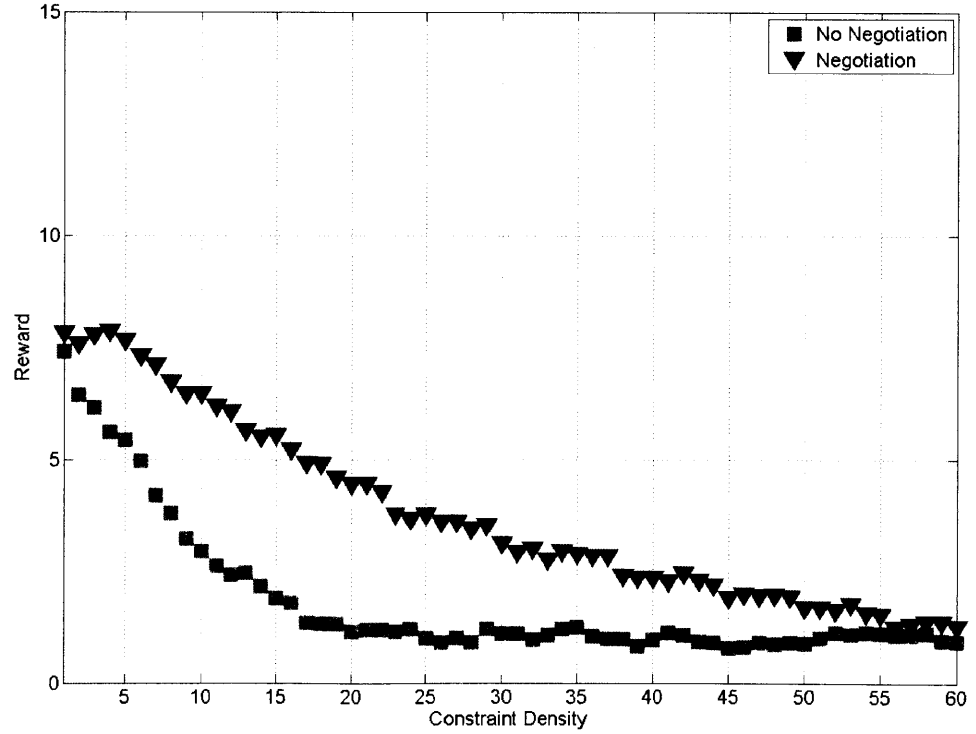

Figure 5-11: Reward obtained by a team of five agents using fixed risk allocation and risk negotiation

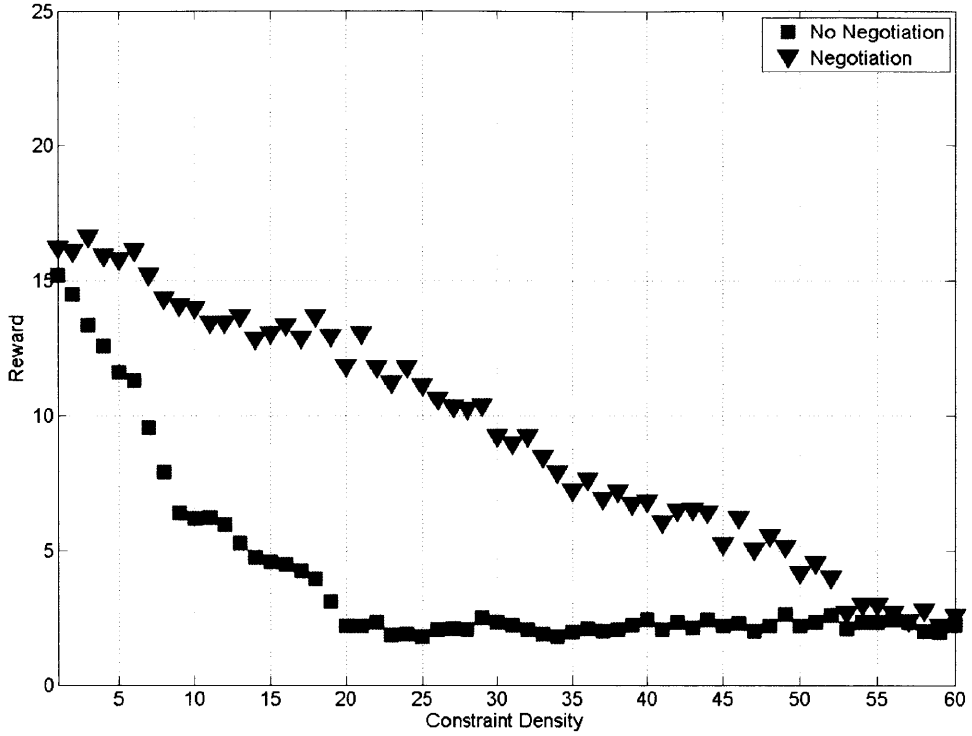

Figure 5-12: Reward obtained by a team of ten agents using fixed risk allocation and risk negotiation

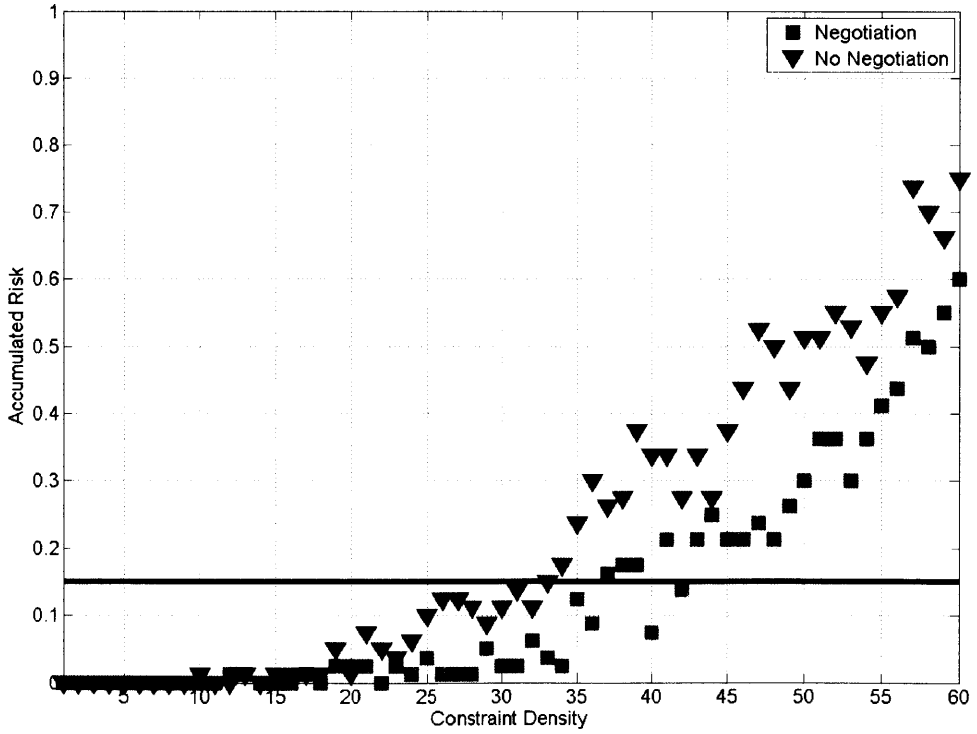

Figure 5-13: Comparison of the risk accumulated by the risk negotiation algorithm with a fixed risk allocation for ten agents. The horizontal line shows the bound on the *future* risk

are either too conservative or too risky, respectively.

5.4.1 Impact of Team Structure

One way to reduce the complexity of the algorithm is to eliminate Stage 1, i.e. eliminate the leader election stage. In place of this stage, we impose a team structure - we always permit the same agent to bid first, another agent to bid second, and so on. The convergence time of the algorithm is faster (D instead of $2D$) but team performance is in fact negatively impacted. We see why this is the case by looking at the example problem in Figures 5-1 and 5-2. Suppose only Agent B is allowed to initiate Stage 2, i.e. the team structure is defined such that Agent B is always computes its best policy and other agents have to adjust their policies to ensure constraint feasibility. In such a case, Agent B would always plan the straight, risky path shown in Figure 5-1, and Agent A will have no option but to plan the path shown in Figure 5-1. In reality, the team can benefit from having Agent B take a longer, less risky path as shown in Figure 5-2. Thus having a rigidly defined team structure makes Agent A unable to execute a higher-reward policy even when such a policy is feasible. Stage 1 of the algorithm picks the agent with the potential to improve performance the most, which in a dynamic environment and a finite-horizon planner can vary through the course of the mission. Thus having a rigid team structure prevents the team from adapting to these unforeseen opportunities and risks.

5.5 Summary

In this chapter, we investigated the problem of achieving good performance for a team of unmanned agents in the presence of constraints. The agents are allowed some probability of violating the constraints, a quantity defined as "risk". We seek a planning mechanism that will maximize the team performance, quantitatively measured as a reward, subject to a constraint on the total risk the team can take. The previous state-of-the-art was to divide the risk allowed for each agent initially and keep those allocations fixed for the duration of the mission. In this work, we proposed an auc-

tioning mechanism through which agents can bid for higher risk allocations if they perceive a performance advantage for taking more risk. This requires other agents to reduce their own risk allocation, potentially reducing their performance. Whether the re-allocation of risk results in a team performance improvement or not is decided by the agents through consensus. Simple examples showed that a significant improvement can be expected through this process. Monte Carlo simulation results were also presented showing a significant improvement in overall team performance particularly in environments with tight constraints.

Chapter 6

Experimental Results

In this chapter, we apply the methods developed previously to solve a planning problem for a large heterogeneous team. The algorithms are tested in a realistic environment - a simulator with a physics engine, Unreal Tournament [105]. First, this test environment is briefly discussed. Next the task planning problem is formulated, and finally the results of the experiments are presented.

6.1 Test Environment: Unreal Tournament

The game Unreal Tournament® is used in this work as a simulation environment. The game has a built-in Physics Engine that can simulate winds, obstacles, collisions, terrains and other real-world dynamics. The game also allows for multiple agents, called "avatars" to be operating in the same environment, thus providing a platform for testing multi-agent algorithms. While some real-world testbeds allow for rapid prototyping [106], in general it is difficult to field many agents at once to test scenarios that involve complex multi-agent teaming. Furthermore, the overhead of maintaining a large team of autonomous agents is significant and can distract from the testing of the algorithm itself. Unreal Tournament thus provides a good balance between eliminating many of the constraints and overhead of a real-world test, while retaining some degree of realism in the operating environment. An additional advantage is the availability of models for a wide range of commonly used robots such as Pioneers,

Figure 6-1: A UGV (left, foreground) and UAV (right, foreground) operating in Unrealville. Also shown are several victims and obstacles in the operating environment such as lamp-posts and trees.

AirRobots, TALONs and MDS robots through the Unreal add-on, USARSim [105].

The operating environment used is an urban setting. Buildings, streets, stationary cars, trees, and various terrains (sidewalks, grass) are provided. Victims, some moving and some stationary, are also provided. A screenshot of the environment, including victims and two robots is shown in Figure 6-1. A map of the environment is shown in Figure 6-2. Grassy areas are shown in green, sidewalks in gray, streets in black, buildings in tan, and water bodies in blue. The main feature of the environment is the inner courtyard area which can be accessed only through a few narrow doorways. Besides the large-scale features shown on the map, there are several smaller ones such as parked cars and lamp-posts. Victims are not shown on the map, but are located throughout the courtyard area.

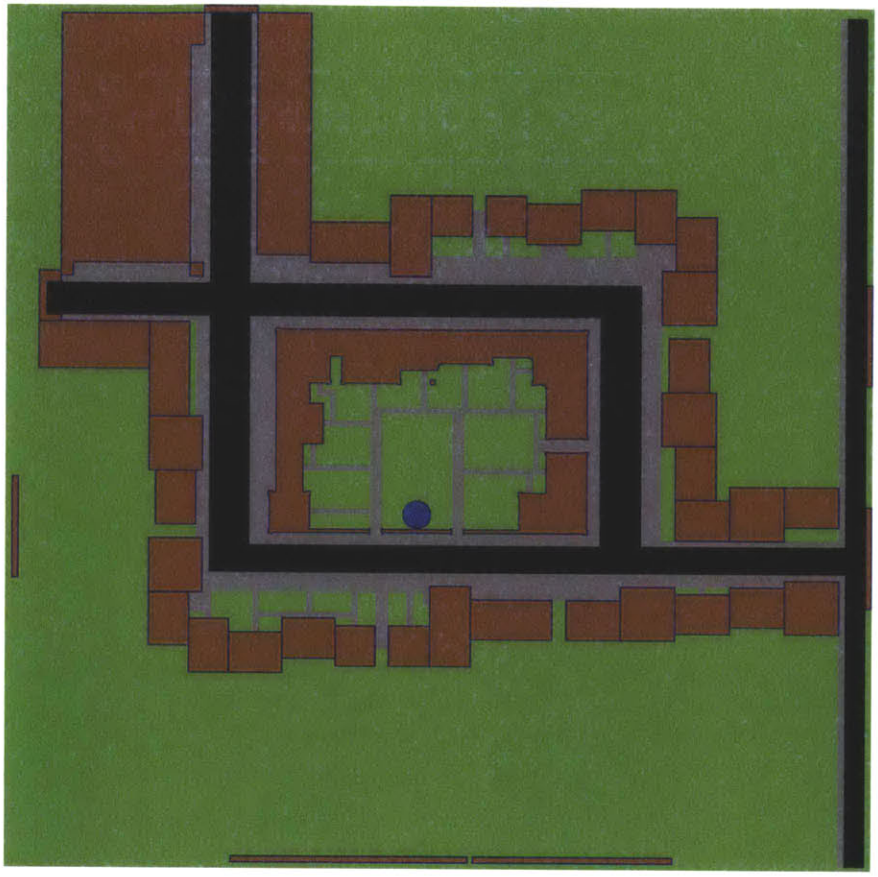

Figure 6-2: The Unrealville test environment. This birds-eye view shows an urban landscape with streets (black), buildings (brown), paved sidewalks (grey), grassy areas (green) and water bodies (blue).

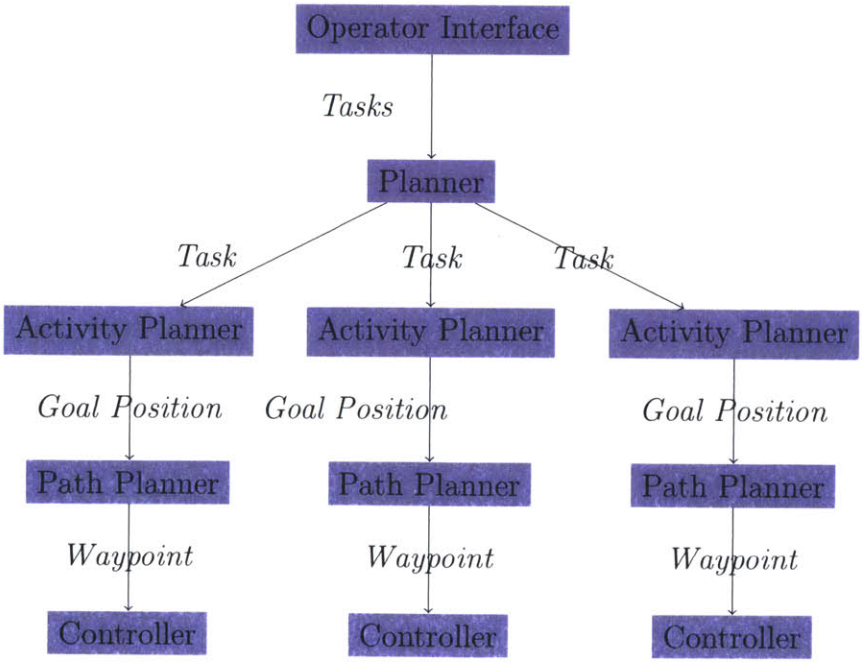

Figure 6-3: System architecture

6.2 Task Planning Problem

The objective of a team of UAVs, UGVs and MDS robots operating in the environment described above is to rescue victims. The team consists of four UAVs, three UGVs (Pioneer P2ATs) and three humanoid MDS (Mobile, Dextrous, Social) robots. The mission requires the execution and completion of several tasks which are specified by a human operator. The tasks generated by the human operator are high level tasks, e.g. "Guide Victims to Safety". These high-level instructions must be interpreted and acted upon by the autonomous planner. The four types of tasks that are realistic for a search and rescue problem are described below.

- **Area Search**: Aerial search of an operator-defined area. The aircraft executing this mission fly search patterns (e.g. spiral, perimeter search, lawnmower patterns) that are also specified by the operator.

- **Bomb Surveillance**: Observe and investigate a suspicious device. This task is executed by ground robots by traveling to the location of the suspicious device and sending video feedback to the operator, and includes potentially manipulating the device as instructed by the operator.

- **Guide Victims**: Guide victims from incident area to safety. Victims that are still inside the area where the CBRNE incident has occurred, but are still mobile are guided out of the incident area by ground robots.

- **Victim Assessment**: Assess whether victims are mobile, ambulatory or not responsive. This requires ground robots to travel to the locations of victims and attempt interaction with them.

The behaviors associated with these tasks are generated by "Activity Planners". There exists one Activity Planner for each agent, as shown in the system architecture in Figure 6-3. The assignment of agents to tasks is performed by the Planner, which is the main object of interest in this work. The tasks themselves are created by the operator, and the operator also specifies the parameters associated with each task.

An interface that allows the operator to intuitively set the parameters defined above is important in the functioning of the overall system. Such an interface would have to include the following.

- **Task Creation**: The operator decides which tasks need to be executed, and the parameters associated with the tasks. For example, the operator decides whether there needs to be a search, and if so which area is to be searched and possibly how many robots are required. Note that the operator can specify more than one task of the same type, e.g. there may be several Area Search activities.

- **Failure Probability Specification**: Associated with each task is the probability that the robot performing this task might fail. These probabilities are also decided by the operator. It is possible that the operator simply selects from several subjective options (e.g. "Very Dangerous", "Safe") but in general it is assumed that the operator has a better intuition (based on past experience or training) as to which tasks are risky which safe.

- **Constraint Specification**: The operator also sets the constraints on the system. For instance, the operator may decide that a minimum of two UAVs and one MDS robot is required for a certain task, or required for the mission overall. This is directly translated into the constraint model in the planner. This information would have to be provided to the planner at the time of task or mission creation.

- **Risk Specification**: Finally, the operator decides the probability with which the constraints can be violated. This is again a probability, and a reasonable interface would be one that allows the operator to select from several options - for instance, the operator could indicate the importance of a constraint rather than the actual allowed risk, and these importance ratings could then be converted into a risk specification. This is preferable since humans are known to be poor at assigning numerical values to abstract notions such as likelihood of

failure. Thus a good interface would query the operator for to select an option that is then converted into a probability.

Clearly, some activities can only be performed by some types of agents. Also the human operator may explicitly specify that a task be performed by a certain type of robot. For instance, the operator may specify that the Area Search activity be performed by an aerial platform. We quantify these requirements as follows. The state space of each vehicle S_i is defined as

$$\mathbf{s}_i = [x_i, y_i, v_i, t_1, t_2, \cdots t_N] \tag{6.1}$$

Where x_i, y_i is the location of the agent on the map, v_i is the status of the agent (active or failed) and t_j is the status of activity j (active or completed). The action space of each agent A_i is the set of tasks that are available. Thus at any time, an agent can execute a maximum of N actions, action j corresponding to task j. However, not all agents can execute all activities. We have three types of agents - UAVs, UGVs and MDS robots. Each of these agents has different capabilities, and Table 6.1 below lists which agents can perform which tasks.

Table 6.1: Capabilities of the three different types of agents

	Area Search	Bomb Surveillance	Guide Victims	Victim Assessment
UAV	Yes	No	No	No
UGV	No	No	Yes	Yes
MDS	No	Yes	Yes	Yes

Thus while each agent can perform a maximum of N activities, in reality the action space of each agent is smaller. If an agent can perform an activity, the reward for executing that activity is given by the reward model shown below:

$$R(\mathbf{s}_i, a_j) = r \quad \text{if } t_j = ACTIVE \tag{6.2}$$

$$R(\mathbf{s}_i, a_j) = 0 \quad \text{if } t_j = COMPLETED \tag{6.3}$$

In other words, an agent that executes an active task receives a reward of $r \in \Re$. r is interpreted as the priority of that task and is set by the operator when the task is first created.

We also account for the fact that an agent might fail while executing a task. This is captured in the transition model as follows:

$$T(\mathbf{s}_i', a_j, \mathbf{s}_i) = p_j \quad \text{if } v_i' = FAILED \text{ and } v_i = ACTIVE \tag{6.4}$$

Note that the probability of an agent failing, p_j, depends on the task that it is executing. For instance, p_j might be very high for a Bomb Surveillance task, but significantly lower for an safer Area Surveillance task. The exact numbers used are shown in Table 6.2.

Table 6.2: Probability of an agent failing during the execution of an activity

Task	Agent Failure Probability
Area Search	0.01
Bomb Surveillance	0.25
Guide Victims	0.1
Victim Assessment	0.1

The probabilities reflect the fact that Bomb Surveillance is an extremely dangerous activity. Guide victims and victim assessment are also considered risky since they involve entering a zone with potential hazards. Area search is considered the safest, since it is executed by air (only UAVs are capable of performing Area Search).

Next, we define the constraint model. In this problem, we impose two constraints, C_1 and C_2. The first constraint which we label C_1, constrains the total number of active agents to be greater than or equal to a minimum number M_{min}. Note that this constraint is a *joint constraint*, i.e. it acts upon the team as a whole and not individual agents. As noted previously, it is through joint constraints that coupling is introduced between the plans of each individual agent. In order to write the joint constraint, we must first define the *event* e_i as the event than an agent i fails. Formally, this is

expressed as

$$\mathbf{e}_i = (\mathbf{s}'_i, a_j, \mathbf{s}_i) \quad \forall j, \forall \mathbf{s}_i \in S, \forall \mathbf{s}'_i : v'_i = FAILED \tag{6.5}$$

Since we want to ensure that there are M_{min} active agents out of a total of M agents, we have $E = \binom{M}{M_{min}}$ joint events, each with a different combination of agents. In the following set of results we set the joint constraint penalty is set to $JV = 1$, and M_{min} is set to 7. Note that M_{min} can be set to any value between 0 and M. Using $M_{min} = 0$ essentially removes the constraint. Using $M_{min} = M$ results in extremely conservative assignments. Therefore we pick an intermediate value of M_{min} and in a later section discuss ways in which varying M_{min} impacts the performance of the planner. The second constraint C_2 requires that at least two MDS robots be operational at all times. This is due to the fact that MDS robots have capabilities that the other agents do not, specifically the ability to interact with humans. The constraint model therefore also has to capture this requirement. This is set by defining an additional set of events specifically for the MDS robots.

$$\mathbf{e}_{MDS_i} = (\mathbf{s}'_{MDS_i}, a_j, \mathbf{s}_{MDS_i}) \quad \forall j, \forall \mathbf{s}_{MDS_i} \in S, \forall \mathbf{s}'_{MDS_i} : v'_{MDS_i} = FAILED \tag{6.6}$$

Where MDS_i represents the ith MDS robot. Besides C_1 and C_2, we impose no further constraints. No local constraints are imposed on any of the agents. We run the decentralized online planner developed in Chapter 5 and test its performance in several cases. We investigate the behavior of the planner closely and show that it is capable of handling agent failures and environmental changes during mission execution. Specifically, we look at the behavior of the planner in five cases:

1. **Single UAV Failure**: During mission execution, we simulate the failure of a UAV conducting an Area Search activity and observe that the total team risk increases due to the reduction in team size. The planner therefore re-assigns an MDS robot from the risky Bomb Surveillance task to a Victim Assessment task.

2. **Single MDS Failure**: We simulate the failure of an MDS robot performing a Victims Assessment activity and show that the planner responds by re-assigning the MDS performing the Bomb Surveillance task to a Victim Assessment task. Furthermore, the planner also pulls back a second MDS robot that was previously assigned to the risky Bomb Surveillance activity, since we require that at least two MDS robots be operational at all times. However, even this does not reduce the team risk sufficiently, and therefore a UAV performing an Area Search task is idled.

3. **Two MDS Failures**: With the failure of two MDS robots, the joint constraint that at least two MDS robots must be operational is violated. The remaining team thus does not have any feasible actions remaining, and is prevented from performing any more activities. We discuss ways in which this behavior can be modified by changing the joint constraint penalty JC.

4. **UAV, MDS Failures**: When a single UAV and a single MDS both fail, the team size is close to the minimum allowed ($M_{min} = 7$). The planner therefore removes one UAV from operation, and re-assigns one of the remaining MDS robots from a Bomb Surveillance task to a Victim Assessment task.

5. **New Activity Created**: When the operator creates a new activity, the planner has to re-assign agents. Thus the agents have to re-negotiate their risk distribution. In our case, we add a second Bomb Surveillance task, and observe that since one of the agents needs to take on more risk to perform that task, the risk available to other agents is reduced. This leads the planner to idle two UGVs, and assign an MDS to the new Bomb Surveillance task.

6.3 Results

In the results to follow, a team of ten agents begins by executing a series of tasks initially created by the operator. There is an Area Search task that four UAVs are currently assigned to, a Bomb Surveillance task that an MDS robot is assigned to, and seven Victim Assessment tasks, five of which have been assigned to three UGVs and two MDS robots. Table 6.3 shows the initial assignment. Highest priority is given to the Bomb Surveillance task with a priority of 5, the Victim Assessment tasks all have a priority of 2, and the Area Search task has a priority of 1. Under this initial assignment, the risk of violating constraint C_1 (the $M_{min} \geq 7$ constraint) is 0.0817, and the risk of violating C_2 (the constraint that the minimum number of MDS robots must be 1) is 0.0729. The total probability of violating $C_1 \cup C_2$ is 0.1484, which is less than $\alpha = 0.15$. Thus the initial assignment is feasible. We then simulate the set of failures listed above, and observe the planner's behavior in response.

Table 6.3: Initial agent assignment

Agent	Task
UAV 1	Area Search
UAV 2	Area Search
UAV 3	Area Search
UAV 4	Area Search
UGV 1	Victim Assessment
UGV 2	Victim Assessment
UGV 3	Victim Assessment
MDS 1	Victim Assessment
MDS 2	Victim Assessment
MDS 3	Bomb Surveillance

6.3.1 Case 1: Single UAV Failure

When a single UAV fails, the existing plan becomes infeasible because there are now only nine agents and the probability of any constraint ($P(C_1 \cup C_2)$) increases to 0.1860. The increase is due to the greater likelihood that more than three agents will eventually fail, given that one agent has already failed. Thus the planner now

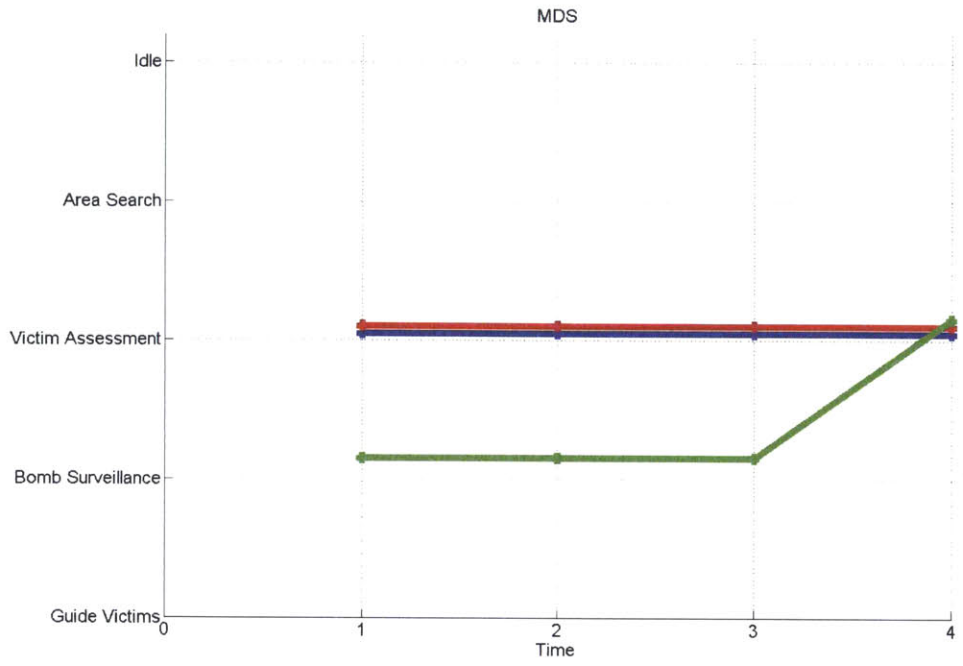

Figure 6-4: Response of the MDS robots to a single UAV failure

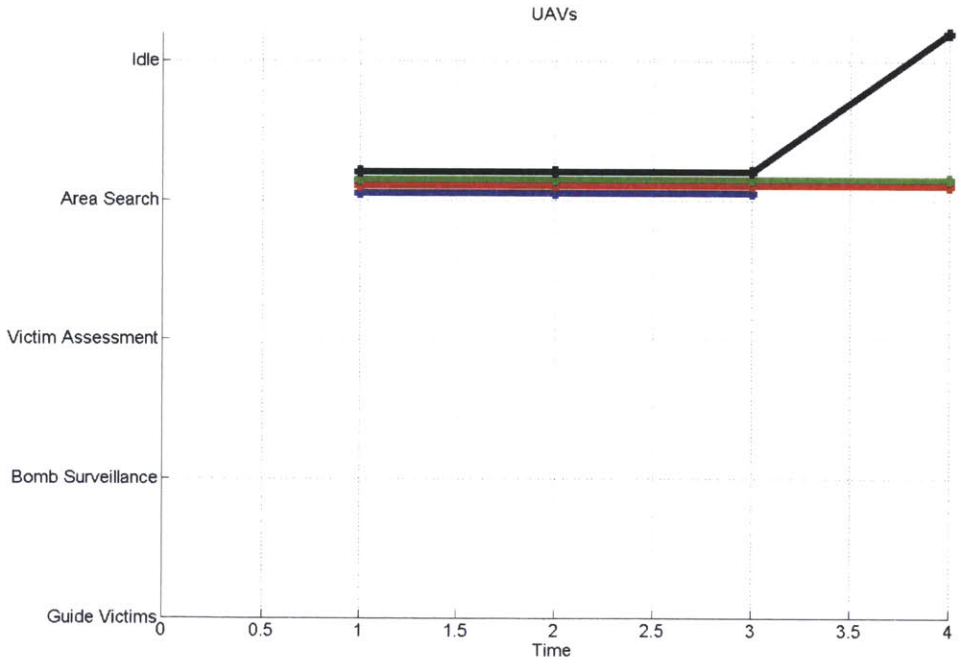

Figure 6-5: Response of the UAVs to a single UAV failure

re-assigns agents in such a way that the team remains constraint-feasible. This requires two changes - first, as shown in Figure 6-4 the MDS robot performing Bomb Surveillance is re-assigned to perform Victim Assessment - a less risky task. However, even this single change only brings the overall team risk to 0.1626, still slightly higher than $\alpha = 0.15$. In order to reduce risk further, the planner also idles an additional UAV (Figure 6-5), reducing its probability of failure to zero. This brings the overall risk down to 0.1189. The reduction could also have been achieved by idling one of the MDS robots or UGVs, but since Area Search has lower priority than the Victim Assessment tasks the planner prefers to idle a UAV.

Table 6.4: Agent re-assignment after a single UAV failure

Agent	Before Failure	After Failure
UAV 1	Area Search	Failed
UAV 2	Area Search	Idle
UAV 3	Area Search	Area Search
UAV 4	Area Search	Area Search
UGV 1	Victim Assessment	Victim Assessment
UGV 2	Victim Assessment	Victim Assessment
UGV 3	Victim Assessment	Victim Assessment
MDS 1	Victim Assessment	Victim Assessment
MDS 2	Victim Assessment	Victim Assessment
MDS 3	Bomb Surveillance	Victim Assessment

6.3.2 Case 2: Single MDS Failure

When a single MDS robot assigned to the Victim Assessment task fails, the overall risk increases to 0.3231, making the starting assignment infeasible. The increase is primarily due to a large increase in the probability that constraint C_2 (that there must be at least two live MDS robots) will be violated - the risk drastically increases from 0.0729 to 0.3250. The probability of violating C_1 (that there must be at least seven live agents) also increases slightly to 0.0974. In order to account for this failure, one of the MDS robots is switched from the risky Bomb Surveillance task to a Victim Assessment task (Figure 6-7). However this only brings the overall risk down to 0.1932, still higher than α. To achieve constraint feasibility, the planner also idles a UAV (Figure 6-6), bringing the total risk to 0.1189.

Table 6.5: Agent re-assignment after a single MDS failure

Agent	Before Failure	After Failure
UAV 1	Area Search	Idle
UAV 2	Area Search	Area Search
UAV 3	Area Search	Area Search
UAV 4	Area Search	Area Search
UGV 1	Victim Assessment	Victim Assessment
UGV 2	Victim Assessment	Victim Assessment
UGV 3	Victim Assessment	Victim Assessment
MDS 1	Victim Assessment	Victim Assessment
MDS 2	Victim Assessment	Failed
MDS 3	Bomb Surveillance	Victim Assessment

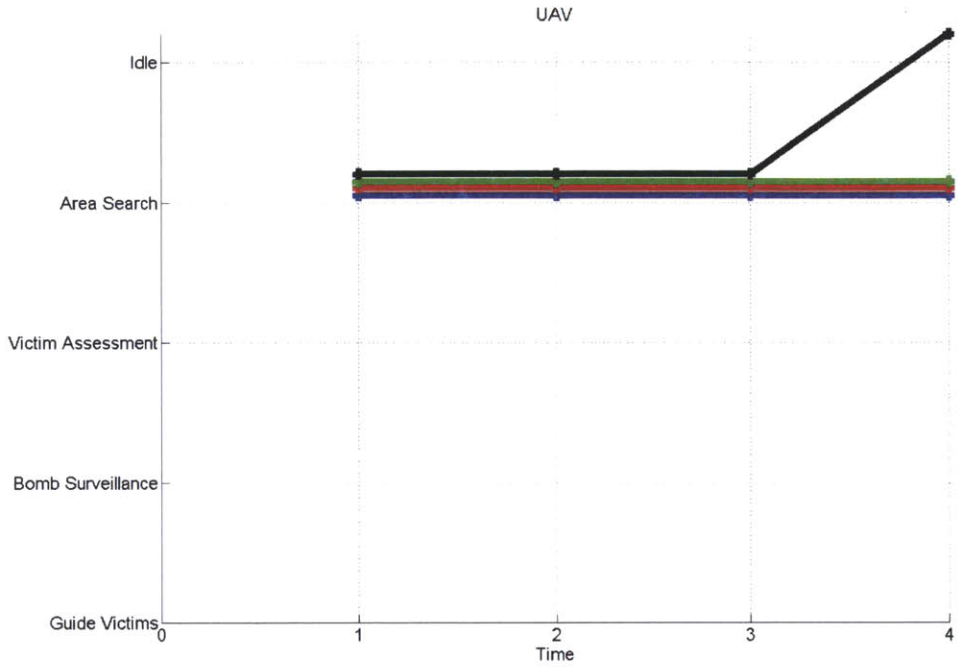

Figure 6-6: Response of the UAVs to a single MDS failure

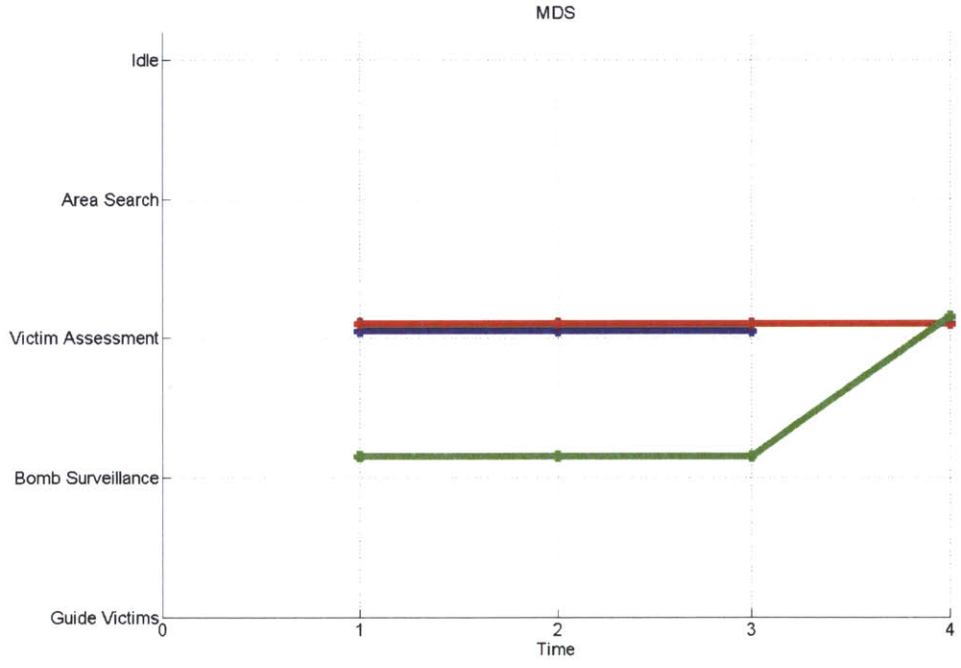

Figure 6-7: Response of the MDS robots to a single MDS failure

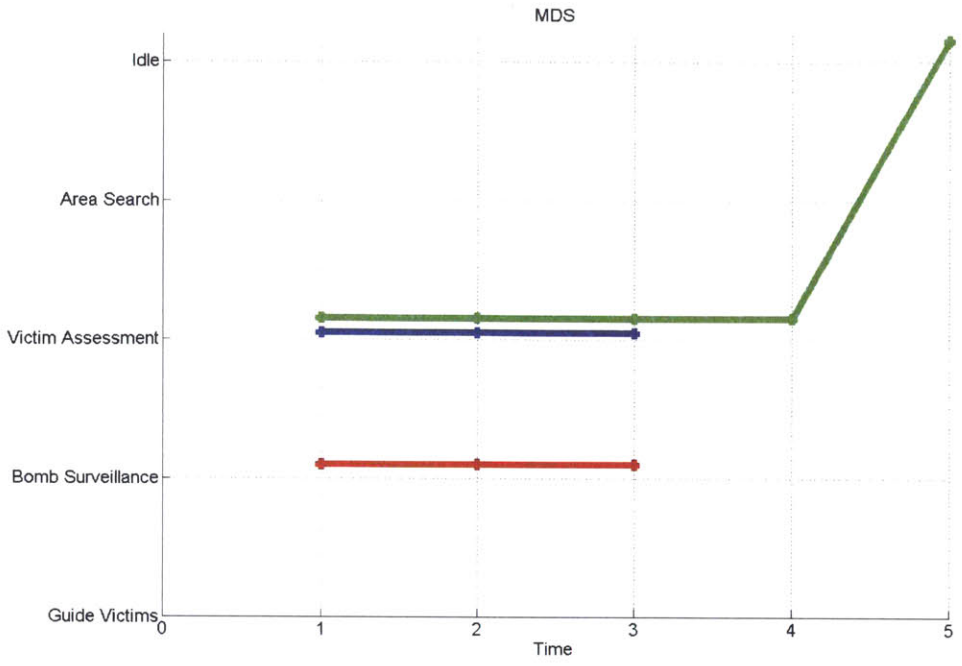

Figure 6-8: Response of the planner to two MDS failures

6.3.3 Case 3: Two MDS Failures

The case where two MDS robots fail is one where constraint C_2 has already been violated. Thus the risk associated with C_2 is now 1, and there are no constraint-feasible actions remaining for the live agents. Recovery from such a situation is impossible without some intervention. Specifically, the constraint model has to be modified to remove constraint C_2, or the penalty associated with violating the constraint, given by JC, should be lowered from $JC = 1$ to $JC < 1$ so that the remaining agents can continue operating. Setting $JC = 0$ would effectively remove the constraint, in which case the planner would re-assign the UAVs to the Area Search task, the UGVs to Victim Assessment tasks, and the single remaining MDS robot to the Bomb Surveillance task for a total risk of 0.1340. This assignment is shown in Figure 6-8. Thus the human operator of the team is provided with the flexibility to adapt to a catastrophic failure.

Table 6.6: Agent re-assignment after two MDS failures

Agent	Before Failure	After Failure	After Failure, No C_2
UAV 1	Area Search	Idle	Area Search
UAV 2	Area Search	Idle	Area Search
UAV 3	Area Search	Idle	Area Search
UAV 4	Area Search	Idle	Area Search
UGV 1	Victim Assessment	Idle	Victim Assessment
UGV 2	Victim Assessment	Idle	Victim Assessment
UGV 3	Victim Assessment	Idle	Victim Assessment
MDS 1	Victim Assessment	Idle	Bomb Surveillance
MDS 2	Victim Assessment	Failed	Failed
MDS 3	Bomb Surveillance	Failed	Failed

6.3.4 Case 4: UAV, MDS Failure

In the case of a UAV and MDS failure, no constraint has yet been violated. Hence the remaining agents continue to operate with modified assignments. The probability of a constraint being violated after a UAV and MDS failure is 0.2998, which is significantly higher than α. Thus the planner removes an MDS robot from operation. Furthermore, the one MDS robot that was performing a Bomb Surveillance task is now switched to a safer Victim Assessment task (Figure 6-10). However these measures reduce the risk only to 0.1664 which is still greater than α, and the planner has to idle yet another agent. Thus a UAV is also removed from the Area Search task (Figure 6-9).

It is interesting to note that the failure of one of the MDS robots on the Victim Assessment task causes three re-assignments (a UAV and MDS are idled, and a second MDS is switched to the Victim Assessment task). However, when the MDS robot assigned to the Bomb Surveillance task fails, only two re-assignments are required (a UAV and MDS are idled). Thus the failure of the MDS during Victim Assessment "disturbs" the original plan more than the failure during Bomb Surveillance. Intuitively, this is because it was known *a priori* that the Bomb Surveillance task was risky. The planner therefore had already accounted for a likely failure. The Victim Assessment task, on the other hand, was assumed to be safer and a failure is more unexpected. The original plan therefore requires a greater modification.

The same behavior can be noticed if we let two UAVs fail during the Area Search task, which was *a priori* assumed to be a very safe task. The probability of violating constraint C_1 (that there must be at least seven live agents at any time) greatly increases to 0.2048, and a large number of agents must be idled to ensure constraint feasibility. On the other hand, if two UGVs had failed, the risk increases to only 0.1348, and the team has more options. The failure of an agent that was assumed to be relatively safe is much more difficult to adapt to, than the failure of an agent that was known to be undertaking a risky task.

Table 6.7: Agent re-assignment after a UAV and MDS failure

Agent	Before Failure	After Failure
UAV 1	Area Search	Failed
UAV 2	Area Search	Idle
UAV 3	Area Search	Area Search
UAV 4	Area Search	Area Search
UGV 1	Victim Assessment	Victim Assessment
UGV 2	Victim Assessment	Victim Assessment
UGV 3	Victim Assessment	Victim Assessment
MDS 1	Victim Assessment	Failed
MDS 2	Victim Assessment	Victim Assessment
MDS 3	Bomb Surveillance	Victim Assessment

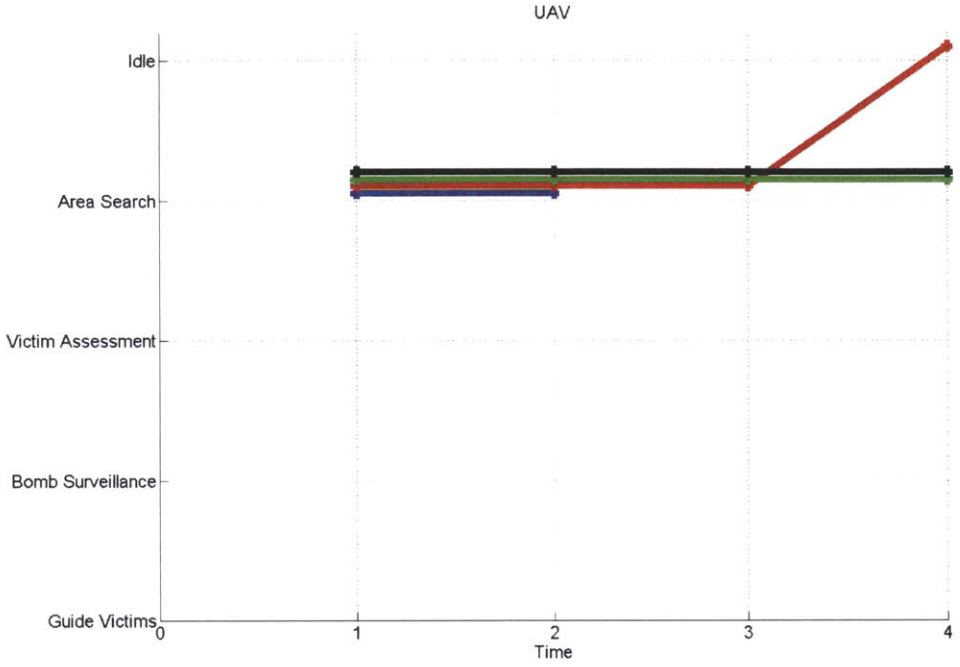

Figure 6-9: Response of the UAVs to a UAV and MDS failure

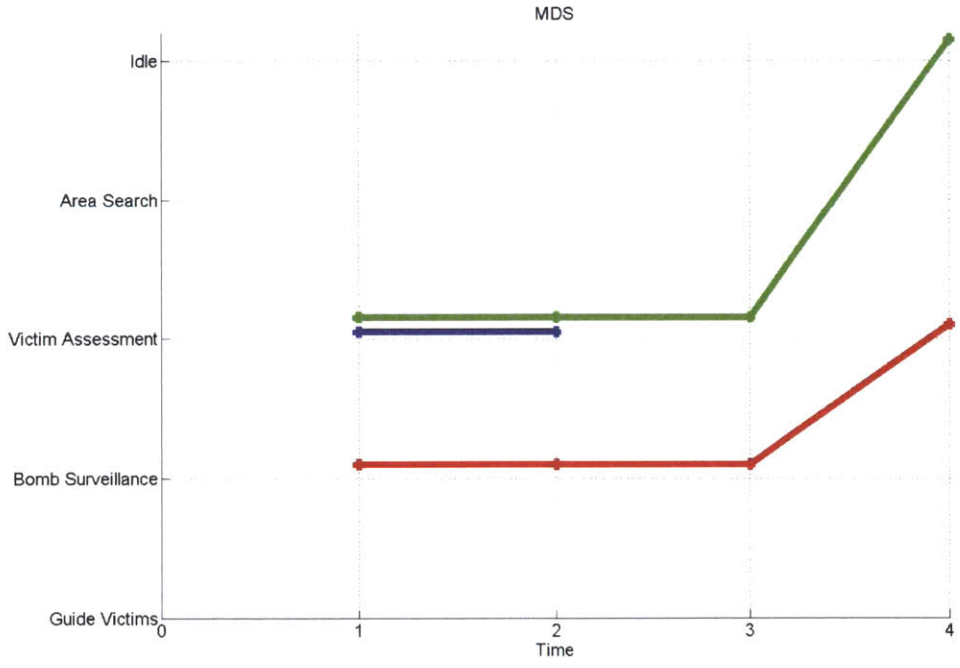

Figure 6-10: Response of the MDS robots to a UAV and MDS failure

6.3.5 Case 5: New Activity

When a new activity is created, the agents have to decide whether to undertake the new activity, and if so which agent. In this case, we add a Bomb Surveillance task and give it high priority, so that the reward for being assigned to that task is very high. However, a Bomb Surveillance task is high-risk and therefore requires that an agent previously performing a relatively low-risk task take on more risk. In order to keep the team risk bounded, some other agents will have to take less risk.

Table 6.8: Agent re-assignment after a UAV and MDS failure

Agent	Before Failure	After Failure
UAV 1	Area Search	Area Search
UAV 2	Area Search	Area Search
UAV 3	Area Search	Area Search
UAV 4	Area Search	Area Search
UGV 1	Victim Assessment	Idle
UGV 2	Victim Assessment	Idle
UGV 3	Victim Assessment	Victim Assessment
MDS 1	Victim Assessment	Victim Assessment
MDS 2	Victim Assessment	Bomb Surveillance
MDS 3	Bomb Surveillance	Bomb Surveillance

The outcome of adding a Bomb Surveillance task is shown in Table 6.8. First, since the Bomb Surveillance task has high priority, an MDS robot is re-assigned from a Victim Assessment task to the new Bomb Surveillance task. However, this requires the MDS robot to take more risk, from 0.1 to 0.25. In order to keep the team risk bounded and less than α, some agents must take less risk. This is done by idling two UGVs that were previously assigned to the Victim Assessment task, therefore lowering their risk from 0.1 to 0. With this re-assignment, the team risk is kept at 0.1390, which is less than $\alpha = 0.15$.

It is interesting to note that even a high-risk task, given sufficiently high priority (i.e. high reward) will be assigned to an agent. For instance, suppose the new Bomb Surveillance task had a risk of 1. In that case, given sufficiently high priority, all other MDS robots and one UGV would be idled to give a total team risk of less than α (in

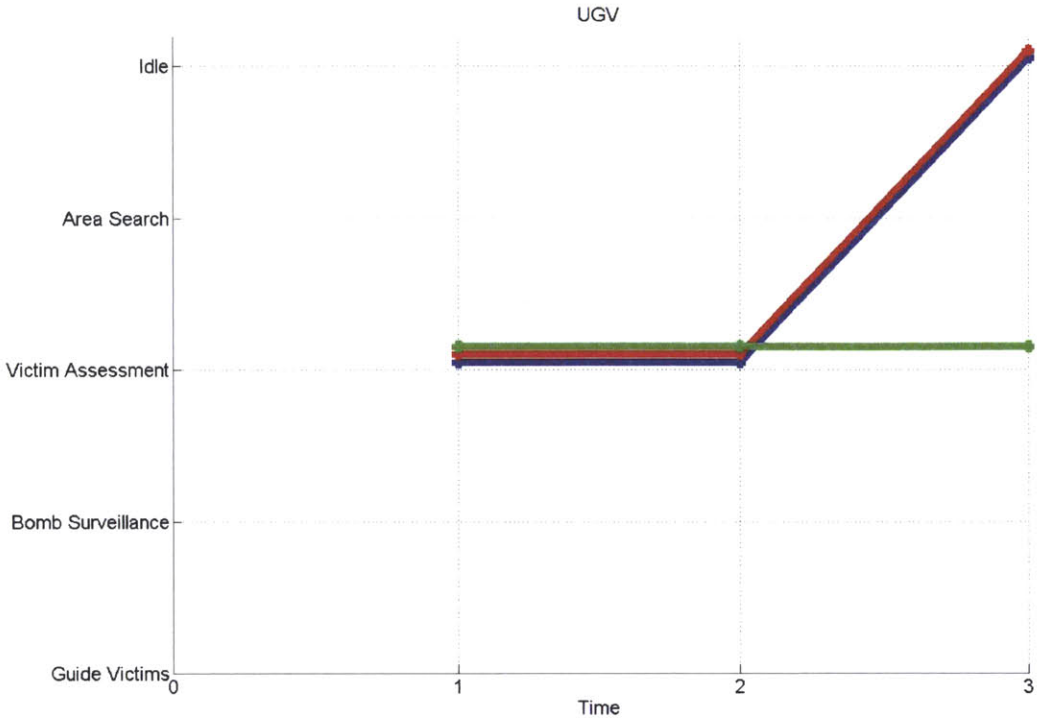

Figure 6-11: Response of the UGVs to the appearance of a new Bomb Surveillance task

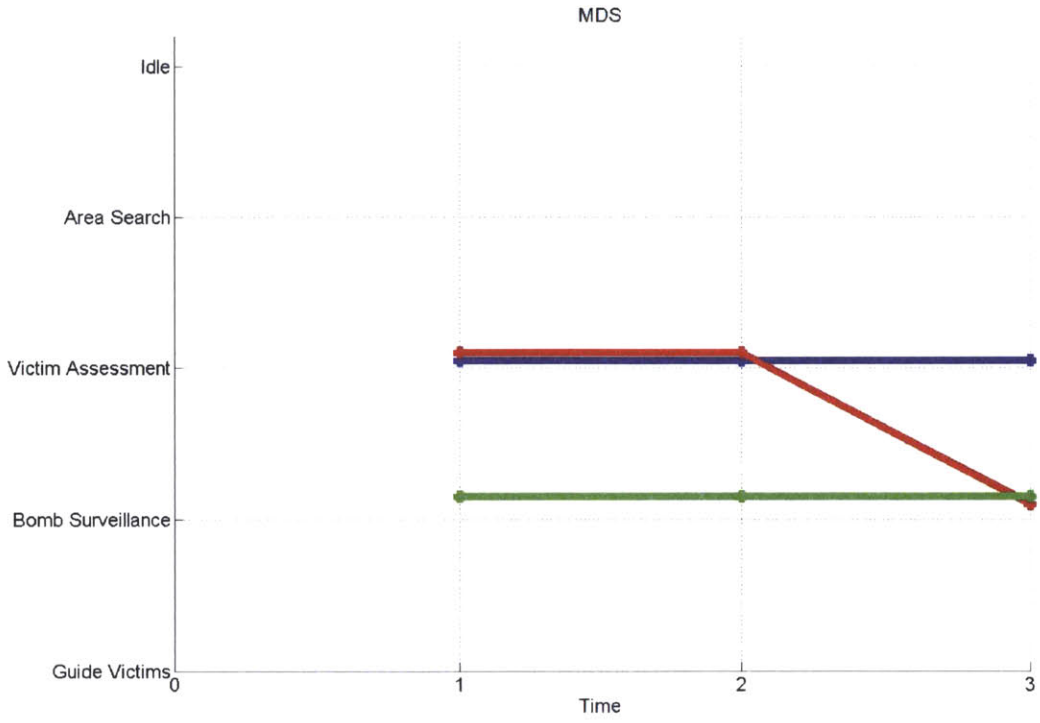

Figure 6-12: Response of the MDS robots to the appearance of a new Bomb Surveillance task

fact, nearly zero). Thus a high priority task, even if practically suicidal in terms of risk, will be assigned to an agent as long as there are sufficient agents remaining to satisfy the team constraints.

6.4 Summary

In this chapter, the methods developed in the previous chapters were applied to a realistic CBRNE operational environment for a significant team size of ten agents, composed of UAVs, UGVs and human-like MDS robots. The only constraints imposed were joint constraints, and the joint risk was kept bounded. The performance of the planner in the presence of agent failures, constraint violations, and environmental changes (creation of a new task) were investigated. The impact of the joint constraint penalty JC, and the manner in which an operator can change that penalty to achieve various behaviors was discussed. Finally, the balancing of risk and reward in such a heterogeneous, multi-agent team was shown to lead to team behaviors that ensure that high-priority tasks get done, even if it means individual agents taking significant risk.

Chapter 7

Conclusions and Future Work

This work seeks to solve the problem of planning under uncertainty and constraints for multi-agent teams of unmanned vehicles. The presence of both constraints and uncertainty introduces the possibility that constraints might be violated - a probability that we define in this work as *risk*. We then seek to generate policies for the agents that maximize the performance, as measured by a reward function, while keeping the risk bounded to a finite, acceptable and operator-defined value.

This planning problem as a Constrained Markov Decision Process (C-MDP). We choose the framework of C-MDPs for several reasons. First, discrete MDPs naturally incorporate uncertainty and noise and are particularly convenient when the noise is not Gaussian and the system dynamics are not linear. Second, MDPs provide a means by which to encode the mission objectives in the form of a reward model. There are no restrictions on the form and nature of the reward model - the model can be nonlinear, continuous or discrete, convex or non-convex. Third, MDPs can be extended to include the *partially observable* case, and that is one of the areas this work will investigate. Partially Observable Markov Decision Processes (POMDPs) are capable of deciding when to take actions that improve knowledge of the world and when to act upon information that is already available.

The flexibility and power of MDPs comes at the price of computational complexity. MDPs suffer from "the curse of dimensionality" [63], i.e. the computational effort required to solve an MDP grows dramatically as the size of the problem state space

increases. This is especially true in the case of multi-agent systems since the size of the problem is directly related to the number of agents in the team. This work seeks to address the issues of computational complexity and scaling to multi-agent teams through the contributions listed below.

Contribution 1: Planning Under Uncertainty and Constraints This work began by proposing an extension to the standard MDP. The extension is the *constraint model* and provides a means by which to incorporate any general constraints into an MDP, thus turning it into a C-MDP. A fast, computationally efficient online algorithm was then presented to solve the C-MDP. This algorithm relied on a finite horizon forward search to optimize the reward and an offline approximate solution to estimate constraint feasibility beyond the search horizon. It was shown that this algorithm achieves good performance in constrained environments. However, the offline approximate solution becomes a computational bottleneck, and this is particularly acute when the environment and the constraint map are not static. The complexity of this offline approximate solution grows exponentially in the number of agents, and therefore becomes important when we attempt to extend this algorithm to large teams. This directly motivated the third contribution. But before addressing the multi-agent problem, we extended the algorithm to continuous domains.

Contribution 2: Planning Under Uncertainty and Constraints in Continuous Domains Solving MDPs defined over continuous domains (continuous state and action spaces) is in general a difficult problem. A solution to an MDP is a *policy* - a mapping from states to actions. In other words, a policy prescribes which action to take in every state. In discrete MDPs, this mapping can be expressed (conceptually, if not actually) as a look-up table. In continuous MDPs, representing policies is significantly more challenging. One approach is to express the policy as the gradient of the *value function*, which is the cost-to-go from any state. However, solving for the value function is just as computationally challenging as solving for the policy itself. One method that has been used successfully in the literature [64] [65] [66] for finding

the value function (which can be applied to both discrete and continuous domains) is *function approximation*. Function approximation finds the value function by assuming it to be a linear combination of a set of *basis functions*. These basis functions, also known as *features*, are picked by the designer to be appropriate for the specific problem of interest. In this work, we extended MDP function approximation techniques for constrained continuous MDPs. We showed that when the underlying system is continuous, approximating continuous domains by discretization can yield poor results even for reasonable discretizations. On the other hand, function approximation with the right set of features achieves good performance.

Contribution 3: Planning for Multiple Agents Under Uncertainty and Constraints The size of an MDP for multi-agent teams grows exponentially in the number of agents. This "curse of dimensionality" makes the use of MDPs for teams of agents computationally difficult. However, there are some significant simplifications that can be made under sensible assumptions. One key assumption we made was to assume *transition independence*, i.e. the action of one agent only affects its own dynamics, and not those of other agents. While this is not strictly true (the action of one agent is connected to the actions of other agents at the high-level planning stage) it is a good approximation. However we do need to account for the fact that the agents are still coupled through rewards and constraints. Some rewards may require a joint action by more than one agent, and some constraints might be applicable to the entire team rather than individual agents. Specifically this work investigates the case where there is constraint coupling and proposes a mechanism by which agents can plan their own individual actions while accounting for their impact on the team constraints. In planning for themselves, the agents must make some assumptions about the actions of the other agents. Specifically, they have to assume that the probability of the other agents violating a team constraint (henceforth referred to as *joint constraints*), defined as the *risk*, is fixed. This assumption introduced some conservatism in the agents' behavior, and in the next contribution we eliminated some of that conservatism through team communication and consensus.

Contribution 4: Distributed Planning Under Uncertainty and Constraints The final contribution of this work looked at the benefit of having team communication and consensus in the presence of joint constraints. As mentioned previously, agents can plan their own individual actions even in the presence of constraint coupling provided they make some assumptions about the actions of other agents. Specifically, they have to assume the other agents will not take actions that exceed a certain threshold of risk. In this section, we removed that assumption and instead allowed the agents to communicate with each other and arrive at a consensus about how much risk each agent can take. This risk negotiation can take place throughout the mission, so that as the environment changes and some agents have to take more (or less) risk than was originally foreseen, the team can adapt. The properties and complexity of this risk negotiation were discussed, and it was shown that significantly higher rewards can be expected particularly in highly constrained environments.

7.1 Future Work

The work done in this thesis explores the area of planning under uncertainty and constraints, and the scope for further work in this area is significant. Some extensions of the work presented here are suggested below.

7.1.1 Approximate Dynamic Programming

We have already seen that one key component in the online algorithm proposed in Chapter 3 is solving for the offline minimum risk policy, which is obtained by solving the following unconstrained MDP.

$$U_{Ci}(\mathbf{s}_i) = \min_{\pi_i} E\left[\sum_{t=0}^{T} C_i(\mathbf{s}_{it}, \pi_i(\mathbf{s}_{it}))\right] + \sum_{j=0}^{E} JC_j \prod_{k=0}^{N} p_{kj}(\pi_k) \tag{7.1}$$

In the work presented here, value iteration was used to solve this unconstrained MDP. However, significant work has been done in the field of approximate dynamic programming, for instance by Bethke and How [37], Geramifard, Doshi, Roy and How[107],

Powell [64], and Bertsekas. These methods can be used to generate approximate policies efficiently, increasing the speed of the offline computations and allowing the methods presented here to be expanded to even more complex problems.

7.1.2 Continuous Domains

While the work done here has been extended to continuous domains for a single agent, the extension to continuous domains with multiple agents has not been considered here. Transition independence in continuous domains has been addressed previously, for instance by Yin, Rajan, and Tambe [92]. Definitions used in discrete domains, such as the definitions of "events" will need to be revised to be appropriate for continuous domains. Furthermore, the risk negotiation algorithm would have to be modified to include a modified termination condition, since the total number of possible policies is infinite in continuous domains, and hence it is unlikely that the proposed algorithm without modifications would terminate in a reasonable time.

7.1.3 Asynchronous Networks

The risk negotiation algorithm presented in Chapter 6 implicitly assumes that the agents are able to communicate synchronously. However, it is possible that some communications links between agents have time delays that make it difficult to carry out certain parts of the algorithm. For instance Stage 1 of the algorithm requires the agents to arrive at a consensus. In the absence of reliable synchronous communication, this consensus might be difficult to carry out. Thus modifications to the algorithm to account for the absence of reliable synchronous communications are a potentially useful area of work.

7.1.4 Operator Interface

Although beyond the scope of this thesis, the design of an operator interface for a human to interact with the planning system is a major challenge. Some of the features that such an interface must have were previously discussed, in Chapter 6.

Of the several parameters that human operator is required to provide, one is the risk associated with each task. The risk is a probability, and it is unlikely that a human operator will be able to produce a reasonable estimate. A more viable solution is for the interface to provide the operator with a visual representation of risk - for instance a slider bar indicating the risk level. The operator would manipulate this visual representation to set the risk. The design, calibration and implementation of such visual representations is a significant challenge in the human-computer interaction community and must be addressed before the planning algorithms discussed here can be made operational.

Appendix A

Computational Complexity

In this section, we investigate the computational implications of assuming transition independence in problems where transition independence is a poor assumption. We show that the worst-case computational complexity of the decoupled MDPs approaches that of solving the single large MDP, and in the case where transition independence does not exist (i.e. every single state-action-state transition for every agent depends on the states and actions of other agents) the complexity is exactly the same as solving the complete MDP.

The computational complexity of a single iteration of MDP value iteration is given by $|S|^2|A|$, where $|S|$ is the size of the state space, $|A|$ is the size of the action space. This is due to the fact that in the worst-case, in each iteration MDP value iteration must loop through all possible actions and successor states to compute the updated value function. Thus the worst-case complexity for a team of N agents, whose joint state space is given by $S = S_1 \times S_2 \cdots S_N$ and whose joint action space is given by $A = A_1 \times A_2 \cdots A_N$ is given by $\prod_{i=1}^{N} |S_i|^2|A_i|$. Assuming $|S_1| = |S_2| = \cdots |S_N| = |S_i|$, and $|A_1| = |A_2| = \cdots |A_N| = |A_i|$, we can simplify the complexity to $|S_i|^{2N}|A_i|^N$. By decomposing the single MDP problem into several individual MDPs, as in Chapter 4 and as in Becker, Zilberstein, Lesser and Goldman [90], we convert the problem into one of solving several MDPs of size $|S_i| \times |A_i|$, thus reducing the complexity to $|S_i|^2|A_i|$. However by introducing events (See Chapter 4) we require value iteration to also loop through all possible events to check to see if a particular state-action-

state triple is part of an event. This increases the complexity to $|S_i|^2|A_i|E$. As long as $|S_i|^2|A_i|E \ll |S_i|^{2N}|A_i|^N$, i.e. $E \ll |S_i|^{2N-2}|A_i|^{N-1}$, the computational gain of decomposing the MDP will be significant.

Now consider the case where transition independence is not a correct assumption, i.e. the agents are in fact closely coupled. To capture this coupling behavior, one needs to define a very large number of events. Specifically, *every* possible state-action-state triple for *all other agents* will have to be considered as a joint event. Since each agent has $|S_i|^2|A_i|$ state-action-state triples, the total number of joint events for each agent will be $E = |S_i|^{2(N-1)}|A_i|^{N-1}$. Clearly the condition $E \ll |S_i|^{2N-2}|A_i|^{N-1}$ is no longer true (in fact $E = |S_i|^{2N-2}|A_i|^{N-1}$) and hence there is no computational gain from decoupling. Note that the total computational complexity in this case is the *same* as for value iteration. This is easily verified by substituting $E = |S_i|^{2N-2}|A_i|^{N-1}$ into the relation for computational complexity $|S_i|^2|A_i|E$, which yields the expression for the worst-case MDP value iteration, $|S_i|^{2N}|A_i|^N$.

Bibliography

[1] R. V. Welch and G. O. Edmonds, "Applying robotics to HAZMAT," in *The Fourth National Technology Transfer Conference and Exposition*, vol. 2, pp. 279–287, 2003.

[2] A. Hemmerly-Brown, "CBRNE Command Zeros in on Ever-Changing Threats." `http://www.cbrne.army.mil/cbrne%20defense%20summit.htm`, May 2010.

[3] P. Morgan, "This is not a game: Fukushima robots operated by xbox 360 controllers." Discover Magazine (Electronic) 2011, April 2011.

[4] M. Yamaguchi, "Japan sends robots in to stricken nuclear plant." Associated Press, 2011, April 2011.

[5] "Robots Find Radiation Still Too Deadly for Humans to Enter two Japan Reactor Units." Associated Press, 2011, April 2011.

[6] B. A. Jackson, J. C. Baker, M. S. Ridgely, J. T. Bartis, and H. I. Linn, *Protecting Emergency Responders: Safety Management in Disaster and Terrorism Response*. Department of Health and Human Services, Centers for Disease Control and Prevention, National Institute for Occupational Safety and Health.

[7] D. Brittain, *Hong Kong's Reponse to a Chemical, Biological, Radiological or Nuclear Attack*. Hong Kong Hospital Authority, April 2010.

[8] L. Greenemeier, "Robots Arrive at Fukushima Nuclear Site with Unclear Mission." Electronic, March 2011. `http://www.scientificamerican.com/article.cfm?id=robots-arrive-fukushima-nuclear&page=2`.

[9] P. Jaillet, *Probabilistic Travelling Salesman Problems*. PhD thesis, Massachusetts Institute of Technology, 1985.

[10] A. M. Campbell and B. W. Thomas, "Probabilistic travelling salesman problems with deadlines," *Transportation Science*, vol. 42, pp. 1–21, February 2008.

[11] M. Gendreau, G. Laporte, and R. Seguin, "Stochastic vehicle routing," *European Journal of Operational Research*, vol. 88, pp. 3–12, 1996.

[12] M. Gendreau, G. Laporte, and R. Seguin, "An exact algorithm for the vehicle routing problem with stochastic demands and customers," *Transportation Science*, vol. 29, no. 2, pp. 143–155, 1995.

[13] M. Dror, G. Laporte, and P. Trudeau, "Vehicle routing with stochastic demands: Properties and solution frameworks," *Transportation Science*, vol. 23, pp. 166–176, August 1989.

[14] C. D. J. Waters, "Vehicle-scheduling problems with uncertainty and omitted customers," *Journal of the Operational Research Society*, vol. 40, pp. 1099–1108, 1989.

[15] G. Laporte, F. V. Louveaux, and H. Mercure, "The vehicle routing problem with stochastic travel times," *Transportation Science*, vol. 26, pp. 161–170, August 1992.

[16] D. Bertsimas, *Probabilistic Combinatorial Optimization Problems.* PhD thesis, Massachusetts Institute of Technology, 1988.

[17] M. Pavone, N. Bisnik, E. Frazzoli, and V. Isler, "Decentralized vehicle routing in a stochastic and dynamic environment with customer impatience," in *ACM International Conference Proceedings*, vol. 318, 2007.

[18] M. Pavone, E. Frazzoli, and F. Bullo, "Decentralized algorithms for stochastic and dynamic vehicle routing with general demand distribution," in *Proceedings of the 46th Conference on Decision and Control*, December 2007.

[19] M. Alighanbari, *Robust and Decentralized Task Assignment Algorithms for UAVs.* PhD thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, Cambridge MA, September 2007.

[20] M. Alighanbari, Y. Kuwata, and J. P. How, "Coordination and control of multiple UAVs with timing constraints and loitering," in *American Control Conference (ACC)*, vol. 6, pp. 5311–5316 vol.6, 4-6 June 2003.

[21] M. Alighanbari and J. P. How, "A robust approach to the UAV task assignment problem," *International Journal of Robust and Nonlinear Control*, vol. 18, pp. 118–134, January 2008.

[22] L. F. Bertuccelli, *Robust Decision-Making with Model Uncertainty in Aerospace Systems.* PhD thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, Cambridge MA, September 2008.

[23] K. Ozbay, W. Xiao, C. Iyigun, and M. Baykal-Gursoy, "Probabilistic programming models for response vehicle dispatching and resource allocation in traffic incident management," I&SE-Working Paper 04-014,, Industrial and Systems Engineering Department, Rutgers University, 2007.

[24] E. Gagon, C. A. Rabbath, and M. Lauzon, "Receding horizons with heading constraints and collision avoidance," in *AIAA Guidance, Navigation, and Control Conference (GNC)*, August 2005.

[25] D. A. Castanon and J. M. Wohletz, "Model predictive control for stochastic resource allocation," *IEEE Transactions on Automatic Control*, vol. 54, no. 8, pp. 1739 – 1750, 2009.

[26] C. Schumacher, P. R. Chandler, and S. R. Rasmussen, "Task allocation for wide area search munitions via network flow optimization," in *AIAA Guidance, Navigation, and Control Conference (GNC)*, August 2001.

[27] P. B. Sujit, J. M. George, and R. Beard, "Multiple mav task allocation using distributed auctions," in *AIAA Guidance, Navigation and Control Conference and Exhibit*, August 2007.

[28] D. P. Bertsekas, "A distributed algorithm for the assignment problem." Laboratory for Information and Decision Systems Unpublished Report, MIT.

[29] D. Bertsekas and D. A. Castanon, "The auction algorithm for the transportation problem," tech. rep., Laboratory for Information and Decision Systems, MIT, 1989.

[30] D. P. Bertsekas, "The auction algorithm for assignment and other network flow problems," tech. rep., MIT, 1989.

[31] D. P. Bertsekas and D. A. Castanon, "Parallel synchronous and asynchronous implementations of the auction algorithm," *Parallel Computing*, vol. 17, pp. 707–732, 1991.

[32] "Auction mechanism design for multi-robot coordination," in *In Proc. 17th Annual Conf. on Neural Information Processing Systems*, 2003.

[33] H.-L. Choi, L. Brunet, and J. P. How, "Consensus-based decentralized auctions for robust task allocation," *IEEE Transactions on Robotics*, vol. 25, pp. 912–926, August 2009.

[34] L. B. Johnson, S. Ponda, H.-L. Choi, and J. P. How, "Improving the efficiency of a decentralized tasking algorithm for UAV teams with asynchronous communications," in *AIAA Guidance, Navigation, and Control Conference (GNC)*, August 2010 (AIAA-2010-8421).

[35] M. L. Puterman, *Markov Decision Processes: Stochastic Dynamic Programming.* John Wiley and Sons, 1994.

[36] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, vol. I–II. PO Box 391, Belmont, MA 02178: Athena Scientific, 2007.

[37] B. Bethke and J. How, "Approximate Dynamic Programming Using Bellman Residual Elimination and Gaussian Process Regression," in *American Control Conference (ACC)*, (St. Louis, MO), 2009.

[38] B. Bethke and J. How and A. Ozdaglar, "Approximate Dynamic Programming Using Support Vector Regression," in *IEEE Conference on Decision and Control (CDC)*, (Cancun, Mexico), 2008.

[39] C. Boutilier and D. Poole, "Computing optimal policies for partially observable decision processes using compact representations," in *Proc. Thirteenth National Conference on Artificial Intelligence*, August 1996.

[40] M. T. J. Spaan and N. Vlassis, "Perseus: Randomized point-based value iteration for pomdps," *Journal of Artificial Intelligence Research*, vol. 3501, pp. 450–455, 2005.

[41] H. Kurniawati, D. Hsu, and W. Lee, "SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces," in *Proc. Robotics: Science and Systems*, 2008.

[42] N. Roy, *Finding Approximate POMDP Solutions Through Belief Compression.* PhD thesis, Carnegie Mellon University, Pittsburgh, Pennsylvania, 2003.

[43] N. Roy, G. Gordon, and S. Thrun, "Finding Approximate POMDP Solutions Through Belief Compression," *Journal of Artificial Intelligence Research*, vol. 23, pp. 1–40, 2005.

[44] S. Prentice and N. Roy, "The belief roadmap: Efficient planning in lienar POMDPs by factoring the covariance," in *Proceedings of the International Symposium of Robotics Research*, (Hiroshima, Japan), November 2007.

[45] C. Amato, D. S. Bernstein, and S. Zilberstein, "Solving pomdps using quadratically constrained linear programs," in *International Joint Conference on Artificial Intelligence*, 2007.

[46] J. D. Isom, S. P. Meyn, and R. D. Braatz, "Piecewise linear dynamic programming for constrained pomdps," in *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence* (A. Cohn, ed.), vol. 1, pp. 291–296, AAAI Press, 2008.

[47] S. Paquet, L. Tobin, and B. Chaib-draa, "Real-time decision making for large pomdps," *Advances in Artificial Intelligence*, vol. 3501, pp. 450–455, 2005.

[48] D. P. Bertsekas and D. A. Castanon, "Rollout algorithms for stochastic scheduling problems," *Journal of Heuristics*, vol. 5, pp. 89–108, 1999.

[49] S. Ross, J. Pineau, S. Paquet, and B. Chaib-draa, "Online planning algorithms for pomdps," *Journal of Artificial Intelligence Research*, vol. 32, pp. 663–704, 2008.

[50] E. Altman, *Constrained Markov Decision Processes*. Chapman and Hall/CRC, 1999.

[51] E. Altman, "Denumerable Constrained Markov Decision Processes and Finite Approximations," *Mathematics of Operations Research*, vol. 19, p. 169, 1994.

[52] R. C. Chen and G. L. Blankenship, "Dynamic programming equations for constrained stochastic control," *Proceedings of the 2002 American Control Conference*, vol. 3, pp. 2014–2022, 2002.

[53] R. C. Chen and G. L. Blankenship, "Dynamic programming equations for discounted constrained stochastic control," *IEEE Transactions on Automatic Control*, vol. 49, pp. 699–709, 2004.

[54] A. B. Piunovskiy and X. Mao, "Constrained Markovian Decision Processes: The Dynamic Programming Approach," *Operations Research Letters*, vol. 27, pp. 119–126, 2000.

[55] R. C. Chen and E. A. Feinberg, "Non-randomized policies for constrained markov decision processes," *Mathematical Methods of Operations Research*, vol. 66, pp. 165–179, 2007.

[56] R. C. Chen, "Constrained stochastic control and optimal search," *Proceedings of the 43rd IEEE Conference on Decision and Control*, vol. 3, pp. 3013–3020, 2004.

[57] D. Dolgov, *Integrated Resource Allocation and Planning in Stochastic Multiagent Environments*. PhD thesis, University of Michigan, 2006.

[58] D. Dolgov and E. Durfee, "Stationary deterministic policies for constrained MDPs with multiple rewards, costs and discount factors," in *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pp. 1326–1331, 2005.

[59] A. Zadorojniy and A. Shwartz, "Robustness of policies in constrained markov decision processess," *IEEE Transactions on Automatic Control*, vol. 51, 2006.

[60] P. Geibel, "Reinforcement Learning Approaches for Constrained MDPs," *International Journal of Computational Intelligence Research*, vol. 3, pp. 16–20, 2007.

[61] E. A. Feinberg and A. Shwartz, "Constrained Markov Decision Models with Weighted Discounted Rewards," *Mathematics of Operations Research*, vol. 20, pp. 302–320, 1995.

[62] Z. Gabor, Z. Kalmar, and C. Szcpcsvari, "Multi-criteria reinforcement learning," in *Proceedings of the 15th International Conference on Machine Learning* (M. Kaufmann, ed.), pp. 197–205, 1998.

[63] A. E. Bryson and Y.-C. Ho, *Applied Optimal Control*. New York: Hemisphere Publishing Corp., 1975.

[64] W. Powell, *Approximate Dynamic Programming*. John Wiley and Sons, 2007.

[65] M. Irodova and R. H. Sloan, "Reinforcement learning and function approximation," in *American Association for Artificial Intelligence*, 2005.

[66] F. S. Melo and M. I. Ribeiro, "Q-learning with linear function approximation," *Proceedings of the 20th Annual Conference on Learning Theory*, pp. 308–322, 2007.

[67] M. Ono and B. C. Williams, "An efficient motion planning algorithm for stochastic dynamic systems with constraints on probability of failure," in *Proceedings of the Twenty Third AAAI Conference on Artificial Intelligence*, 2008.

[68] M. Ono and B. C. Williams, "Iterative risk allocation: A new approach to robust model predictive control with a joint chance constraint," in *Proceedings of 47th IEEE Conference on Decision and Control*, 2008.

[69] B. Luders, M. Kothari, and J. P. How, "Chance constrained RRT for probabilistic robustness to environmental uncertainty," in *AIAA Guidance, Navigation, and Control Conference (GNC)*, (Toronto, Canada), August 2010. (AIAA-2010-8160).

[70] D. A. Dolgov and E. H. Durfee, "Approximate Probabilistic Constraints and Risk-Sensitive Optimization Criteria in Markov Decision Processes," in *Proceedings of International Symposiums on Artificial Intelligence and Mathematics*, vol. 28, 2004.

[71] R. Cavazos-Cadena and R. Montes-De-Oca, "The Value Iteration Algorithm in Risk Sensitive Average Markov Decision Chains with Finite State Space," *Mathematics of Operations Research*, vol. 28, pp. 752–776, 2003.

[72] P. Geibel and F. Wysotzki, "Risk-sensitive reinforcement learning applied to control under constraints," *Journal of Artificial Intelligence and Research*, vol. 24, pp. 81–108, 2005.

[73] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA: MIT Press, 2005.

[74] J. Pineau, G. Gordon, and S. Thrun, "Point-based value iteration: An anytime algorithm for POMDPs," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 2003.

[75] W. Zhang and T. G. Dietterich, "High-Performance Job-Shop Scheduling With A Time-Delay TD(lambda) Network," in *Advances in Neural Information Processing Systems 8*, pp. 1024–1030, MIT Press, 1995.

[76] R. S. Sutton, "Generalization in reinforcement learning: Successful examples using sparse coarse coding," in *Advances in Neural Information Processing Systems 8*, pp. 1038–1044, The MIT Press, 1996.

[77] G. Tesauro, "Programming backgammon using self-teaching neural nets," *Artificial Intelligence*, vol. 134, no. 1-2, pp. 181–199, 2002.

[78] R. Munos and A. Moore, "Variable resolution discretization in optimal control," *Mach. Learn.*, vol. 49, no. 2-3, pp. 291–323, 2002.

[79] A. Geramifard, M. Bowling, M. Zinkevich, and R. Sutton, "iLSTD: Eligibility traces and convergence analysis," in *Advances in Neural Information Processing Systems 19* (B. Schölkopf, J. Platt, and T. Hoffman, eds.), Cambridge, MA: MIT Press, 2007.

[80] J. N. Tsitsiklis and B. Van Roy, "An analysis of temporal-difference learning with function approximation," *IEEE Transactions on Automatic Control*, vol. 42, no. 5, pp. 674–690, 1997.

[81] R. S. Sutton and S. D. Whitehead, "Online learning with random representations," in *proceedings of the Tenth International Conference on Machine Learning*, pp. 314–321, 1993.

[82] D. Silver, R. S. Sutton, and M. Müller, "Sample-based learning and search with permanent and transient memories," in *ICML '08: Proceedings of the 25th international conference on Machine learning*, (New York, NY, USA), pp. 968–975, ACM, 2008.

[83] R. Parr, C. Painter-Wakefield, L. Li, and M. Littman, "Analyzing feature generation for value-function approximation," in *ICML '07: Proceedings of the 24th international conference on Machine learning*, (New York, NY, USA), pp. 737–744, ACM, 2007.

[84] M. J. Valenti, *Approximate dynamic programming with applications in multi-agent systems*. PhD thesis, M.I.T., Cambridge, MA, USA, 2007. Adviser-Farias, Daniela Pucci and Adviser-How, Jonathan P.

[85] M. Bowling and M. Veloso, "Scalable learning in stochastic games," 2002.

[86] P. Stone, R. S. Sutton, and G. Kuhlmann, "Reinforcement learning for robocup soccer keepaway," *International Society for Adaptive Behavior*, vol. 13, no. 3, pp. 165–188, 2005.

[87] G. Gordon, "Stable function approximation in dynamic programming," in *Proceedings of the Twelfth International Conference on Machine Learning*, (Tahoe City, California), p. 261, Morgan Kaufmann, July 9-12 1995.

[88] L. Busoniu, R. Babuska, B. D. Schutter, and D. Ernst, *Reinforcement Learning and Dynamic Programming Using Function Approximators.* CRC Press, 2010.

[89] J. N. Tsitsiklis and B. V. Roy, "Feature-based methods for large scale dynamic programming," pp. 59–94, 1994.

[90] R. Becker, S. Zilberstein, and C. V. Goldman, "Solving transition independent decentralized Markov decision processes," *Journal of Artificial Intelligence Research*, vol. 22, p. 2004, 2004.

[91] J. Wu and E. H. Durfee, "Mixed-integer linear programming for transition-independent decentralized mdps," in *Proceedings of the Fifth Interational Joint Conference on Autonomous Agents and Multi-Agent Systems*, 2006.

[92] Z. Yin, K. Rajan, and M. Tambe, "Solving Continuous-Time Transition-Independent DEC-MDP with Temporal Constraints," in *The Sixth Annual Workshop on Multiagent Sequential Decision-Making in Uncertain Domains*, 2011.

[93] L. Blackmore, H. Li, and B. Williams, "A probabilistic approach to optimal robust path planning with obstacles," in *American Control Conference (ACC)*, 2006.

[94] L. Blackmore, "A probabilistic particle control approach to optimal, robust predictive control," in *AIAA Guidance, Navigation, and Control Conference (GNC)*, 2006.

[95] L. Blackmore, "A probabilistic particle control approach to optimal robust predictive control," in *American Control Conference (ACC)*, 2007.

[96] L. Blackmore, A. Bektassov, M. Ono, and B. C. Williams, "Robust, optimal predictive control of jump Markov linear systems using particles," in *Hybrid Systems: Computation and Control*, Lecture Notes in Computer Science, Springer Berlin, 2007.

[97] L. Blackmore, "Robust path planning and feedback design under stochastic uncertainty," in *Proceedings of the AIAA Guidance, Navigation and Control Conference*, (Honolulu, HI), August 2008.

[98] L. Blackmore and M. Ono, "Convex chance constrained predictive control without sampling," in *Proceedings of the AIAA Guidance, Navigation and Control Conference*, 2009.

[99] L. Blackmore, M. Ono, A. Bektassov, and B. C. Williams, "A probabilistic particle-control approximation of chance-constrained stochastic predictive control," *IEEE Transactions on Robotics*, vol. 26, no. 3, pp. 502–517, 2010.

[100] A. Nedic, A. Ozdaglar, and P. Parrilo, "Constrained consensus and optimization in multi-agent networks," LIDS Report 2779, Laboratory for Information and Decision Systems, MIT, December 2008.

[101] A. Undurti and J. P. How, "An online algorithm for constrained POMDPs," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3966–3973, 2010.

[102] D. Fudenberg and J. Tirole, *Game Theory.* MIT Press, 1991.

[103] E. Stringham, "Kaldor-Hicks Efficiency and the Problem of Central Planning," *The Quarterly Journal of Austrian Economics*, vol. 4, pp. 41–50, 2001.

[104] F. Bullo, J. Cortes, and S. Martinez, *Distributed Control of Robotic Networks.* Princeton Series in Applied Mathematics, Princeton University Press, 2009.

[105] J. Wang, *USARSim V3.13.* National Institues of Standards and Technology, University of Pittburgh and The Carnegie Mellon School of Computer Science.

[106] M. Valenti, B. Bethke, D. Dale, A. Frank, J. McGrew, S. Ahrens, J. P. How, and J. Vian, "The MIT Indoor Multi-Vehicle Flight Testbed," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2758–2759, 10-14 April 2007.

[107] A. Geramifard, F. Doshi, J. Redding, N. Roy, and J. P. How, "incremental Feature Dependency Discovery," in *Proceedings of the 23rd International Conference on Machine Learning (ICML)*, 2011.