Online Risk-Aware Conditional Planning with Qualitative Autonomous Driving Applications

by

Matthew Quinn Deyo

B.S., Massachusetts Institute of Technology, 2016

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2018

Author ..... Signature redacted .....
Department of Aeronautics and Astronautics
January 29, 2018

Certified by ..... Signature redacted
Brian C. Williams
Professor, Aeronautics and Astronautics
Thesis Supervisor

Accepted by ..... Signature redacted
Hamsa Balakrishnan
Associate Professor, Aeronautics and Astronautics
Chair, Graduate Program Committee

Online Risk-Aware Conditional Planning with Qualitative Autonomous Driving Applications

by

Matthew Quinn Deyo

Submitted to the Department of Aeronautics and Astronautics
on January 29, 2018, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

Driving is often stressful and dangerous due to uncertainty in the actions of nearby vehicles. Having the ability to model driving maneuvers qualitatively and guarantee safety bounds in uncertain traffic scenarios are two steps towards building trust in vehicle autonomy. In this thesis, we present an approach to the problem of Qualitative Autonomous Driving (QAD) using risk-bounded conditional planning. First, we present Incremental Risk-aware AO* (iRAO*), an online conditional planning algorithm that builds off of RAO* for use in larger dynamic systems like driving. An illustrative example is included to better explain the behavior and performance of the algorithm. Second, we present a Chance-Constrained Hybrid Multi-Agent MDP as a framework for modeling our autonomous vehicle in traffic scenarios using qualitative driving maneuvers. Third, we extend our driving model by adding variable duration to maneuvers and develop two approaches to the resulting complexity. We present planning results from various driving scenarios, as well as from scaled instances of the illustrative example, that show the potential for further applications. Finally, we propose a QAD system, using the different tools developed in the context of this thesis, and show how it would fit within an autonomous driving architecture.

Thesis Supervisor: Brian C. Williams
Title: Professor, Aeronautics and Astronautics

Acknowledgments

I'd like to begin by thanking my family and friends for all of the support for the past six years. I would not have been able to transition to and thrive at MIT without the camaraderie from friends in Simmons Hall, peers in my department, and fellow cadets (now officers) alongside me in AFROTC. This also includes the ten classes of MIT Cross Country and Track and Field athletes that I was fortunate enough compete with or coach during my time here.

This work would not have been possible without the guidance of my research advisor, Professor Brian Williams, and the support from all the MERS members during my time in the group, especially those that worked with me on the Geordi project: Tiago Vaquero, Ashkan Jasour, and Xin "Cyrus" Huang, and undergraduate Yun Chang.

Special thanks is necessary to those that helped me manage my Air Force commitment with my Masters program. Those were my undergraduate advisor, Professor Sheila Widnall, and the cadre of AFROTC Detachment 365, including Lt Col Karen Dillard (Ret.), Capt Peterson Dela Cruz, and Lt Col Sheryl Ott. Without their help, I would not have been able to pursue this Masters following my commissioning nor been able to secure the transfer to my dream job with the 129th Rescue Wing.

Finally, I would like thank the Toyota Research Institute for sponsoring this work.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

The development of autonomous vehicles for consumer use has captured the attention of the public and investors in recent years. Most car manufacturers have either bought AI start-ups, funded their own autonomy research, or partnered with ride-sharing companies that are actively pursuing autonomous capabilities. Many of these companies or alliances, including Waymo, GM/Cruise, Ford/Argo AI, Volvo/Uber, and Daimler/Bosch to name a few, are competing to control the market and have promised deployment of fully autonomous capabilities by 2021. The increased interest in autonomous driving has also highlighted the dangers currently experienced with automobile transportation. And while many disagree on technical approaches, safety metrics, and legality, it appears that forward progress is not being hindered by naysayers and slow legislation. It should be clear that the possible benefits to safety and mobility are great, which is additional motivation for the multitude of research efforts [5].

Safety has always been a big focus for automobile companies. However, previous engineering solutions were unable to address all of the problems and risks, since human error still causes most accidents. A report to Congress in 2008 on Motor Vehicle Crash Causation showed that over 90% of accidents were attributed to human errors like decision, recognition, and performance errors [1]. Out of the roughly 43,400 fatal crashes in the US in 2016, 27% of them included alcohol-impaired driving. In the same year, 26% of US driving fatalities were speeding-related [2]. Without claiming

that autonomous systems will be perfect in all edge cases, we can say that human-specific risks like mental fatigue, speeding, and operating under the influence of drugs, can be reduced.

The need for physical mobility is also very important, manifested recently with the growth and domination of ride-sharing services like Uber and Lyft. The use of autonomous ride-sharing fleets, proposed as future business models by many companies, could further revolutionize access to inexpensive transportation. Researchers at Harvard have identified commuting time as the single strongest factor in the odds of someone escaping poverty, with longer commute times hurting opportunity for upward mobility [10]. Increased access to inexpensive automated ride-sharing services could address the problems of American public transportation decay, which has worsened inequality as more individuals lose affordable mobility options.

1.1 Autonomous Driving as Conditional Planning

Driving is often stressful and dangerous due to uncertainty in the actions of nearby vehicles. Operating a single vehicle within the spatial and temporal limits set by driving laws is not what makes driving difficult. First-time drivers can quickly learn how to safely stay in their lane and to stop at red lights. Instead, most of the complexity of driving comes from nearby agents, both vehicles and pedestrians, which have their own goals and plans. Uncertainty, or incomplete information, is a product of simply not knowing the intentions of all the nearby agents. The use of vehicle-to-vehicle communication and coordination would greatly decrease these uncertainties [14], but the reality is that the vast majority of vehicles are not yet connected. These are the unknowns that make planning for autonomous vehicles complex, especially while trying to minimize the risk of undesirable states, including collisions.

We claim that having the ability to model driving maneuvers qualitatively and to guarantee safety bounds in uncertain traffic scenarios are two steps towards building trust in vehicle autonomy. Why not just learn the control policies by leveraging artificial neural networks and deep learning? The major reason is transparency. An

end-to-end deep learning approach lacks transparency in its decision making, cannot make guarantees on risk, and hence should not be trusted with high-level safety-critical choices that it cannot then defend with reasoning [15]. Instead, we are using qualitative driver models with explicit conditional planning, which allows researchers and operators to understand each choice being made by the planner and make probabilistic guarantees on the execution. Not to be discounted, deep learning technology is vital to the sensing and perception systems, which gather information about the world. Recent advances in computer vision with convolutional neural networks and semantic labeling have achieved impressive results, which are crucial to enabling autonomous vehicles [23]. Machine learning techniques can also be used to build our qualitative models from driving data, which we discuss more in Chapter 6.

As a disclaimer, this work overlooks many real-world driving problems including the sensing, perception, localization, navigation, and control, to name a few. As mentioned, many of these problems have been addressed using machine learning techniques with great success. Instead, we are focused on addressing the planning required for decision making in autonomous driving, including prediction and reasoning. We are calling this sub-problem, nestled in between high-level navigation and lower-level motion planning systems, Qualitative Autonomous Driving (QAD).

In this thesis, we present an approach to the QAD problem using risk-bounded conditional planning. The capabilities of nearby vehicles can be modeled as functions of their state and the environment, analogous to how human drivers often plan their maneuvers. Conditional planning deals with incomplete information by constructing plans that account for every possible situation that could arise. This planning framework allows us to condition the execution of our own plan on the nearby maneuvers that are observed. Stated another way, the autonomous system will always have backup plans for everything. Non-deterministic problems like this also need a model that captures the uncertainty of our agent acting in world. This is often done with Markov Decision Processes.

1.1.1 Partially Observable MDPs

The Markov Decision Process (MDP) is a popular mathematical framework for modeling decision making problems with outcome uncertainty. In the case of autonomous driving, the system transitions are not only dependent on our decisions but can be probabilistic and uncontrollable. Formally, an MDP is a discrete time stochastic control process, where the process is said to be in a state s at each time step. An available action a can be chosen to transition the process to a new state s' with probability described by a transition function $T(s'|s,a)$, while also earning a reward defined by $R(s,a,s')$. These models use the Markov assumption, which says that the conditional probability distribution of future states only depends on the current state. This is how we can write tractable transition functions that only depend on the previous state and action taken instead of all previous states.

We use a Partially Observable MDP (POMDP), a generalized MDP framework that includes hidden states. Many real-world planning and control problems have been modeled as POMDPs [24]. For the problem of autonomous driving, hidden states allow us to model nearby vehicle behavior and the underlying intentions of the drivers. Unable to directly observe the hidden states, we have to maintain a probability distribution over possible states, called a belief state. We say that a belief state b_k is a set of state-probability pairs that describe the distribution of possible states s_k at time k.

1.2 Problem Statement

This thesis is motivated by generating plans for an autonomous vehicle that make guarantees on safety and that are transparent, or explainable, in their methods. These priorities can be met with an online approach in conditional planning with explicit models. This identifies two different problems we are trying to solve; first, the online planning capability for risk-bounded and dynamic partially observable environments, and second, developing the models for representing driving scenarios as CC-POMDPs. We are interested in generating risk-bounded conditional plans

(or policies) for Chance-Constrained Partially Observable Markov Decision Processes (CC-POMDPs) with a finite horizon, defined here in Definition 1.

Definition 1. A CC-POMDP is the tuple $\mathrm{H} = < \mathcal{S}, \mathcal{A}, \Omega, T, O, R, b_0, h, \mathcal{C}, \Delta >$, where

- $\mathcal{S}, \mathcal{A}, \Omega$ are the discrete sets of states, actions, and observations, respectively;
- T is the transition function where $T(s_k, a_k, s_{k+1}) = Pr(s_{k+1}|s_k, a_k)$
- O is the observation function where $O(s_k, o_k) = Pr(o_k|s_k)$
- R is the reward function;
- b_0 is the initial belief state;
- h is the execution horizon;
- $\mathcal{C} = [\mathcal{C}^1, ..., \mathcal{C}^q]$ is the set of q constraints over $\mathcal{S}$;
- $\Delta = [\Delta^1, ..., \Delta^q]$ is the vector of probabilities for each chance constraints.

A chance constraint bound Δ defines a limit for the likelihood that an event happens during execution of the policy. Let b_k be a belief state, and let $Safe_k(\mathcal{C})$ (for "safe at time k") be a Bernoulli random variable denoting whether the system has **not violated** any constraints in $\mathcal{C}$ at time k. We define the execution risk of a policy π measured from the belief b_k out to horizon h as:

$$er(b_k, C|\pi) = 1 - Pr(\bigwedge_{i=k}^{h} Safe_i(C)|b_k, \pi) \tag{1.1}$$

For the offline planning problem, we bound the execution risk of our policy, starting with an initial belief state, to the chance constraints defined in the model. This condition is described in Equation 1.2.

$$er(b_k, C^i|\pi) \leq \Delta^i, C^i \text{ for } i = 1, ..., q \tag{1.2}$$

However, the online problem is to have a conditional plan at each time step, given its updated posterior belief, that satisfies the chance constraints for the whole planning horizon. This bound now includes risk from actions taken in previous steps. While we use the same definitions of expected risk, executing actions and updating the belief now means that risk is being 'spent'. This spent risk cannot be reallocated in subsequent replanning, otherwise the chance constraints are not met over the whole planning horizon. This requires the definition of local execution risk, where $ler(b_k|\pi(b_k))$ is the immediate risks from taking the action specified by policy π in belief b_k.

$$ler(b_k, C|\pi) = 1 - Pr(\bigwedge_{i=k}^{k+1} Safe_i(C)|b_k, \pi) \tag{1.3}$$

However, risk is spent through execution and cannot be ignored in replanning. Successful execution of a risky action is not cause to reallocate that risk to the remaining policy and roll the dice of fate again.

With the local execution risk, we can update the original chance constraints and prevent risk from being reallocated in the subsequent updated policies. Our online risk-bounded conditional planning objective is defined in Equation 2.4.

$$er(b_{k+1}, C|\pi_{k+1}) \leq \Delta - \sum_{j=0}^{k} ler(b_j, C|\pi_j) \tag{1.4}$$

1.3 Summary of Approach

Widespread acceptance and use of any autonomous system is going to be built on trust, which clear, understandable decision making and probabilistic guarantees can help foster. The need for a risk-aware online conditional planner in our autonomous driving applications drove the development of the iRAO* algorithm. Based on RAO*, the new incremental approach allows us to apply risk-aware conditional planning to bigger problems. The algorithm efficiently uses the dynamic programming structure from previous iterations to perform quick risk assessments and focus replanning on

parts of the policy that violate chance constraints. To handle larger or dynamic search problems, iRAO* can easily be used in a receding horizon approach, reducing space complexity and the number of a prior assumptions required.

The hybrid, multi-agent models presented, built on the POMDPs model from Section 1.2, are based on many ideas from classical planning. There is a collection of agent models, one for each vehicle in the current planning horizon. Each of the agent models has a library of action models that represent possible driving maneuvers. Action models are composed of the preconditions, effects, duration, probabilistic flow tubes, and control input sequences to represent the dynamics of vehicle driving.

These proposed driving models are further developed to include maneuvers of variable durations to address the need for asynchronous multi-agent decision epochs. Adding durations to the multi-agent models makes an already complex problem exponentially worse with regards to branching factors. To address the exploding space complexity, we present two algorithmic approaches, mid-permutation pruning and state clustering. The risk-based pruning of atomic, synchronized actions is extended to durative actions by checking chance constraints during the permutation of possible action combinations. Similar to the Markov assumption, the second approach comes from our ability to ignore the sequence of actions taken and instead focus on the resulting states, reducing the complexity by grouping similar states and conditioning on those instead.

1.4 Thesis Contributions and Outline

This thesis presents three main contributions. First, the Incremental Risk-Aware AO* (iRAO*) algorithm gives us the ability to extend risk-bounded conditional planning to larger dynamic systems with a receding horizon approach. Second, we present Hybrid Multi-Agent models, built on POMDPs, to represent the Qualitative Autonomous Driving (QAD) problem. Finally, the models are extended to durative actions and we develop two algorithmic approaches to address the increased space complexity.

Chapter 2 presents additional background on the planning problem and previous

approaches. An illustrative example is outlined that is later revisited to motivate concepts like execution risk and to highlight the performance of the conditional planners. Further discussion is presented on bounding risk and how we define the safety guarantees of a policy. Finally, RAO* is reviewed for the reader, along with the reasons that the existing offline planner falls short for our QAD application.

Chapter 3 starts with the inspiration and ideas behind iRAO*. The algorithm itself is presented and worked through in an example. Then the completeness of the algorithm is shown and practical uses of the algorithm are detailed.

Chapter 4 defines the Hybrid Multi-Agent CC-POMDP models developed in efforts to represent the Qualitative Autonomous Driving problem. The chapter goes through each part of the model, describing how they address the challenges in modeling traffic scenarios, and then discusses limitations. Chapter 5 is an extension of the driving models with variable durations on the actions and presents two algorithmic approaches to deal with the resulting exponential branching.

Finally, Chapter 6 presents quantitative results from iRAO* and the other tools outlined in this thesis, while Chapter 7 is a conclusion with insights and recommended future research to continue this work in Qualitative Autonomous Driving.

Chapter 2

Conditional Planning and RAO*

This thesis approaches the problem of safely maneuvering an autonomous vehicle through a conditional planning framework. Conditional planning is a way to model and deal with uncertainty in the world. Conditional planners produce contingency plans, also known as policies, that are conditioned on possible outcomes [26]. Basically, we have a plan on hand in advance to deal with any contingency that could occur. With this framework, we can account for uncertainties in the execution of a plan, either from stochastic action outcomes, uncontrollable agents in the environment, or partially observable states.

2.1 Illustrative Example

We motivate the following problem as an illustrative example and will revisit it a few times in this thesis to promote understanding of the risk-bounded conditional planning concepts. Note that this problem was designed to highlight import concepts with regards to safety guarantees and our definition of execution risk. This example fails to accurately represent the complexity faced in most conditional planning problems.

Our example problem is robot navigation through discrete location states to a goal. We model a 2x3 grid world where our robot needs to transition from his starting location in the bottom left cell to the goal location marked with a flag. Our robot hero's name will be Colony-Bot. Colony-Bot can take one of three actions at each

time step, either *moveRight*, *moveUp*, or *moveDown*. The deterministic problem is on the left in Figure 2-1, where each action transitions Colony-Bot to an adjacent cell based on the action name, shown with green arrows. The solution to the first problem is a deterministic plan, which is a sequence of actions to execute. The optimal plan is this example being [*moveRight,moveRight*] because it is the fewest actions required to reach the goal (the shortest path).

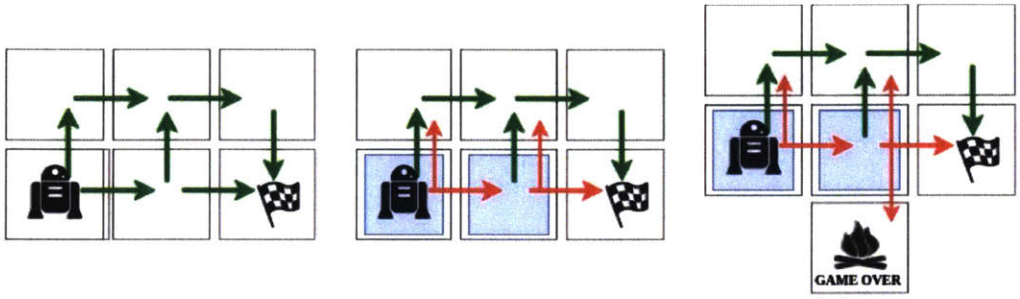


Figure 2-1: Example problem of robot navigation as an MDP that is first deterministic, then non-deterministic, and finally with added failure state.

Extending to a non-deterministic problem, the center image in Figure 2-1 contains icy conditions in the two lower left cells. These icy blue cells add stochastic outcomes to the *moveRight* action, shown with the red arrows, where Colony-Bot could transition one cell to the right or to the adjacent cell above. All other actions, including *moveRight* from non-icy cells remain the same. A deterministic plan is now not enough, as [*moveRight,moveRight*] does not guarantee that Colony-Bot reaches the goal. Instead, the solution to the second problem is a conditional plan, or policy, that maps states to actions. The second problem motivates the need for conditional planning to handle modeled uncertainties, in this case some action transitions not being deterministic.

The third image on the right of Figure 2-1 finally incorporates the concept of risk. The lower cell labelled *GameOver* is a failure state for Colony-Bot, which it can reach by executing *moveRight* in the center cell. The objective then becomes the following: navigate to the goal while avoiding the failure state. If safety is the highest priority, then this example has the lower utility (longer route) solution of [*moveUp,moveRight,moveRight,moveDown*] that avoids the uncertainty of the ice.

However, if some risk is acceptable while maximizing utility, we need to ensure the policy does not foolishly endanger Colony-Bot. In order to make guarantees on Colony-bot's safety, we want to bound the likelihood that it transitions into the failure state. This likelihood of failure, or risk, is analogous to the constraints that we model with CC-POMDPs in this thesis, where we are interested in bounding the likelihood (the chance) that constraints are violated. We discuss how to define and calculate these risk bounds in Section 2.3.

We can formally define this example problem in Definition 2 as a CC-MDP, since the state of the Colony-Bot is fully observable.

Definition 2. Our icy robot CC-MDP is the tuple $\mathrm{M} = < \mathcal{S}, \mathcal{A}, T, R, s_0, s_{goal}, \mathcal{C}, \Delta >$, where

- $\mathcal{S} = [upperLeft, ..., goal, gameOver]$ are discrete grid locations;
- $\mathcal{A} = [moveRight, moveUp, moveDown]$ are the actions;
- T is the transition function where $T(s_k, a_k, s_{k+1}) = Pr(s_{k+1}|s_k, a_k)$
- R is the reward function;
- s_0 is the initial state;
- h is the execution horizon;
- $\mathcal{C}$ constraint of not being in a *gameOver* state;
- Δ is a probability for bounding the single chance constraint.

We can also scale the icy robot problem as shown in Figure 6-4 to any length to include n *GameOver* states, where the original 2x3 world in Figure 2-1 had $n = 1$. The scaled icy robot problem is important for understanding additional concepts, including how future risk can be discounted by branching and how risk is 'spent' through the execution of unsafe actions. These two concepts are introduced in later sections which will refer back to this Figure 2-2.

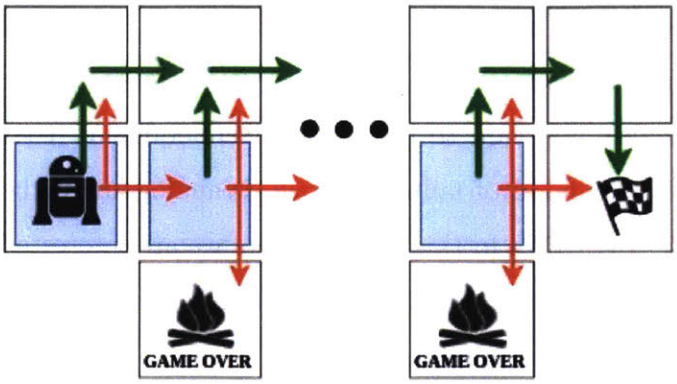


Figure 2-2: Scaling the illustrative example in length and number of failure states to motivate weaknesses in offline risk-bounded conditional planning.

2.2 Chance-Constrained POMDPs

While Chapter 1 introduced POMDPs and chance constraints for the Problem Statement, this section takes more time to develop the concepts of the CC-POMDP model.

The generalization from MDPs to the Partially Observable MDPs (POMDPs) allows us to model hidden states in our problem. This is important because almost all real world autonomous systems have to plan with limited or noisy information. For example, in robot localization, even with calibrated landmarks or triangulation, the true position or orientation is not a given. Sensing actions can be a designed into autonomous agents, with the goal of reducing uncertainty, but the true states are ultimately not observable and decisions need to be made that respect the uncertainty. As another way to ground the model: POMDPs are simply Hidden Markov Models (HMMs) with controllable actions, and HMMs are the partially observable extension to Markov chains. This is why we can use estimation techniques for HMMs like filtering with our POMDPs when planning ahead to account for possible observations.

Modeling constraints for our systems is another problem. The standard POMDP does not provide the means to model hard constraints, such as collisions or resource constraints (think time or fuel constraints). A number of extensions to POMDPs, and respective planners, have been proposed with the goal of generating optimal plans while bounding risks with uncertainties. For example, constrained POMDPs (C-POMPDs) were made to solve the same problem [22, 32]. C-POMDPs use a con-

straint penalty to assign unit cost to states in violation of the constraints. With this framework, a planner optimizes utility of a policy while keeping the expected constraint penalty value below a bound. There has been a lot of research on solving constrained-POMDPs using approaches such as linear programming and value iteration [12, 13, 22, 32].

However, this definition only works if constraint violations are terminal states, meaning that policy execution ends if a constraint is violated. Many problems have constraints that are not terminal. As one example, this thesis presents models for the QAD problem where the constraint is on our Ego vehicle being in near-collision states, which is exactly when planned recovery maneuvers that avoid collision or minimize damage should be executed. Another example is when violation can reduce uncertainties, like a mobile robot bumping into a wall, where the collision is not destructive. While the planner attempts to bound the risk of collision with walls, the event could also help the robot improve localization estimates if it does occur.

The CC-POMDP extension, in contrast, defines execution risk as the likelihood of transitioning from a safe state to a violating state and does not make any assumptions about violating states being terminal. has a more definition of risk allocation is used in a chance-constrained POMDP (CC-POMDP) framework [28].

We defined the CC-POMDP in Section 1.2. The solution to a CC-POMDP is an optimal policy $\pi^* : \mathcal{B} \rightarrow \mathcal{A}$, mapping belief states in $\mathcal{B}$ to actions in $\mathcal{A}$, such that both Equations 2.1 and 2.2 hold.

$$\pi^* = arg\max_{\pi} \mathbb{E}\left[\sum_{k=0}^{h} R(s_k, a_k)|\pi\right] \tag{2.1}$$

$$er(b_k, C^i|\pi) \leq \Delta^i, C^i \in 2^C, i = 1, ..., q \tag{2.2}$$

For the offline planning problem, we defined the execution risk of our policy based on the initial belief state as presented again in Equation 2.3.

$$er(b_k, C|\pi) = 1 - Pr(\bigwedge_{i=k}^{h} Safe_i(C)|b_k, \pi) \tag{2.3}$$

However, in planning online with CC-POMDPs, we redefine the execution risk of our policy to be a function of updated posterior beliefs during execution. We are bounding the execution risk over the entire planning horizon, including previous actions, by taking into account spent risk and incrementally updating the policy as necessary. This online CC-POMDP condition is presented again in Equation 2.4, where the local execution risk $ler(b_k, C|\pi_k)$ is in Equation 2.5.

$$er(b_{k+1}, C|\pi_{k+1}) \leq \Delta - \sum_{j=0}^{k} ler(b_j, C|\pi_j) \tag{2.4}$$

$$ler(b_k, C|\pi_k) = 1 - Pr(\bigwedge_{i=k}^{k+1} Safe_i(C)|b_k, \pi) \tag{2.5}$$

2.3 Bounding Risk

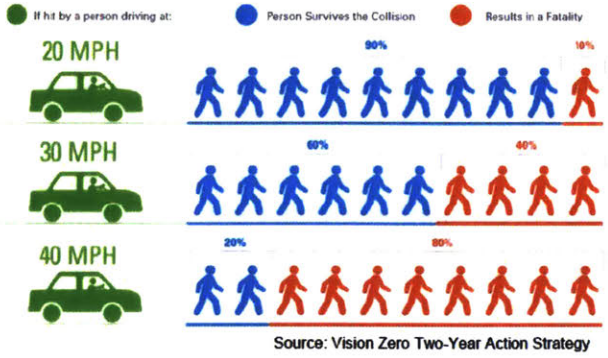


Figure 2-3: Real life example of the trade-offs between utility and safety.

To ensure safe operation of autonomous systems, we enforce limits on the amount of risk in the execution plans. This is important because utility (what we are trying to maximize) and safety are often at odds with each other. In driving, for example, we want to get to our destination faster, wasting less time in transit. Unfortunately, driving faster to minimize travel time (maximize our utility) increases the risk of

accidents and the amount of damage done in any collision. Figure 2-3 is a practical example, and tragic reality, of this trade-off between utility and safety when it comes to pedestrian accident fatalities vs car velocity.

We define risk as the likelihood of violating a constraint. For a given fully observable states, this is a binary true or false if the state is in violation of the constraint. However, when planning with partially observable states we use the belief b_k to represent the probability distribution of possible hidden states at time k. Therefore the risk of a belief state is the expected risk over the states as shown in Equation 2.6.

$$r(b_k) = \sum_{s \in \mathcal{S}} r(s_k) b(s_k) \text{ where } r(s_k) \in \{0, 1\} \text{ and } b(s_k) \in [0, 1] \tag{2.6}$$

We are using the same definition of execution risk as Santana, defined with RAO* [28], and included in Equations 2.3 and 2.2. However there are alternative approaches to bounding the risk of a policy.

There are alternative ways to define the execution risk of a policy. One example is bounding the state risk for all non-terminal nodes in the policy to the chance constraint Δ, shown in Equation 2.7. This approach is like thinking about actions as the risky part and only looking one action ahead, which appears to produce a policy that excludes all actions with risk greater than the chance constraints.

$$risk(b_k, C^i | \pi(b_k) \leq \Delta^i, \forall i, b_k \text{ s.t. } b_k \text{ is nonterminal} \tag{2.7}$$

However, it is easy to show that this makes no guarantees on the overall safety of a policy. Revisiting the scaled icy-robot problem from Figure 2-2, if the probability of sliding into each *gameOver* state is 10% and our risk bound is 15% then we can continue with *moveRight* across as many icy cells as we want. However, the likelihood of success when $n > 1$ is the probability of safely transitioning between each icy cell, such that $Pr(success|\pi) = (1 - action\text{-}risk)^n$, which is not able to make guarantees on violating the chance constraints as it is a function of the number of risky actions.

Another option is to bound the execution risk from all non-terminal nodes in the policy to chance constraint Δ, shown in Equation 2.8, using the definition of

execution risk from Equation 2.3. However this approach is overly conservative, not taking into account the likelihood of reaching all of the beliefs that have been bound to Δ. This approach is important to understand because it will appear that the incremental RAO* algorithm enforces the same policy. However, this approach will result in less optimal solutions that RAO* and iRAO*, because of the conservative approach to bounding execution risk.

$$er(b_k, C^i|\pi) \leq \Delta^i, \forall i, b_k \text{ s.t. } b_k \text{ is nonterminal} \tag{2.8}$$

2.4 Risk-Aware AO* (RAO*)

The main contribution of this thesis is augmenting a conditional planner called Risk-bounded AO* (RAO*) [28, 27] for online applications. Based on the conditional planning AO* algorithm [19], RAO* utilizes value and risk heuristics to guide the search towards safe and optimal policies. In the end, RAO* finds optimal policies with maximum expected reward over a finite horizon while satisfying all chance constraints. The remainder of this chapter covers the RAO* algorithm as detailed in Santana's PhD thesis [27], including all of the equations from the original work and additional commentary. We made a few small edits to the algorithms and equations for clarity but the overall approach is the same as the thesis.

Understanding how RAO* works is aided by a familiarity with the AO* conditional planning algorithm. Both look for a policy with heuristic forward search, starting with an initial node in the state space graph and always keeping track of the current best policy. Admissible heuristic values are used to guide the search towards an optimal solution. Upon expansion of the most promising leaf node, the updated state values are backed up to the initial node and the current best policy is reevaluated given the new information. This series of expanding nodes and updating the policy continues until the best policy is complete, meaning the policy leaf nodes all meet terminal conditions. RAO* adds to this with risk heuristics and pruning actions that exceed chance constraints.

2.4.1 Expanding Belief States

The partially observable states in the CC-POMDP models are represented by belief states, introduced in Section 1.1.1, which are probability distributions of possible hidden states. Each belief b_k is a set of state-probability pairs, called particles, for time k such that all particle probabilities $b(s_k)$ sum to 1.

$$\sum_{s_k \in \mathcal{S}} b(s_k) = 1, \forall k \in \mathbb{N} \tag{2.9}$$

There are actually two different belief states at each time to differentiate between, the prior belief $\bar{b}_k$ and the posterior belief $\hat{b}_k$. The prior belief at time k uses all previous observations and actions to estimate the state distribution. The posterior belief goes one step further and incorporates the latest observation o_k into the estimation. The two belief states are formally defined as the following:

$$\begin{aligned} \bar{b}(s_k) &= Pr(s_k | o_{1:k-1}, a_{0:k-1}) \\ \hat{b}(s_k) &= Pr(s_k | o_{1:k}, a_{0:k-1}) \end{aligned} \tag{2.10}$$

Given a selected action a_k from belief b_k we can predict the next prior $\bar{b}(s_{k+1})$ for each state using the transition function T from the CC-POMDP model. This prediction step is outlined in Equation 2.11. After calculating prior likelihoods for each state, we can also perform measurement updates for the possible observations to find the posterior belief $\hat{b}(s_{k+1})$. This update step uses the models observation function and is outlined in Equation 2.12, with normalization constant in Equation 2.13. While the prediction step produces one prior belief $\bar{b}_{k+1}$ from the previous posterior $\hat{b}_k$, the measurement update is a function of which observation occurs, so there are d resulting posteriors $\hat{b}_{k+1}$ for each prior, where d is the number of possible observations. This process is often called *filtering* when applied to HMMs, however this approach is *conditional filtering* because we are conditioning for all possible observations before they are measured.

$$\bar{b}(s_{k+1}|o_{1:k}, a_{0:k}) = \sum_{s_k} T(s_k, a_k, s_{k+1})\hat{b}(s_k) \tag{2.11}$$

$$\hat{b}(s_{k+1}) = Pr(s_{k+1}|o_{1:k+1}, a_{0:k}) = \frac{1}{\eta} O(s_{k+1}, o_{k+1})\bar{b}(s_{k+1}) \tag{2.12}$$

$$\eta = P(o_{k+1}|o_{1:k}, a_{0:k}) = \sum_{s_{k+1}} O(s_{k+1}, o_{k+1})\bar{b}(s_{k+1}) \tag{2.13}$$

2.4.2 Risk Calculations

The execution risk of a policy, as defined in Equation 2.3, is basically $1 - P(safe)$, where *safe* is not violating the constraints. We rewrite Equation 2.3 in Equation 2.14, where we isolate the probability of being safe at step k in the second term. This second term is equivalent to the expected violation of belief b_k given in Equation 2.15

$$er(b_k|\pi) = 1 - Pr(\bigwedge_{i=k+1}^{h} Sa_i|Sa_k, b_k, \pi)Pr(Sa_k|b_k, \pi) \tag{2.14}$$

$$Pr(Sa_k|b_k, \pi) = 1 - r_b(b_k, C) = 1 - \sum_{s_k \in S} b(s_k)c_v(s_k, C) \tag{2.15}$$

Execution risk for the policy π at belief state b_k

Observation distribution at time $k+1$ given non-violating b_k

$$er(b_k|\pi) = r_b(b_k) + (1 - r_b(b_k)) \sum_{b^{sa}_{k+1}} Pr^{sa}(o_{k+1}|\pi(b_k), b_k)h_{er}(b^{sa}_{k+1}|\pi) \tag{2.16}$$

$$Pr^{sa}(o_{k+1}|a_k, b_k) = \sum_{s_{k+1}} O(s_{k+1}, o_{k+1})\bar{b}^{sa}(s_{k+1}|a_k) \tag{2.17}$$

Execution risk bounds are also propagated down to children to facilitate pruning risky actions during heuristic forward search. This is done with Equation 2.18, where the pruning to prune risky actions on expansion, based on the risk heuristic values.

$$\Delta'_{k+1} = \frac{\frac{\Delta_k - r_b(b_k)}{1 - r_b(h_k)} - \sum_{o_{k+1} \neq o'_{k+1}} Pr^{sa}(o_{k+1}|\pi(b_k), b_k) h_{er}(b^{sa}_{k+1}|\pi)}{Pr^{sa}(o'_{k+1}|\pi(b_k), b_k)} \tag{2.18}$$

2.4.3 The Forward Search

This section presents the RAO* algorithm as outlined in [27], with some modifications for clarity and improved performance. The two improvements to the algorithm are pruning low-likelihood branches and handling dead-ends. To prune low-likelihood branches, the search prioritizes the highest probability leafs for expansion and then terminates the search when guarantees can be made over the remaining leafs. Dealing with planning dead-ends is discussed more in the algorithm descriptions that follow. One change for clarity was the function *expand-policy* being renamed to *expand-graph*, as the expanded leaf node n is from the policy but not added to the policy in this function.

The search begins in Algorithm 2.1 with the creation of explicit graph G and policy π containing only the initial belief b_0. After that, *RAO** alternates between expanding nodes and updating the policy, only stopping after enough of the leafs have been expanded to terminal beliefs. Previously, the termination condition was that all of the leafs in the policy were terminal beliefs, usually defined as reaching the planning horizon h defined in the model H.

Algorithm 2.1 RAO*

Input: CC-POMDP H, initial belief b_0.
Output: Optimal policy π, where $er(b_0|\pi)$ meets chance constraints.

```
1: function RAO*(H,b_0)
2:     Start explicit graph G and policy π with b_0
3:     while not termination-condition(H, b_0, π) do
4:         n, G ← expand-graph(G, π)
5:         π ← update-policy(n, G, π)
6:     return π

1: function termination-condition(H,b_0,π)
2:     leaf-risk = sum of non-terminal leaf likelihoods in π
3:     if er(b_0|π) + leaf-risk < Δ then
4:         return True
5:     return False
```

The *expand-graph* function, outlined in Algorithm 2.2, is responsible for building out the explicit graph, expanding the most likely non-terminal leaf node each time. The expansion of children ch in line 4 is a result of branching on all transitions and observations. If fully observable, then the children are simply the distribution of outcomes from stochastic transitions. For the partially observable case, the children are also conditioned on observations, meaning there are children belief states for each of the possible posteriors from Equation 2.12.

Algorithm 2.2 expand-graph

Input: Explicit graph G, policy π.
Output: Updated graph G', expanded leaf node n.

1: **function** *expand-graph*(G, π)
2: $G' \leftarrow G$, $n \leftarrow$ most-likely-leaf(G, π)
3: **for** each action a available at n **do**
4: $ch \leftarrow$ expand-children(n, a) with (2.11, 2.12, 2.13)
5: $\forall c \in ch$ estimate execution risk er with (2.16)
6: $\forall c \in ch$ compute execution risk bound with (2.18)
7: **if** no $c \in ch$ violates $er >$ execution risk bound **then**
8: $\forall c \in ch$ estimate value Q^* with (2.19),
9: $G' \leftarrow$ add ch to G with hyperedge
10: **if** no hyperedge added to n **then**
11: mark n as `dead-end-terminal`
12: **return** G', n

The actions available at node n are based on the model H, which is implicitly accessible throughout the algorithm for transition and observation calculations. The execution risk of each child is estimated with risk heuristics in line 5. The forward-propagated execution risk bounds then use these estimates to calculate the maximum risk for each child of action a based on how much their siblings use up (Equation 2.18). Early pruning of actions occurs in line 7 of *expand-graph* if the heuristic risk estimates from line 5 exceed the upper bounds propagated in line 6. If the risk estimate for the children do not violate the execution risk bound during expansion, then the action becomes a candidate for the policy. Finally, this is when we estimate the value of taking action a and bundle the children ch together as a hyperedge in explicit graph G (lines 8 and 9).

How we choose which action to take in update-policy:

$$\hat{Q}(b_k, a_k) = \sum_{s_k} R(s_k, a_k) b(s_k) + \sum_{o_{k+1}} Pr(o_{k+1}|a_k, b_k) h_Q(b_{k+1}) \tag{2.19}$$

$$\hat{\pi}(b_k) = \arg\max_{a_k} \hat{Q}(b_k, a_k) \tag{2.20}$$

One change to the original RAO* algorithm is the specification of dead-end terminal nodes in line 13 of *expand-graph*, and later in line 11 of *update-policy*. Without this, RAO* would just mark a belief as terminal when all of the actions were pruned, based on children risk (line 12). The algorithm then treated the node the same as those that were goal terminal or on the frontier of horizon h and terminal. These beliefs could then remain on the best policy graph, if the action from their parent had optimal Q values (see *update-policy* function), regardless of having achieved the planning objectives. Now we are labeling these nodes as `dead-end-terminal` so that they can be treated differently than goal or horizon-terminal nodes.

We identified and tested two potential approaches to handling these `dead-end-terminal` nodes. The choice is ultimately up to the modeler as to the desired behavior when dealing with planning dead-ends. One approach is to set the state risk $r(b_k)$ to 1 for dead-end beliefs. This has the effect of treating model dead-ends as failures, or chance constraint violations, so the action into the dead-end is pruned if having the dead-end exceeds the risk bounds. This means that if the execution risk bound for the node is greater than 100%, which often happens in low-likelihood parts of the graph, then it could be included in the policy if there are not more favorable alternatives based on Equation 2.19. The second approach is to set the Q value for the b_k to the extreme *-infinity* (or *+infinity* if minimizing). This will prioritize the removal of dead-ends from the policy above all other considerations. One potential issue with maxing out the value/cost is ruining the admissible heuristic guide if the node cannot be removed from the policy, for example, if it was the only action that met the chance constraints and it could not be replaced.

The *update-policy* function then investigates the newly expanded node n and all

Algorithm 2.3 update-policy

Input: Expanded n, explicit graph G, policy π.
Output: Updated policy π'.

```
1: function update-policy(n, G, π)
2:     π' ← π, Z ← set containing n and its ancestors in π
3:     while Z ≠ ∅ do
4:         n ← remove(Z) node n with no descendant in Z
5:         while there are available actions at n do
6:             a ← next best action at n (2.20) that satisfies execution risk bound
7:             recompute execution risk bound for children of hyperedge (n, a)
8:             if no children violates its Δ_c then
9:                 π'(n) ← a; break
10:        if no action selected at n then
11:            mark n as dead-end-terminal
12:    return π'
```

of its ancestors in the current policy π. This process starts at the bottom of the policy tree with n before moving up to each ancestors. The best action for each node is chosen by updating the $Q*$ estimates with Equation 2.19. These values are backed up from leaf heuristic estimates so the best action for a node can change once the children are expanded and more information is gained. We also recompute the execution risk bounds for each action as we work our way up the policy because those are also based on heuristics until more information is available from expansions. Actions that met the bounds previously could be pruned on a subsequent *update-policy*. And as discussed earlier, we now identify `dead-end-terminal` nodes in line 11 when no actions remain available.

2.4.4 Example

This section includes an example with implementation of the RAO* steps. While there is also a grounded example available in [27], this specific problem is going to be used again in Chapter 3 as the example for online incremental planner. We also include examples of the improvements, handling dead-ends and pruning based on likelihood.

There are many symbols and colors used in these search diagrams. Each circle is

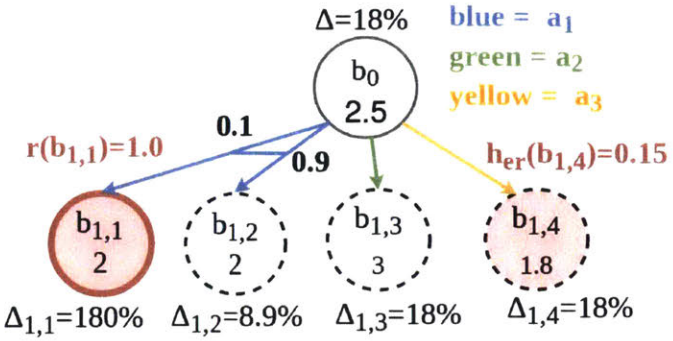


Figure 2-4: First expansion of the explicit graph, starting with b_0, where there are three different actions to choose from.

a node in the explicit search graph, representing a separate belief state. The nodes have dotted outlines if they are non-terminal leaf nodes, while solid outlines for leafs indicate terminal beliefs. Red filing indicates that there is some associated state risk, which is specified by nearby text. If $r(b_k)$ is specified, it is a terminal belief where the state risk is used, while if $h_{er}(b_k)$ is used, then it is a non-terminal belief where we use an admissible risk heuristic. The colored edges are transitions that result from actions, where connected edges of the same color indicate a hyperedge with multiple possible belief outputs that can be conditioned on observation. In the case of a hyperedge with multiple children, the posterior likelihood of each belief is indicated next to each edge. Finally, the numbers in each node below the identification $b_,$ indicate the estimated value of the belief. These are purely heuristic values for leaf nodes and a backup value otherwise. In this example we are attempting to minimize the cost, so the lowest value action is selected and the admissible heuristic should underestimate the actual value.

The *RAO** algorithm begins with an explicit graph and policy containing only b_0. We then use *expand-graph* on b_0, as it is the only non-terminal leaf in the policy, with the result shown in Figure 2-4. There are three possible actions for this first belief. Action a_1 results in two possible beliefs, with a 10% chance of transitioning to a terminal belief with risk of 1.0 and a 90% chance of a non-terminal belief with risk heuristic of zero. Actions a_2 and a_3 have only one possible belief to transition to, but $b_{1,4}$ has a risk heuristic value of 0.15. The forward execution risk bounds

are calculated using Equation 2.18 and shown below each node. Notice how $b_{1,2}$ is allocated a lower risk bound due to its sibling 'consuming' some of the risk. None of the children beliefs violate their risk bounds so no pruning occurs.

The *policy-update* algorithm is then applied to the expanded belief and all ancestors in the policy, shown in Figure 2-5, but in this case b_0 is the root. We are assuming that all actions in the this example have unit cost. This means the estimated value of taking actions a_1, a_2, and a_3 are 3, 4, and 2.8 respectively. The selected action for b_0 is then a_3.

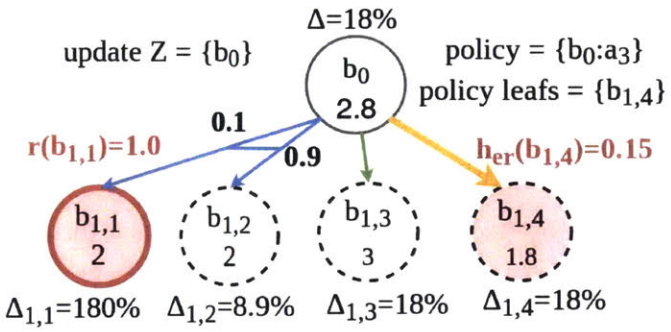


Figure 2-5: First policy update where we update the best action for b_0, select action a_3, and backup the new $Q*$ estimate of b_0 as 2.8.

The node selected for the next expansion is the only candidate $b_{1,4}$. We see that there are two available actions at $b_{1,4}$ and both exceed the propagated execution risk bounds. In Equation 2.18, the left term in the numerator represents risk bound that the parent can pass down while the right term captures the risk 'consumed' by siblings. In the case of $b_{2,2}$, the bound is negative, which is a result of the sibling $b_{2,1}$ consuming more than the allowable risk from the parent. At this point, both actions a_1 and a_2 are pruned from belief $b_{1,4}$, as shown in Figure 2-6. The previous *RAO** algorithm would terminate the search at this point because $b_{1,4}$ was marked as terminal, but *update-policy* for root b_0 still chose action a_3 because it had the best estimated value. We address this by labelling the node `dead-end-terminal` to differentiate it from `terminal` nodes that indicate a goal or planning horizon. We proposed two approaches to dealing with these `dead-end-terminal` nodes and in this example we set $r(b_{1,4}) = 1.0$.

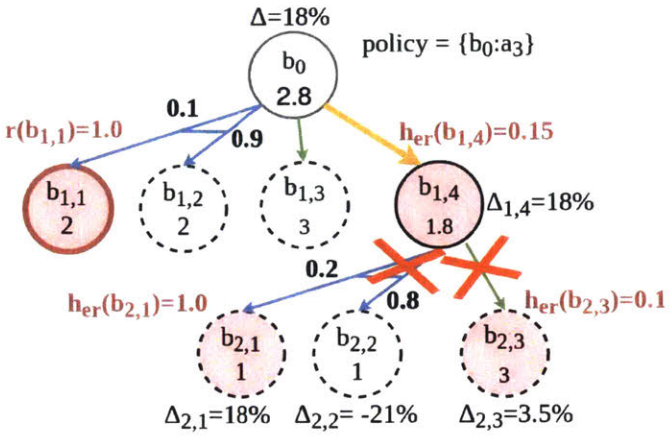


Figure 2-6: Second expansion of a policy leaf node, where $b_{1,4}$ is selected and both possible actions are pruned by forward execution risk bounds. The belief $b_{1,4}$ is marked as `dead-end-terminal` as a result.

Updating the risk on the dead-end forces *update-policy* for root b_0 to select the next best action a_1, providing us with a new non-terminal leaf in the policy $b_{1,2}$. Expansion of this newest leaf shows two available actions and neither are pruned with the propagation of execution risk bounds. The action a_2 appears to be the best action for $b_{1,2}$ so it is added to the policy. The second update, third expansion, and third update described in this paragraph are shown in Figure 2-7.

The final diagram in Figure 2-8 shows the expansion of the leaf $b_{2,4}$ and the policy update where $Z = \{b_{2,4}, b_{1,2}, b_0\}$. The value estimate of $b_{2,4}$ is update to 1.7 instead of 1, and this change makes action a_1 more attractive at $b_{1,2}$. The value estimate of $b_{1,2}$ is updated accordingly. The belief b_0 is the last to be checked in *update-policy* and after updating the value estimate, the policy for the root belief remains the same.

The second improvement to RAO^* is also illustrated here, where we do not need to expand the non-terminal leaf $b_{2,3}$. Based on the *termination-condition* in the $iRAO^*$ algorithm, the likelihood of $b_{2,3}$ occurring is so low that even if the branch was guaranteed to violate the chance constraints, $r(b_{2,3}) = 1.0$, the overall chance constraint Δ will still be met. This can also been seen in the propagated execution risk bound for belief $b_{2,3}$, where violation is impossible for that branch. The RAO^* search is terminated at this point, with a resulting policy of $\pi = \{b_0 : a_1, b_{1,2} : a_1\}$. This resulting policy is revisited in the Section 3.2.3 as the first input for the incremental

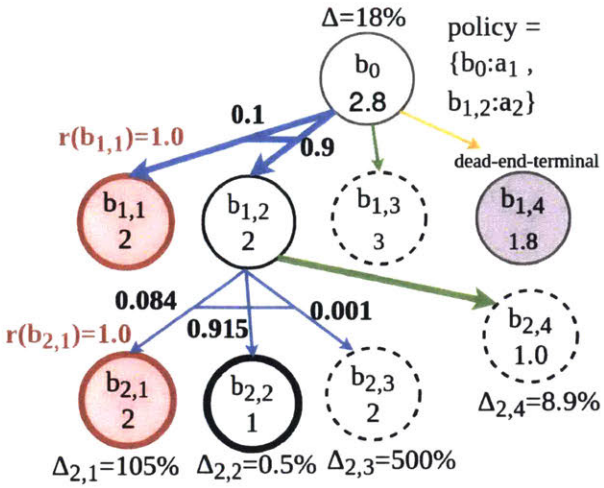


Figure 2-7: Second policy update, third expansion, and third policy update in one step. Belief b_0 is mapped to action a_1, new non-terminal leaf $b_{1,2}$ is expanded, and $b1, 2$ is mapped to action a_2.

version of the risk-aware planner.

2.4.5 Limitations for QAD Applications

Now, if we model an autonomous driving scenario with a CC-POMDP as outlined in Definition 1, RAO* returns the optimal offline risk-bounded policy, providing us with a contingency plan of maneuvers to execute. And that's exactly where we begin in approaching this problem with conditional planning. However, there are some limitations that appear in applying the algorithm as an offline planner for QAD applications.

To reiterate, it is the uncertainty in the actions of nearby vehicles that we want to condition on in our QAD system. Unfortunately, conditional planning and probabilistic domains are both known for their combinatorial size. The tractability of the search space quickly becomes an issue when trying to reason through driving scenarios where nearby vehicles each have complex driver models to consider. In conditioning on the possible maneuvers of multiple nearby agents, we are actually planning for all the contingencies from permutations. For example, if we have 5 nearby vehicles and we model them each as having 5 possible maneuvers, we are looking at 3125 possible

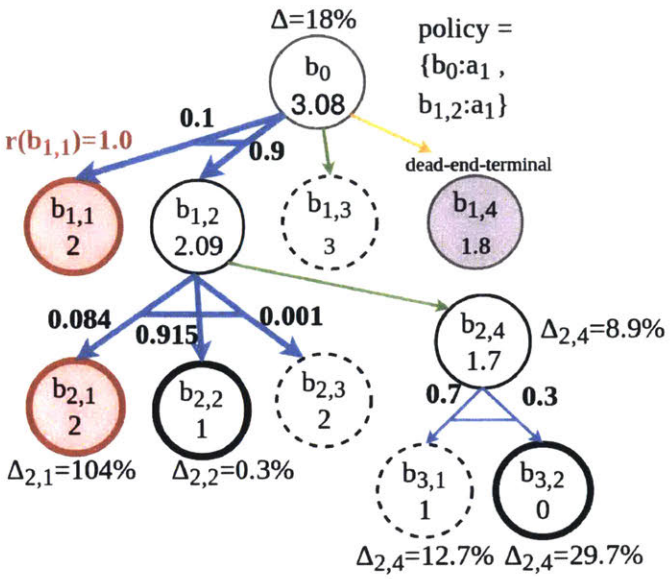


Figure 2-8: Final expansion of non-terminal leaf $b_{2,4}$ and policy update so that $b_{1,2}$ now has action a_1. Non-terminal leaf $b_{2,3}$ can be ignored because of the low-likelihood pruning.

contingencies. Planning for all of the permutations after just 3 maneuvers is not tractable, with over 30 billion to consider. RAO* makes the first steps by pruning risky branches of the search and now pruning low-likelihood outcomes. However, as an offline conditional planner that is complete and makes no approximations, RAO* remains limited in its ability to handle the size of long planning horizons in large state spaces.

One approach to minimizing the branching factor from multi-agent action combinations is to attempt to model different behaviors in the agents. For example, instead of considering 5 maneuvers for each of the 5 nearby vehicles, maybe our model recognizes 3 of the vehicles as cautious drivers, who are each likely to take 1 of just 3 actions. That modeling choice reduces the space to 675 contingencies, still a lot of possible outcomes but moving in the right direction. Unfortunately, effective recognition of behaviors that could prune the model complexity needs to happen online. The second limitation of RAO* is that as an offline planner we require more a priori knowledge of the world and confidence in the models used. The lack of complete knowledge means more uncertainty and contingencies to consider. This limitation is

quickly encountered when trying to model driving scenarios in probabilistic domains. In order to generate a finite horizon policy offline, we need to make assumptions about driver models for many different vehicles, assumptions that could prove wrong after additional observations are made.

The third limitation, ignoring search complexity, is the way that offline execution risk is calculated, which can allow low-probability high-risk actions to be included in our policy. RAO* can essentially discount future risk in search spaces with high branching factors. This side effect can be illustrated with the simple example defined in Section 2.1 of a robot planning in icy conditions. Using RAO* to solve this problem, there exist transition probabilities and chance constraints that result in a policy of moving right twice for the most likely outcomes. For example, if the relevant transition probabilities are $Pr(center|start, moveRight) = Pr(goal|center, moveRight) = 0.8$ and $Pr(fire|center, moveRight) = 0.1$, and the chance constraint on reaching the fire "Game Over" state is 0.09, then RAO* will result in a policy to move right twice. Before execution, the risk of reaching the fire state is less than the chance constraint, sitting at 0.08. However, after executing the first action of *moveRight*, the most likely next state is *center*, where the offline policy wants us to *moveRight* again though execution risk is now 0.1. This disconnect between offline execution risk calculations and real risks during execution only gets worse as the branching and depth of the policy increases, sometimes resulting in a 100% probability of violating a constraint when the system moves down the most likely path. This probabilistic behavior is not a flaw in RAO* and meets its definition of execution risk. Simulated 1000 times, the scenario will violate the chance constraint approximately 80 times because the execution risk is 0.08. While those edge cases might only occur with the low probability associated with their offline execution risk, it requires us to depend entirely on our a priori probabilistic models being accurate.

The approach we are presenting with iRAO* reduces these concerns by making incremental planning updates online, which can be used in a receding horizon fashion, executing the first action in a policy before expanding the planning horizon to look further ahead. The transition to an online risk-aware conditional planner with

receding horizon addresses the concern of model confidence, allowing us to incorporate observations and intent recognition results in subsequently expanded policies. The incremental algorithm repairs the policy during execution to keep risk bounds updated, especially as we move towards less likely branches.

Chapter 3

Incremental Risk-Aware AO*

"A plan is only a scenario, and almost by definition, it is optimistic...
As a result, scenario planning can lead to a serious underestimate of the risk of failure."

DANIEL KAHNEMAN, *Economics Nobel Laureate*
The psychology of judgment and decision making

At this point we are able to compile conditional plans that meet our safety requirements, maintaining tractability by pruning risky and unlikely scenarios. However, these plans are only as good as the prior information, and our plans are generated offfine, before anything happens. This chapter, the main contribution of the thesis, describes how to continuously refine the conditional plan based on new information using the iRAO* algorithm. After discussing the ideas behind the approach, the steps of iRAO* are outlined and the algorithm is shown to be complete and optimal for the defined problem. Finally, we propose an online executive for using iRAO* in the wild.

3.1 Inspiration

The thesis goal thus far has been to plan for the uncertainties of driving with POMDPs and risk-bounded conditional planning. To this end, the previous chapter defined the execution risk of a policy and motivated the use of RAO* in the previous sections

[28, 27].

But as an offline planner, RAO* has limitations when applied to autonomous driving applications. The biggest concern was complexity, as the branching for partially observable domains is notoriously large and any partially observable multi-agent approach (see Chapter 4) risks being intractable. Finite-horizon POMDPs have been shown to be PSPACE-complete, while infinite-horizon POMDPs are actually undecidable [21, 17]. Due to the complexity, there have been many online and approximate methods developed for solving POMDPs, including Monte-Carlo [24, 30]. Online search approaches are attractive because they have shown very good results planning over shorter horizons or making approximations while meeting real time constraints. They have the advantage of knowing the current situation and using that information to limit the number of possible contingencies. Many online approaches can also handle dynamic environments, where unexpected changes could occur, which would require offline planners to recompute a full policy. Unfortunately, many approximate or sampling-based online approaches would not be able make the guarantees on risk bounds that we are interested in.

Looking for inspiration from similar problems not necessarily in conditional planning, we identify two works that resonated with our own problem. Mixed integer linear programs (MILPs) have been used extensively in trajectory optimization problems to plan around collision constraints. Receding horizon approaches are often used in MILP path planning to keep the number of variables tractable, however there are cases where dead-ends occur beyond the horizon, which causes the planner to fail at the dead-end. To address the concern of these dead-ends, a "Safe Receding Horizon" approach was developed for autonomous vehicle path planning [29]. At each planning horizon there is also a second MILP encoding that solves for a backup trajectory to a safe state within the horizon. While conditional planning accounts for all possible contingencies defined in a model, this approach addressed the unexpected, beyond the horizon, by always having a backup plan that was dynamically feasible. As a result, the work in this thesis assumes that a receding horizon iRAO* approach would plan far enough ahead to identify a constraint dead-end and reach a safe state. The online

updates to the previous policy at each step allow us to exhibit these behaviors, where if a new constraint appears at our horizon, the updated policy reflects the new risks.

The second inspiration was the D* Lite algorithm which is an incremental heuristic search often used in motion planning with unknown environments. Based on Lifelong Planning A*, D* Lite gracefully handles unexpected constraints by focusing it's replanning to local inconsistencies [16]. This approach was motivation for how iRAO* implements focused incremental updates at each execution step while avoiding unnecessary computation. Both take advantage of the fact that most of the plan or policy is unaffected by a localized change in risk, so much of the previous plan is untouched. However D* Lite is for planning in dynamic environments, so it was also motivation for the dynamic iRAO* proposed at the end of this thesis.

3.2 iRAO*

In this section, we present the incremental RAO* (iRAO*) algorithm that takes an online approach to the original RAO* by making updates to the policy during execution. The algorithm is detailed in Algorithm 3.1, which uses modified *expand-graph* and *update-policy* functions from RAO*, as well as the *risk-update* defined in Algorithm 3.2.

3.2.1 Online Execution Risk

Beyond the state risk $r(b_k)$ and policy-wide execution risk $er(b_k|\pi)$ used in RAO*, we now need to define a local execution risk $ler(b_k|\pi_k)$. As a review, the state risk $r(b_k)$ is the likelihood that a belief b_k is violating a chance constraint at time k, shown in Equation 3.1. The execution risk of a policy is based on the current state risk of b_k and the policy π, shown in Equation 3.2. The Pr^{sa} and b^{sa}_{k+1} denote the transition to safe states from the current belief at b_k. I should write another sentence or two explaining why this important.

$$r(b_k) = \sum_{s \in \mathcal{S}} r(s_k) b(s_k) \tag{3.1}$$

$$er(b_k|\pi) = r(b_t) + (1 - r(b_t)) \sum_{b_{k+1}^{sa}} Pr^{sa}(o_{k+1}|\pi(b_k), b_k) er(b_{k+1}^{sa}|\pi) \tag{3.2}$$

The local execution risk $ler(b_k|\pi_k(b_k), b_{k+1})$ is the immediate risk of taking the next action in the policy from belief b_k, excluding the resulting belief state b_{k+1}, as shown in Equation 3.3. This quantity is important for online iRAO* so that the execution of risky actions is not overlooked in future planning. By risky actions, we are referring to actions where the next possible states violate the chance constraints, as opposed to risky policies where the violations could be multiple steps down the plan. The Equation for local execution risk is analogous to execution risk but without the recursion through the whole planning horizon h. Keeping track of executed local execution risks prevents 'spent' risk from being reallocated in future in incremental updates.

$$ler(b_k|a_k) = r(b_t) + (1 - r(b_t)) \sum_{b_{k+1}} Pr(b_{k+1}|b_k, a_k) r(b_{k+1}) \tag{3.3}$$

3.2.2 Algorithm

The iRAO* algorithm was designed and written to be used online, for example, with a system executive that is acting on resulting policies and taking observations to update posterior beliefs. This main algorithm is for the CC-POMDP model with a finite planning horizon, but there is a receding horizon extension presented after this section. The first call of the algorithm only requires the model H and initial belief b_0. The initial spent risk value sr_0 should be zero. Line 2 is checking for this initial case, where the input explicit graph G and policy π do not exist. If that is the case, then $iRAO*$ proceeds to line 9 to initialize the explicit graph G_k and policy pi_k to contain only the initial belief b_0. Lines 9-12 are the same as RAO*, building the initial search graph from scratch as expected from an offline planner.

After the first iteration and action, the inputs to $iRAO*$ will include the previous

Algorithm 3.1 incremental-RAO*

Input: CC-POMDP H, explicit graph G_{k-1}, policy π_{k-1}, spent risk sr_{k-1}, belief b_k.
Output: Explicit graph G_k, optimal policy π_k, and spent risk sr_k for time k.

1: **function** *iRAO**$(H, G_{k-1}, \pi_{k-1}, sr_{k-1}, b_k)$
2: $G_k \leftarrow G_{k-1}, \pi_k \leftarrow \pi_{k-1}$
3: **if** G_k and $\pi_k \neq \emptyset$ **then**
4: $sr_k \leftarrow sr_{k-1} + ler(G_{k-1}.root|\pi_{k-1}(G_{k-1}.root))$
5: $\theta \leftarrow G_k.likelihood(b_k))$
6: set $G_k.root$ and $\pi_k.root$ to b_k, remove nodes $n \neq b_k$ non-descendants of b_k
7: $G_k, \pi_k \leftarrow$ *risk-update*$(G_k, \pi_k, b_k, \theta, sr_k)$
8: **else**
9: G_k and π_k initialized with b_0
10: **while not** *termination-condition*(H, b_k, π_k, sr_k) **do**
11: $n, G_k \leftarrow$ *expand-graph*(G_k, π_k, sr_k)
12: $\pi_k \leftarrow$ *update-policy*(n, G_k, π_k, sr_k)
13: **return** G_k, π_k

1: **function** *termination-condition*(H, b_k, π_k, sr_k)
2: *leaf-risk* = sum of non-terminal leaf likelihoods in π_k
3: **if** $er(b_k|\pi_k) +$ *leaf-risk* $< \Delta - sr_k$ **then**
4: **return** `True`
5: **return** `False`

explicit graph G_{k-1}, policy π_{k-1}, spent risk sr_{k-1}, and the current *posterior* belief b_k. The *iRAO** algorithm assumes that the posterior belief update, a direct result of the observation and observation function as outlined in Equation 2.12, is done outside the search. An example executive that handles this correctly is presented in Section 3.3.

The local execution risk $ler(b_{k-1}, \pi_k(b_k))$, from the previous root based on the action taken (from the previous policy), is calculated in line 4. This local execution risk from the last step is added to the spent risk value sr_k to prevent it from being reallocated into the policy again. The realized likelihood θ from line 5, the probability that the current posterior belief occurred after the previous belief, is important for the forward *risk-update*. Line 6 is a purge of the explicit graph G and policy π that updates the root and removes all nodes that are no longer useful. This purge can be done through a recursive function that starts with the previous root, calls the function on each child if not b_k, and then deletes the node. If the explicit graph has

nodes with multiple parents, then this simple purge approach will need to be modified to only remove those nodes that are not also descendants of b_k. It is important that the realized likelihood was calculated before updating the roots and removing the necessary likelihood information.

Algorithm 3.2 risk-update

Input: Explicit graph G, policy π, belief b_k, realized likelihood θ, spent risk sr_k.
Output: Updated explicit graph G' and policy π'.

1: **function** *risk-update*(G,π,b_k,θ,sr)
2: $Q \leftarrow \{b_k\}, G' \leftarrow G, \pi' \leftarrow \pi$
3: **while** $Q \neq \emptyset$ **do**
4: $n \leftarrow$ remove(Q) node n with no descendant in Q
5: scale likelihood $G'.likelihood(n) = G.likelihood(n) * \frac{1}{\theta}$
6: **if** $(likelihood(n) * er(n) > \Delta - sr)$ **then**
7: mark n as `risk-violation`
8: **else**
9: add children(n) to Q
10: $W \leftarrow$ set containing `risk-violation` nodes within π'
11: **for** n in W **do**
12: $\pi' \leftarrow$ *update-policy*(n, G', π')
13: **return** G', π'

The *risk-update* function (Algorithm 3.2) performs a forward check on the new graph G_k and policy π_k to find if the policy still meets the chance constraints given the recent posterior b_k. The forward check for identifying nodes that are now a `risk-violation` is a calculation based on execution risk and likelihood values saved in the graph on previous searches. In lines 5 and 6, the likelihood values are scaled with the realized likelihood θ, then the product with execution risk from n is compared to the overall chance constraint Δ after removing the spent risk sr. If the chance constraint is violated in this lower bound estimation, then we mark the node n as a `risk-violation` and do not waste time investigating the children of n.

The key insight here is that no incremental backup is required to check for violating sections of the policy at the new execution step π_k. Instead, all necessary values can be updated top down, taking advantage of calculations from previous iterations. When performing the forward check, all of the belief states in the policy already took into account the local risk of the first action, which is the quantity removed from the chance

constraint before passing the error bounds to children. This recalculation only occurs in *update-policy*, which investigates from node n back up to the root for risk bounds and optimal actions. We show in Section 3.2.4 how this approach is valid. This helps to make incremental inspections of the policy and the identification of risky actions faster than calling *update-policy* on all of the leaf nodes. If the policy does not violate any constraints, then there is no search necessary and *iRAO** returns the new graph and policy. This happens because none of the nodes are marked as `risk-violation` and therefore *update-policy* is not called. In contrast, when nodes are labelled with `risk-violation`, then*update-policy* is called on each of them in order to choose new actions that meet the chance constraints.

The functions of *expand-graph* and *update-policy* are included again in Algorithms 3.3 and 3.4 for use in *iRAO**. The two functions retain the same overall behavior except that new values need to be stored and the execution risk bound computations include spent risk.

The changes to *expand-graph* are to save the likelihood and execution risk of each child in the graph. While this was most likely done in RAO* implementations, we added it explicitly here because they are important values for the forward *risk-update* calculations. The way that execution risk $er(c)$ for each child is calculated remains the same, because the spent risk is removed from the root node in *update-policy*. The *expand-graph* function is never used on the root after the first expansion of the search, which started with a spent risk of zero.

The *update-policy* function now needs to take into account the spent risk. Instead of changing the original Δ value, we keep it and subtract spent risk from the root execution risk bound, which is then propagated to children in the policy. The new execution risk bound in Equation 3.4 includes the spent risk when recalculating for the root belief state.

$$\Delta'_{k+1} = \frac{\frac{\Delta - sr - r_b(b_k)}{1 - r_b(h_k)} - \sum_{o_{k+1} \neq o'_{k+1}} Pr^{sa}(o_{k+1}|\pi(b_k), b_k) h_{er}(b^{sa}_{k+1}|\pi)}{Pr^{sa}(o'_{k+1}|\pi(b_k), b_k)} \tag{3.4}$$

Algorithm 3.3 expand-graph

Input: Explicit graph G, policy π.
Output: Updated graph G', expanded leaf node n.

function *expand-graph*(G, π)
  $G' \leftarrow G$, $n \leftarrow$ most-likely-leaf(G, π)
  for each action a available at n **do**
    $ch \leftarrow$ expand-children(n, a) (2.11), (2.12), (2.13) and save *likelihood*(c)
    $\forall c \in ch$ estimate and save execution risk $er(c)$ with (2.12), (2.13)
    $\forall c \in ch$ compute execution risk bounds Δ_c
    if no $c \in ch$ violates $er(c) > \Delta_c$ **then**
      $\forall c \in ch$ estimate value $Q^*(c)$ with (2.19)
      $G' \leftarrow$ add ch to graph with hyperedge with Δ_c's and likelihoods
  if no hyperedge added to n **then**
    mark n as `dead-end-terminal`
  return G', n

Algorithm 3.4 update-policy

Input: Expanded n, explicit graph G, policy π, spent risk sr.
Output: Updated policy π'.

function *update-policy*(n, G, π, sr)
  $\pi' \leftarrow \pi$, $Z \leftarrow$ set containing n and its ancestors in π
  while $Z \neq \emptyset$ **do**
    $n \leftarrow$ remove(Z) node n with no descendant in Z
    if n is root **then**
      update execution risk bound Δ_n with (3.4)
    else
      update execution risk bound Δ_n with (2.18)
    while $\exists$ available actions at n **do**
      $a \leftarrow$ next best action at n (2.20) that satisfies execution risk bound
      Propagate execution risk bound of n to the children of hyperedge (n, a)
      if no children violates its new bound **then**
        $\pi'(n) \leftarrow a$; **break**
    if no action selected at n **then**
      mark n as `dead-end-terminal`
  return π'

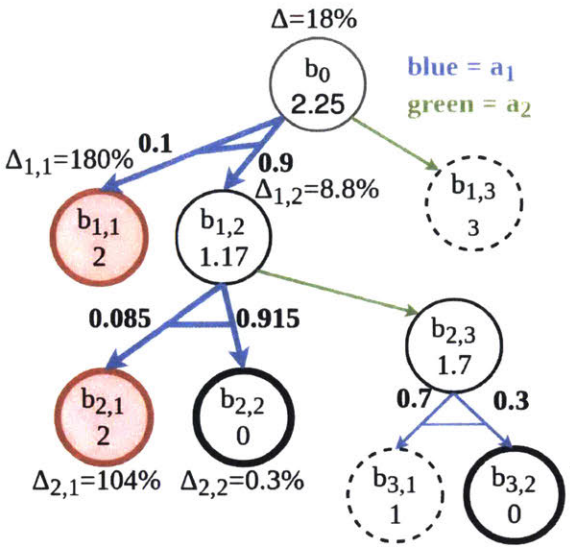


Figure 3-1: Previous G_{k-1} and π_{k-1} given as input to *iRAO**.

3.2.3 Grounded Example

For understanding how the iRAO* algorithm works, this section presents an example with incremental updates to a risk-bounded conditional plan. We are building off of the example from Chapter 2 that was solved with RAO* as an offline planner. All of the shapes and colors of the diagrams are also detailed in Section 2.4.4 for reference.

In this grounded example, we are given the inputs $H, G_{k-1}, \pi_{k-1}, sr_{k-1}, b_k$, which are the CC-POMDP model, previous explicit graph, previous policy, past spent risk, and the new posterior belief, respectively. The first explicit graph seen in Figure 3-1 is G_{k-1} from the previous planning step with π_{k-1} marked as the thicker hyperedges which show that $\pi_{k-1}(b_0)$ was action a_1 and $\pi_{k-1}(b_{1,2})$ is also action a_1. The input spent risk sr_{k-1} is zero and the new posterior belief $b_k = b_{1,2}$.

The first steps are to update the spent risk and find the realized likelihood. Using Equation 3.3, the local execution risk of the previous root given the action taken to get to $b_{1,2}$ was $ler(b_0|a_1) = 0.1$. The realized likelihood θ is pulled directly from *likelihood(c)* which was saved by *expand-graph* when the node was added to the graph. These steps are shown in Figure 3-2. Once that information is gathered from the old graph, we purge the old root and sibling branches of b_k that are no longer relevant to

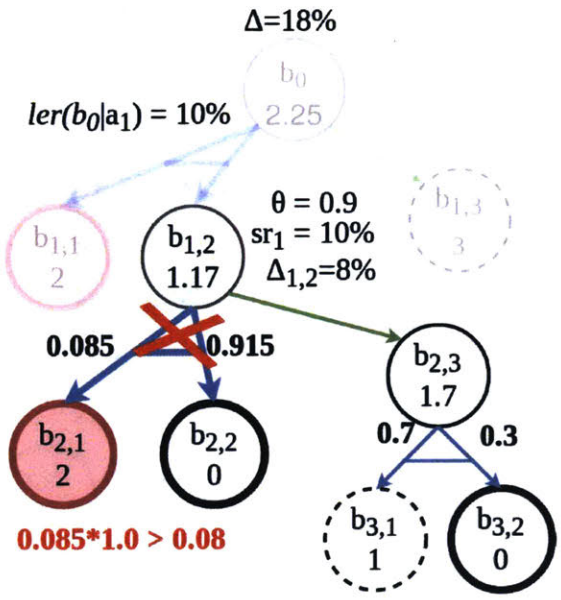


Figure 3-2: First steps of *iRAO**: Updating spent risk, finding realized likelihood, purge of unneeded nodes, and marking $b_{1,2}$ as `risk-violation`.

the search. In this example, the nodes we remove are b_0, $b_{1,1}$, and $b_{1,3}$, but this would also include all of the descendants of $b_{1,2}$ and $b_{1,3}$ that were not also descendants of $b_{1,2}$. The root of both graph and policy are updated to be $b_{1,2}$, shown in Figure 3-2 as the grey shaded circle.

Switching now to *risk-update*, we traverse down the policy, starting with the root, and scale the likelihood of each node before checking it against the overall chance constraint as shown in line 6. In this case, the root $b_{1,2}$ is updated to a likelihood of 1 and compared to the chance constraint $\Delta = 0.18$ after subtracting the spent risk $sr = 0.1$. For $b_{1,2}$ we get the following: $1.0 * er(b_{1,2}) > 0.18 - 0.1$ and $0.085 > 0.08$, so the node is marked as `risk-violation`. No additional nodes are investigated by *risk-update* because children of those in violation are not added to Q. Our set W just contains $b_{1,2}$ so we then we use *update-policy*$(b_{1,2}, G_k, \pi_k)$ to get the new policy.

The Z in *update-policy* is the set $\{b_{1,2}\}$ because it has no ancestors in π. The new execution risk bound for $b_{1,2}$ is calculated using Equation 3.4 since it is the root of the policy, which directly takes into account the spent risk of $sr = 0.1$. The resulting value is shown as $\Delta_{1,2} = 8\%$ in Figure 3-2, reaffirming that the action a_1 with a

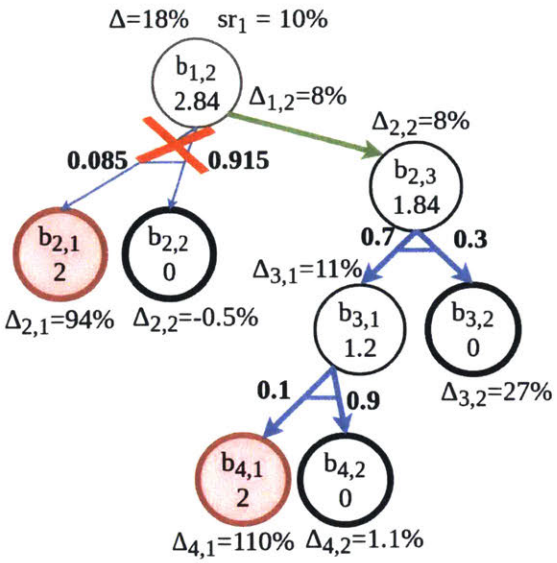


Figure 3-3: Result of iRAO* performing *update-policy* at $b_{1,2}$ and then *expand-graph* at non-terminal leaf $b_{3,1}$.

8.5% risk is not acceptable. The only other available action is a_2 which satisfies the execution risk bound with $er(b_{1,2}|a_2) = 0$ so far. We propagate the execution risk bounds to the children such that $\Delta_{3,1} = 11\%$ and $\Delta_{3,2} = 27\%$. The belief $b_{3,1}$ is non-terminal with $h_{er} = 0$, noted by the lack of red shading, and $b_{3,2}$ is terminal with $r(b_{3,2}) = 0$. None of the children violate the new bounds so we update the policy with $\pi_k(b_{1,2}) = a_2$.

Finally, the last steps of *iRAO** include alternating between leaf node expansions and policy updates (lines 10-12), just like offline *RAO**. With this updated policy, $b_{3,1}$ is now a non-terminal leaf that needs to be expanded. The *expand-graph* function computes the two children beliefs of action a_1, including their likelihoods, execution risk $er(c)$, and execution risk bounds Δ_c. Both children are terminal so the execution risk calculations use the risk $r(c)$ for each instead of the risk heuristic $h_{er}(c)$. Belief $b_{4,1}$ takes some of the risk bound away from $b_{4,2}$ but neither are violated, the Q values are calculated and the hyperedge for action a_1 is added.

The *update-policy* function then has $Z = \{b_{3,1}, b_{2,3}, b_{1,2}\}$ to check. Starting at the deepest belief, we have trivial action selections for $b_{3,1}$ and $b_{2,3}$ which is a_1 for both. When there are multiple actions to select from, we use Equation 2.20 to decide the

best one. Adding these two actions to the policy also updates the estimated value Q for each of the beliefs, where $b_{3,1}$ is now 1.2 and $b_{2,3}$ is now 1.84 . This is because we are still assuming unit costs for all actions, the same as the RAO* example. Refer to the example of RAO* in Section 2.4.4 for more discussion and details on the *update-policy* function. The final belief in Z to update is the new root $b_{1,2}$, for which we select a_2 since a_1 violates the risk bounds, and this results in a value of 2.84 for the new policy.

3.2.4 Complete and Optimal

This section sketches the proof to show the completeness and optimality of the iRAO* algorithm as it is presented in this thesis. The alternation of expanding and updating with the functions *expand-graph* and *update-policy* have already been shown to be complete and optimal by the original or RAO* [28, 27]. Therefore, the discussion here focuses on the processes that are unique from the original RAO* search.

Lemma 3.1. *Using the realized likelihood of the new belief state root b_k to update descendants in the policy is sound.*

Proof. This first lemma comes directly from how likelihood in a policy tree is calculated. The likelihood of the root b_0 is 1.0, because that is the initial state provided. The children of the best action at b_0 all have transition probabilities derived from Equations 2.11, 2.12, and 2.13. It is this transition probability that we pull out of the previous explicit graph when find realizing likelihood θ of a new posterior belief. The likelihood of each child occurring is the likelihood of the parent belief multiplied by the transition probability. The deeper into the tree, the more probability terms are discounting the likelihood. When we make the new posterior belief the root, scaling all of it's descendants by $\frac{1}{\theta}$ cancels out the first term that it contributed in their likelihood calculations. □

Lemma 3.2. *The likelihood of a belief b_i multiplied by $er(b_i|\pi_{k-1})$ is an lower bound on the execution risk given the new posterior belief b_k.*

Proof. The execution risk of each node, $er(n|\pi)$, in the explicit graph was calculated through Equation 2.16 and was saved during the previous incremental plan when n was expanded and updated in the policy. Assuming the CC-POMDP model is constant, then the only change to execution risk calculation for a node is the likelihood that the belief state occurs in execution. This change in likelihood is a function of the new belief state b_k, also called the realized likelihood θ. All of the nodes n descendants of b_k were previously discounted by *theta* the likelihood of b_k. Now that b_k is the current belief state we scale all descendants by $1/\theta$ to quickly update the execution risk without the need for backups.

We can write a simplified execution risk using *expected-risk* in Equation 3.5. This *expected-risk* equation ends up being equivalent to the expected risk that bounds the search in C-POMDPs [22, 32]. We can show that this is explicitly less than or equal to the execution risk from Equation 3.2, because it does not use the definition of safe distributions, and it is easier to work with in the forward propagation of likelihood updates.

$$\textit{expected-risk}(b_k|\pi) = r(b_t) + \sum_{b_{k+1}} Pr(o_{k+1}|\pi(b_k), b_k) er(b_{k+1}|\pi) \tag{3.5}$$

This means that if the new execution risk $er(b_{k+i}|\pi_k)$, for b_{k+i} that are descendants of the new posterior b_k, violates the chance constraints then it will also exceed the likelihood update we do with *expected-risk*.

□

Lemma 3.3. *The labelling of nodes as* `risk-violation` *in risk-update is sound.*

Proof. Directly follows from Lemma 3.2 and Lines 6-7 of Algorithms 3.2. Only beliefs that have *expected-risk* for descendants of b_k to exceed execution risk to exceed □

Theorem 3.4. iRAO* is complete and produces the optimal, finite-horizon policies without violating the chance constraints.

Proof. Offline, the iRAO* algorithm is identical to the original RAO*. For the incremental updates, *risk-update* labels beliefs in the policy that could be in violation of

the new chance constraint after updating likelihood and removing spent risk. We then recalculate the execution risk for nodes labelled `risk-violation` in *update-policy*, as well as their ancestors in the policy up to the root. This has the same guarantees on pruning as it did in RAO*. New actions that lead to non-terminal leafs are then expanded with *expand-graph* and then the policy is updated until we have only terminal leafs again.

Put another way, iRAO* has the same effect as using *update-policy* on all leaf nodes of the policy. The optimal actions that meet the chance constraints will remain while those now in violation will be expanded and searched. Instead of doing that on each increment, the forward *risk-update* finds beliefs that are possibly in violation to focus updates on them. □

3.3 Integration with Executive

This section presents an example executive that can be used with iRAO* search. The executive performs functions that move the agent through policy execution, including acting, observing, and updating posterior belief based on observations.

Algorithm 3.5 Example Executive

Input: CC-POMDP H, initial belief b_0.
Output: None: execution to goal or through planning horizon h.

1: **function** *exampleExecutive*(H, b_0)
2: $G \leftarrow \{\}, \pi \leftarrow \{\}, sr \leftarrow 0, b \leftarrow b_0$
3: **while** not at goal or done finite horizon **do**
4: $G', \pi', sr' \leftarrow iRAO(H, G, \pi, sr, b)$
5: execute action $\pi'(b)$
6: collect observation o
7: $b =$ *measurement-update*(b, o)
8: $G \leftarrow G', \pi \leftarrow \pi', sr \leftarrow sr'$

An executive using $iRAO*$ needs to initialize the search as shown in Algorithm 3.5 and then call for incremental updates after online observations are used to update the posterior belief state. Executing the action specified by the current policy and collecting observations, lines 4 and 5, are application specific. The *measurement-update* function uses the observation functions in the CC-POMDP model H to calculate the

new posterior belief based on the previous belief and newest observation.This process is repeated until execution to the horizon or a goal is completed.

3.4 Advantages

This section is complementary to the iRAO* motivations, discussing how the algorithm addresses the concerns pointed out with offline conditional planning. While this discussion is qualitative, there are quantitative results presented in Chapter 6.

First and foremost, online incremental updates to the policy allow our actions to remain safer throughout execution, regardless of which branches are traversed. As shown in Section 2.4.5, executing an offline policy can result in high risk actions in low-likelihood branches. Although that behavior meets the definition of offline execution risk, we should expect more of our agent or planning system. With the iRAO* algorithm, we are able to utilize observations and the previous planning results to repair and update the policy during execution.

Second, with an online algorithm, we can now apply risk-aware conditional planning to much larger domains using a receding horizon approach, discussed in Section 3.5. This approach is attractive because it reduces the search complexity by building out the explicit graph with a limited depth search.

Finally, the ability to identify and repair the policy based on localized changes to risk opens the door to dynamic online risk-aware planning. Like D* Lite that performs local updates to the plan based on changes in the environment, iRAO* would be more powerful if it could repair the policy after risks change during execution. Going back to the problem from Section 2.1, with dynamic iRAO* the Colony-Bot would be able to plan around changing cells, for example if the *icy* or *gameOver* cells moved. In the example of autonomous driving, this could mean reacting to the addition of nearby vehicles in the middle of online planning. A part of Section 7.1 is dedicated to describing the dynamic iRAO* problem as well as proposing possible approaches.

3.5 Receding Horizon iRAO*

The *iRAO** algorithm will work as a receding horizon approach with a small change. The horizon is updated during each increment with either decrements to node depth or redefining which nodes are at the horizon. This step, in line 8 of Algorithm 3.5, updates the status of the leaf nodes in the policy π_k, forcing them to be expanded in the forward search for the next policy. For example, if the planner has a horizon of five actions, then after generating a policy of depth five and executing an action, all the leaf nodes in the new policy now only have depth four and switch from terminal to non-terminal.

Algorithm 3.6 Receding Horizon incremental-RAO*

Input: CC-POMDP H, explicit graph G_{k-1}, policy π_{k-1}, spent risk sr_{k-1}, belief b_k.
Output: Explicit graph G_k, optimal policy π_k, and spent risk sr_k for time k.

1: **function** $iRAO^*(H, G_{k-1}, \pi_{k-1}, sr_{k-1}, b_k)$
2: $G_k \leftarrow G_{k-1}, \pi_k \leftarrow \pi_{k-1}$
3: **if** G_k and $\pi_k \neq \emptyset$ **then**
4: $sr_k \leftarrow sr_{k-1} + ler(G_{k-1}.root|\pi_{k-1}(G_{k-1}.root))$
5: $\theta \leftarrow G_k.likelihood(b_k))$
6: set $G_k.root$ and $\pi_k.root$ to b_k, remove nodes $n \neq b_k$ non-descendants of b_k
7: $G_k, \pi_k \leftarrow$ *risk-update*$(G_k, \pi_k, b_k, \theta, sr_k)$
8: check and update terminal/nonterminal status of all leaf nodes in π_k
9: **else**
10: G_k and π_k initialized with b_0
11: **while** nonterminal leaf nodes in π_k **do**
12: $n, G_k \leftarrow$ *expand-graph*(G_k, π_k, sr_k)
13: $\pi_k \leftarrow$ *update-policy*(n, G_k, π_k, sr_k)
14: **return** G_k, π_k

Unfortunately, this also means that the safety guarantees are only valid to the horizon. The heuristic forward search with limited look-ahead could result in being trapped in a dead-end with no safe options. This is a case where having a backup plan, seen in the MILP receding horizon approach [29], might be necessary to combat dead-ends that could be encountered in subsequent horizons. We include recommendations in Section 7.1 for future work with this idea of handling the uncertainty beyond the horizon by encoding a second backup plan.

3.6 Multiple-step iRAO*

The algorithm will also work in applications where multiple steps are taken in between incremental replanning. This means the input to *iRAO** is not necessarily from $t = k - 1$ but could be G_{k-2} or G_{k-3}, for example. This would occur in cases when there is not an option to preform replanning between all sequential actions. One example of this is when observations cannot be made after every action in the policy.

Algorithm 3.7 Multiple-Step incremental-RAO*

Input: CC-POMDP H, explicit graph G_{k-i}, policy π_{k-i}, spent risk sr_{k-i}, belief b_k.
Output: Explicit graph G_k, optimal policy π_k, and spent risk sr_k for time k.

1: **function** $iRAO^*(H, G_{k-i}, \pi_{k-i}, sr_{k-i}, b_k)$
2: $G_k \leftarrow G_{k-i}, \pi_k \leftarrow \pi_{k-i}$
3: **if** G_k and $\pi_k \neq \emptyset$ **then**
4: *actions* $\leftarrow$ identify the i actions from $\pi_{k-i}.root$ to b_k
5: $sr_k \leftarrow sr_{k-i}+$ (sum of *ler* for beliefs from $\pi_{k-i}.root$ to b_k taking *actions*)
6: $\theta \leftarrow G_k.likelihood(b_k))$
7: set $G_k.root$ and $\pi_k.root$ to b_k, remove nodes $n \neq b_k$ non-descendants of b_k
8: $G_k, \pi_k \leftarrow$ *risk-update*$(G_k, \pi_k, b_k, \theta, sr_k)$
9: **else**
10: G_k and π_k initialized with b_0
11: **while** $\exists$ non-terminal leaf nodes in π_k **do**
12: $n, G_k \leftarrow$ *expand-graph*(G_k, π_k, sr_k)
13: $\pi_k \leftarrow$ *update-policy*(n, G_k, π_k, sr_k)
14: **return** G_k, π_k

The pseudo code requires a few small changes, as shown in Algorithm 3.7. We need to update the spent risk calculation to include all the actions taken since the last incremental step, described in lines 4 and 5. This can be directly drawn out of the policy π_{k-i}, where i is the number of actions taken since the last iRAO* iteration, and we identify the actions and beliefs between the old root and the new posterior belief b_k. The likelihood update and graph purge in lines 6 and 7 work the same as before.

Chapter 4

Hybrid Multi-Agent Models

After describing the CC-POMDP, reviewing RAO* for offline planning, and introducing iRAO* for online applications, this chapter develops a modeling framework for the QAD problem. The following sections build up the hybrid multi-agent model one piece at a time and then present examples of modeled driving scenarios.

4.1 Modeling Qualitative Autonomous Driving

We begin with important naming conventions and then identify some of the modeling challenges that are inherit to QAD problems. We call our vehicle, the autonomous car that we are planning maneuvers for, the Ego Vehicle. We refer to nearby uncontrollable vehicles, those adding uncertainty to our planning, as the Agent Vehicles. Additionally, while we are calling this a multi-agent model, this approach is not to be confused with Multi-Agent Planning (MAP), as in the end we are only interested in the plan for our Ego Vehicle. We are not assuming that agents have the same goals or reward functions with regard to collaboration.

The problem of modeling autonomous driving scenarios is complex and can be broken into various specific challenges. We have identified four major challenges here. First, there is the need to discretize states at some level for our planner to reason over and condition on in the policy. Autonomous driving is not a game that can be modeled with players taking turns, observing others' choices before choosing

our own action. Instead, the QAD problem is dynamic and continuous,requiring that the extraction of discrete states be sufficient in accounting for all that is occurring in any given traffic scenario.

Second, there is the challenge of modeling driving maneuvers as actions that can be used in planning. The conditional planners presented in this thesis are designed to reason over CC-POMDPs with discrete actions. However, these driving actions also need to represent the underlying continuous motion planning problem and be diverse enough to model real driving dynamics. In the case of heuristic forward search, the modeled actions need estimates on value, as well as costs (or rewards) to decide optimality. The framework for modeling driving maneuvers will also ultimately be used for calculating risk estimates, which are needed to guide the search and make safety guarantees.

The third challenge is modeling the behaviors of multiple agents with regards to prediction and planning ahead. The actions of nearby drivers, which are out of our control, are functions of different driving styles, goals, and ultimately their own local surroundings. Being able to estimate driver styles or classifying behaviors online could help predict future maneuvers. This also means being able to model road geometry, incorporating the driving constraints and conditions into our own decisions and predicting Agents' behaviors.

And finally, there are rules of the road that need to be taken into account and respected when making any decision. Often learned by humans from watching others, attending driver education, or reading a manual (albeit to varying degrees of success), the rules constraining driving behaviors must be encoded in our Ego Vehicle so that planned maneuvers are legal and predictable.

4.2 Chance-Constrained Multi-Agent POMDP

The next sections present how our model addresses the above challenges. As an overview: we develop *agent-models* to represent the vehicles in a scenario, each of which has a library of *action-models* that describe available driving maneuvers. At

each decision epoch, we represent the scenario with a belief over possible *composite states*, which include discrete and continuous states of all involved vehicles as well as hidden states for the driver profiles of Agent vehicles. The agent-models, initial belief, and a model of the road conditions are combined into an extended version of the CC-POMDP. The result is a hybrid, multi-agent, chance-constrained model for the Qualitative Autonomous Driving problem, called a *QAD-POMDP*:

Definition 3. A *QAD-POMDP* is the tuple $QAD = \langle Ego, Agents, road, b_0, h, \mathcal{C}, \Delta, \rangle$, where

- *Ego* is the *agent-model* (defined in Section 4.2.4) for our Ego Vehicle, including all available *action-models* (defined in Section 4.2.2);
- *Agents* is the set of *agent-models*, one for each nearby Agent Vehicle in the traffic scenario, including the hidden state of driver profile;
- *road* is a model of the road constraints, including lanes and speed limits;
- b_0 is the initial belief *composite state* (defined in Section 4.2.1);
- h is the planning horizon;
- $\mathcal{C}$ is a set of q planning constraints over possible *composite states*;
- Δ is a vector of probabilities for q constraints that define the chance constraints.

The goal remains to find an optimal, deterministic, and chance-constrained policy π^* given a QAD-POMDP tuple, in this case formulated as minimum cost, such that:

$$\pi^* = \arg\min_{\pi} \mathbb{E}\left[\sum_{t=0}^{h} Cost(s_t, a_t) \middle| \pi\right]. \tag{4.1}$$

The main constraint of set $\mathcal{C}$ is the near-collision event of our Ego Vehicle with another Agent Vehicle. This is defined as a function of the continuous position variables for each agent model through time. The model presented here calculates the distance between our Ego Vehicle and each Agent Vehicle, to find the risk of near-collision, in

the discretization of the Probabilistic Flow Tubes themselves. This can be calculated differently if an alternative continuous motion representation is used instead.

4.2.1 Driving Scenes as Composite State

The choice of how to discretize the model is our first challenge, due to the planner needing to condition on specific belief states. Our solution is to discretize the hybrid model into a *composite state* at the time of our decision epoch. These *composite states* are basically a snapshot of all the variables that describe the driving scene at a timestep, specifically the time at which the Ego vehicle needs to begin a new action. With the *action-models* presented in this chapter, which have fixed equivalent durations, all of the agents' decision epochs are synchronized. This means that the *composite states* actually capture the timestep when all agents are ending actions and need to start another.

The states that we are interested in for each vehicle include continuous variables like position, orientation, velocity, and acceleration, as well as discrete qualitative variables like lane number and turn signal status. The state of the driver for each nearby Agent Vehicle, modeling factors like goals and driving style, is a hidden state. Say something about how we can use observations to infer the hidden state. Each *composite state* captures the continuous variables and hidden driver state for each vehicle as probability distributions.

We can represent the state of our multi-agent model at a given time t with the following Qualitative Autonomous Driving *composite state*:

Definition 4. A QAD *Composite State* is the tuple $c\text{-}state = \langle Ego, Agents, t \rangle$, where

- *Ego* is the state of our Ego Vehicle, including position, orientation, velocity,
- *Agents* is a list of the Agent Vehicle states.
- t is the time stamp for this state.

The models that describe the evolution of these *composite states* are developed in the following sections.

4.2.2 Maneuvers as Action Models

Modeling driving maneuvers as discrete actions is the second challenge. To address this, we use functional action representations similar to classical planning, with preconditions and effects. These *action-models*, shown in Definition 5, are the building blocks for the rest of the multi-agent hybrid model.

Each action-model has preconditions, effects, and a cost for general planning purposes, as well as additional attributes for domain specific plans. In our case of conditional planning for autonomous vehicles, each action-model also has a probabilistic flow tube (PFT) for collision risk calculations and the associated control inputs to execute the PFT.

Definition 5. An *action-model* is the tuple $A = \langle Pre, Eff, C, PFT \rangle$, where

- Pre: $S \rightarrow \{0, 1\}$ is the preconditions function.
- Eff: $S \rightarrow S$ is the effects function.
- C is the cost associated with the action.
- PFT probabilistic flow tube for the maneuver (more details in Section 4.2.3).

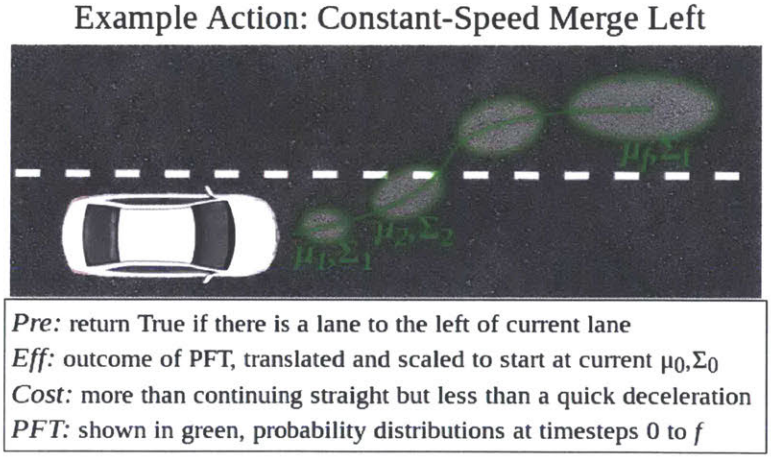


Figure 4-1: Example *action-model* for a smooth merging maneuver at constant speed.

A basic *action-model* example is shown in Figure 4-1 for a merging maneuver. The precondition function allows us to include coarse behavior models, like the fact that

a vehicle should not be executing a merge maneuver when there is no lane on that side. When we introduce the *agent-models* it becomes clear that these preconditions also help to prune the search space, where we only consider actions that meet the preconditions. The effects are derived from the PFT and are a function of the initial state distribution. The cost C associated with each *action-model* guides the heuristic forward search and makes the planner prioritize maneuvers accordingly. This example cost is written such that smooth driving is prioritized, changing lanes to the left would only occur if continuing forward at the current velocity was not an option.

4.2.3 Continuous Dynamics with Probabilistic Flow Tubes

For each *action-model*, we need to represent the continuous dynamics of the maneuver with uncertainties. We choose to model these continuous maneuvers with Probabilistic Flow Tubes (PFTs), which represent a set of continuous trajectories with common characteristics defined over a time interval $[t_0, t_f]$ [7, 8, 11]. Each PFT is characterized as a set of cross-sectional regions, where each cross section s_i stores the mean and covariance of the associated common trajectories at time t_i $(0 \leq i \leq f)$. An alternative approach is the use of funnels, which are presented as regions of finite-time invariance in [31], however they do not model probabilistic variances of the motions. In contrast, the PFTs work well for our probabilistic risk constraints on state, where the intersection of cross-sectional Gaussians is used to calculate collision risks.

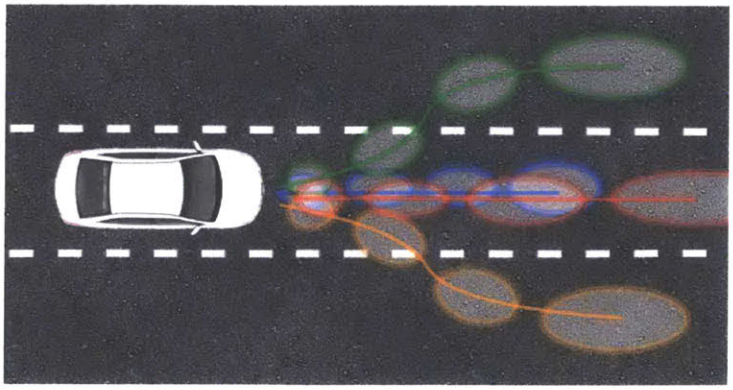


Figure 4-2: Visual representation of Probabilistic Flow Tubes from four example vehicle maneuvers.

PFTs can be generated from vehicle dynamics or demonstrated trajectories to model realistic human motions. Each *action-model* PFT is paramaterized on the initial distribution so that we can translate them to the initial mean positions μ_0 and scale them to the initial uncertainty Σ_0.

4.2.4 Vehicles as Agent Models

To address the challenge of planning safe maneuvers near multiple independent drivers, we use a separate agent model for each vehicle in a scenario. Having separate model instances allow us to propagate and predict driving behaviors based on estimated driver styles.

We model our Ego Vehicle and each nearby Agent Vehicle with a separate *agent-model*, which combines a belief about the driver with *action-models*. First, each *agent-model* is used to track and organize the hybrid belief state for the respective vehicle. This belief includes the fully observable continuous states like position and velocity as well as discrete states like lane number. The belief also includes the hidden state of driver style, for example a careful or aggressive driver, which is important for predicting the likelihood of future maneuvers. Each agent has a library of *action-models* with preconditions and effects, as defined in Definition 5. When we make predictions for the overall traffic scene, we query all of the *agent-models* for available actions, those where the current belief state meets the preconditions, and focus on those possible maneuvers. The observed maneuvers of each vehicle are then used to update the belief about the type of driver. The likelihood of an *agent-model* executing a given action is a function of the driver type hidden state. The *agent-model* is detailed in Definition 6 and the initial belief state for each vehicle is specified in the *composite state* b_0 in Definition 3.

Definition 6. (Agent Model) An agent model is a tuple $M = \langle x, x_D, \mathbf{D}, \mathbf{A}, obs, pmf \rangle$, where

- x is a set of continuous state variables.
- x_D is a set of discrete state variables, including the hidden state of driver type.

- **D** is the domain for x_d.
- **A** is a set of available *action-models* for the agent.
- *obs* is an observation function that maps driver type to the probabilities of *action-models*.
- *pmf* is a probability mass function calculating the likelihood that uncontrollable agents will execute an action in their belief state.

4.3 Rules of the Road

This section, addressing the final challenge, is a discussion of how driving rules and regulations can be encoded into the models. We present a handful of example driving laws and how our models can guide the Ego Vehicle to drive safely within legal limits. We are using the Massachusetts RMV Driving Manual for reference [20].

Posted speed limits, different based on the size of a road, restrict the maximum velocity that a vehicle should be driving on the given road. For our Ego Vehicle, *action-models* with velocities greater than the current speed limit have higher costs. This ensures that higher speeds are only used when there are no safe maneuvers at or slower than the speed limit. Obviously slowing to a stop is safe and legal but there are cases that accelerating out of a scenario is the safest option.

The manual includes many rules and suggestions for highway driving. We are instructed to "stay to the right and only use the left lane for passing or if other lanes are blocked". The example *action-model* in Figure 4-1 shows how maneuver costs can guide lane changing behaviors, in this case we only merge into the left passing lane when the other remaining option is to slow down below optimal speed. There is guidance not to "drive in another drivers blind spot" and if we must, to then "safely drive through the blind spot as quickly as you can". This constraint is actually captured in the maneuver prediction and risk calculation regarding nearby Agent Vehicles. Blind spots can be encoded into the preconditions of *action-models* in Agent Vehicle libraries such that they are allowed to merge when they think it is

clear. Our QAD-POMDP formulation will predict this behavior and the risks will be calculated accordingly, resulting in behavior from our Ego vehicle that avoids blind spots.

There are driving guidelines that can be handled by lower-level control within the autonomous driving architecture, for example "drive in the middle of your lane, staying between the lines". There are also many issues covered in the manual that are specific to human operators. These include regulating speed to be within the posted limits, visually checking blind spots, avoiding fatigue from driving for a long time, and planning to stop for stretching breaks.

4.4 Limitations

"All models are wrong but some are useful."

GEORGE E. P. BOX

Robustness in the Strategy of Scientific Model Building [6]

While we present promising results from the selected traffic scenarios in Chapter 6, there are obvious limitations to the models.

The first and biggest limitation of this model, inherent to conditional planning, partially observable domains, and multi-agent problems, is scaling. The computational complexity of planning with multiple independent agents grows very quickly. This is shown again in Section 5.2 with a grounded example. Smart modeling techniques discussed before, like thoughtful preconditions for *action-models*, can help with scaling by excluding actions that Agents could take given their current state. RAO*'s use of risk bounds to prune the search space helps with performance over AO* but those benefits are domain dependent, obviously doing more to prune the search in riskier scenarios. There are many other approaches to Multi-Agent Planning (MAP) that make approximations to prevent the exponential explosion [9], however it is the systematic search of RAO* and iRAO* that gives us our guarantees on optimality and safety for partially observable models.

Another limitation that we identify is the use of synchronized decision epochs for

the multiple agents. This assumes that the traffic scenario evolves like a turn-based game where all the agents start and end actions together. We discuss this more in Chapter 5 and address this limitation by incorporating durative actions.

Finally, the need to discretize the continuous motion problem at some level is another disadvantage to our models. Our approach with PFTs is limited to how much they can be scaled and transformed. Future development of the continuous maneuvers and how to represent them in discrete actions should be pursued and was not the primary contribution of this work.

Chapter 5

Durative Actions in Multi-Agent Planning

The final contribution of this thesis addresses one of the limitations of the driving models as presented in Chapter 4, specifically the assumption that the actions are all equivalent length and synchronized. This is an artifact of CC-POMDPs and their discrete state and action sets commonly representing single agent problems without temporal constraints. This chapter introduces *durative-action-models*, discusses how they are used in our multi-agent driving model, and presents two approaches for dealing with the combinatorial growth in complexity.

5.1 Asynchronous Decision Epochs

Timing is very important in driving. Being able to recognize situations and make good decisions is how drivers avoid accidents. This is most apparent in intersections, where two or more roads are crossing each other and the potential for conflict is higher. Though a small fraction of miles driven, intersections account for about 40% of all accidents [3]. Turning left and crossing over in intersections, when nearby vehicles are involved, produces dynamic situations, where synchronized actions cannot capture a realistic timing of events. Beyond autonomous driving, asynchronous durative actions are an important extension for planning in settings with any uncoordinated agents

acting in the world.

We begin by redefining the *action-model* to include durations, as shown in Definition 7. This thesis focuses on actions with known fixed durations, but we propose ideas in Section 7.1 to deal with uncertainty in the durations. Note, the Δt is actually redundant, as each new fixed duration is also captured by the *PFT*.

Definition 7. A Fixed-duration action model is the tuple $DA = \langle Pre, Eff, \mathcal{C}, \Delta t, PFT \rangle$, where

- *Pre* is the preconditions function.
- *Eff* is the effects function.
- $\mathcal{C}$ is the cost associated with the action.
- Δt is the action duration.
- *PFT* is the probabilistic flow tube for the maneuver.

5.2 Generating Action Sequences

The downside to planning with durative actions is the increased complexity in the number of contingencies to cover. Instead of conditioning on the number of possible individual actions that other agents could execute, we are now forced to consider the permutations of action sequences that could occur during each of our action durations. We begin with a brute force approach to the generation of action sequences, where these sequences are how we ultimately branch and condition on all possible action permutations, shown in Algorithm 5.1. These action sequences are a function of durations, as more shorter actions can occur while we execute one longer action.

As a motivating example, imagine we are planning maneuvers for our autonomous vehicle with one nearby Agent Vehicle that has 4 possible actions, as shown in Figure 5-1. The Agent's available actions are continue-forward, accelerate, merge-left and merge-right with durations of 1, 2, 3, and 3 time units respectively. If we are investigating an Ego action that is length 3, then there are 10 different action sequences possible for the Agent within that time, shown in Figure 5-2, where the

black circles are sequences in which the final actions are completed and white circles represent incomplete sequences. The incomplete sequences need to be propagated to next prediction step, as we believe the Agent is partway through a maneuver.

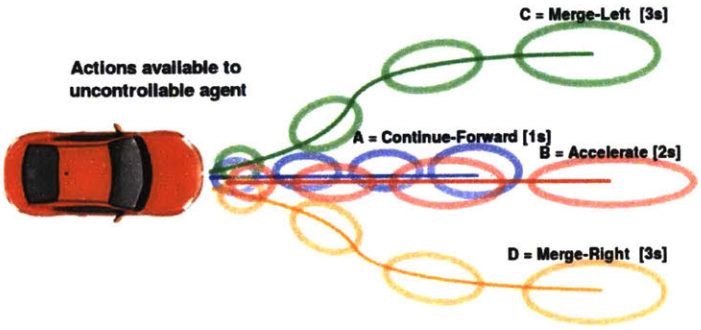


Figure 5-1: Example agent model with four representative maneuver actions with various temporal durations.

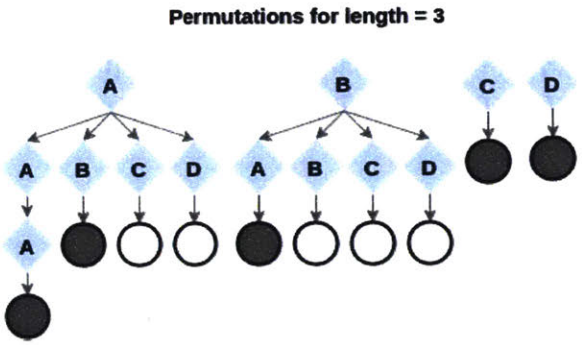


Figure 5-2: Action sequence permutations for the example agent model in Figure 5-1. Sequences with black circles end in completed actions while white circles are incompleted at a length of 3 time units.

The generation of these action sequences is outlined in Algorithm 5.1 for one Agent Vehicle. We start by finishing any incomplete maneuver (saved in the belief state) before recursively building a sequence for each possible action at the belief. For each of these permutations, we record the sequence of actions and intermediate belief states. Permutations of this approach can be used for multiple Agent Vehicles, where the example above would produce 100 possible action sequences for two Agent Vehicles, instead of 10 for the one Agent.

Algorithm 5.1 generate-sequences

Input: QAD Model M, Agent Model ag, Agent belief b, duration d.
Output: Action sequences p.

```
1: function generate-sequences(M, ag, b, d)
2:     p, a-seq ← ∅, t ← 0, b-seq ← [b]
3:     if b has incomplete maneuver then
4:         start a-seq with [b.maneuver]
5:         new-seq ← extend-sequence(M, b-seq, a-seq, t, d)
6:     for actions a in ag.available-actions(b) do
7:         new-a-seq ← add a to copy of action sequence a-seq
8:         new-seq ← extend-sequence(M, b-seq, a-seq, t, d)
9:         add all sequences in new-seq to permutations p
10:    return p

1: function extend-sequence(M, b-seq, a-seq, t, d)
2:     if newest action duration+t > d then
3:         return incomplete action sequence
4:     else if newest action duration+t = d then
5:         return complete action sequence
6:     else
7:         increment t with newest duration & add new belief from effects to b-seq
8:         for actions a in ag.available-actions(b) do
9:             new-a-seq ← add a to copy of action sequence a-seq
10:            new-seq, t ← extend-sequence(M, b-seq, a-seq, t, d)
```

5.3 Related Work

The brute force enumeration in the previous section reflects the complexity of conditioning on all possible sequences. Other approaches to planning with durative actions including concurrent MDPs (CoMDPs) and multi-agent POMDPs (MPOMDPs) [18]. More specifically, the CoMDPs are solved with a variant of Real-Time Dynamic Programming called sampled RTDP, which chooses which action combinations to perform bellman backups based on a distribution that favors more optimal combinations. However, the sampling-based and approximate methods lose the systematic guarantees on risk that we want from iRAO*. We propose two different approaches for addressing the complexity added by planning with durative actions. Instead of sampling, we take advantage of early pruning and clustering.

5.4 Mid-Permutation Pruning

Our first approach to addressing the complexity of multi-agent durative actions is by incorporating our risk pruning into the sequence generation. The idea here is to identify risky maneuver choices before all of the possible sequences are iterated. While generating the sequence permutations, sorted with most-likely first, we can keep a running estimate of our maneuver risk, instead of waiting until the end.

Given the recursive technique in Algorithm 5.1, we are unable to sort all of the maneuver sequences in order based on likelihood. Instead we change line 6 in *generate-sequences* and line 8 in *extend-sequence* to select the next most likely action given the belief and probabilities *pmf* from agent-model ag, so that we sort at each branching level. A non-recursive generation of the permutations would be able to sort all of the current sequences in order of likelihood but would require presenting longer pseudo code. Risk calculations for the intermediate belief states are performed at each recursion in line 7 of *extend-sequence* using the PFTs of the Agent maneuver and the Ego maneuver for timesteps t. These risk values are weighted by the likelihood of that action sequence and we can prune the Ego maneuver as soon as the total risk probability for the expanded beliefs exceeds the execution risk bound assigned to the initial belief in $iRAO^*$.

This approach allows us to focus on the most-likely sequences and prune the Ego maneuver being investigated before finishing the enumeration of all sequences. However, if the action is not too risky, then we still need to deal with the large number of belief states, one from each action sequence, that iRAO* will condition on.

5.5 State Clustering

Our second approach to addressing complexity is by clustering the resulting global states from the action sequences. For this, we use a variant of k-means clustering, where we guide the initial clusters to have relative positions surrounding our Ego Vehicle and various velocities, as shown in Algorithm 5.2. For positions, we focus

on clustering the positions into the 8 grid locations adjacent to the Ego Vehicle. For velocities, we experiment with 1, 2, or 3 seeded velocities for each location. In the case of 1 velocity, we initialize each as the Ego velocity, for 2 velocities, we initialize one faster and the other slower than the Ego, and for 3 velocities, a combination of the previous. This makes a total of 24 possible clusters for a given Agent Vehicle, but when we remove clusters that do not have assignments, it is often much fewer. This is because a single vehicle won't likely be predicted on all sides of the Ego vehicle at multiple speeds.

Algorithm 5.2 state-clustering

Input: Agent Vehicle belief states *beliefs*, iteration limit n
Output: Clustered belief states *clusters*.

1: **function** *state-clustering*(*beliefs*, n)
2: *clusters*.$\mu \leftarrow$, initialize with 8, 16, or 24 guided beliefs.
3: **for** n iterations **do**
4: $\forall b \in$ *beliefs* **Assign** to cluster nearest mean
5: $\forall c \in$ *clusters* **Update** new cluster mean given assigned beliefs
6: **if** assignments and *clusters*.μ unchanged **then break;**
7: $\forall c \in$ *clusters* sum weighted distributions of assigned b and normalize
8: **return** *clusters*

The *state-clustering* algorithm is a guided k-means clustering. We also included the option to limit the number of iterations instead of waiting for convergence. After the guided initialization and traditional **Assign** and **Update** iterations, the final weighted summation of the clustered beliefs is important for retaining all of the information from the enumerated action sequences. Returning to the qualitative example from before, now with 2 Agent Vehicles and up to 100 possible sequences, the ability to reduce that to 4 or 5 clustered states for each Agent makes a big difference in branching. Quantitative results from *generate-sequences* and the two algorithmic approaches to dealing with the complexity are included in the Results chapter.

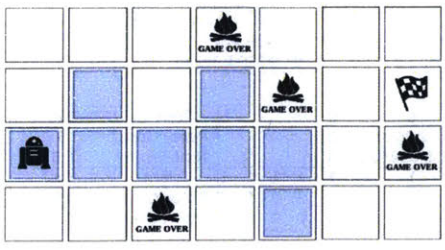
Chapter 6

Results

This chapter presents various results from the contributions outlined in this thesis, organized in the order that the material was introduced.

6.1 Improvements to Offline RAO*

This first section shows how the improvements to RAO*, introduced in Chapter 2, improve the performance of our offline conditional planner. The idea is that we can terminate the search early when the execution risk of the policy and the sum of non-terminal policy leaf likelihoods are less than the overall chance constraint. This means the remaining non-terminal leafs could all violate the constraints and the policy would still be acceptable. We use icy-robot CC-MDP from Definition 2 and the likelihood of sliding up or down when using action *moveRight* on ice is 10%.



5x9 Colony Bot Scenario		
Δ	New RAO*	Old RAO*
0.2	68	68
0.3	48	53
0.4	48	53
0.5	123	164
0.6	30	500
0.7	27	500

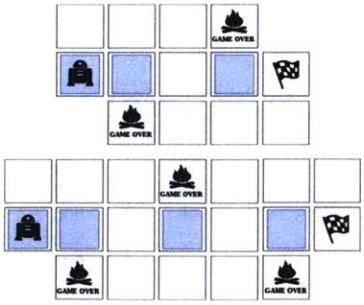
6x10 Colony Bot Scenario		
Δ	New RAO*	Old RAO*
0.2	121	500
0.3	93	500
0.4	63	500
0.5	96	500
0.6	57	500
0.7	40	500

Figure 6-1: Comparison of original RAO* with the new pruning of low likelihood branches. Example randomized scaled Colony-Bot domain on the left, final results shown from 5x9 and 6x10 grids.

Results comparing the old RAO* with the new low-likelihood pruning RAO* are shown in Figure 6-1. The Colony-Bot domain on the left is an example 4x7 randomized scenario, where the start is always in the left column and goal in the right-most. We selected two examples where the new RAO* was able to terminate search. The tables on the right show the number of planning iterations required for the selected 5x9 and 6x10 domains given various chance constraint values. We have an iteration limit of 500 for the search. The new RAO* was able to terminate search for the policy in much fewer iterations by ignoring the large number of low likelihood branches caused by the icy cells. An unintended consequence of this improvement is the ability to solve domains with infinite loops, where the loops were on branches with lower likelihoods, where the original RAO* fails to terminate.

6.2 iRAO*

For analyzing the performance improvements of iRAO*, we compare the incremental policy updates to using RAO* after each new observation. We use variations of the Colony-Bot domain, where the safest path is the longest to the goal, with a winding S-shape around ice and fire blocks. This allows us to see a direct trade-off between risk and reward, as the shortest path contains all the risk. The likelihood of sliding up or down when using action *moveRight* on ice is 10% (11% and 89% in the first cell). The domains used to produce these results are shown on the left in Figure 6-2. The chance constraint for entering the *gameOver* fire state is 0.12.



RAO*	# Iterations	23	22	5	4	0				Total #	54
	Policy Value	6.33	5.12	2.34	1.34	0				Policy Cost	4
	Execution Risk	0.088	0.099	0.099	0.099	0				Risk Taken	0.2
iRAO*	# Iterations	23	1	2	1	1	1	0		Total #	29
	Policy Value	6.33	5.12	4	3	2	1	0		Policy Cost	6
	Execution Risk	0.088	0.099	0	0	0	0	0		Risk Taken	0.1

RAO*	# Iterations	43	43	23	21	5	3	0					Total #	138
	Policy Value	9.72	8.88	6.12	5.11	2.34	1.34	0					Policy Cost	6
	Execution Risk	0.108	0.112	0.099	0.11	0.099	0.099	0					Risk Taken	0.3
iRAO*	# Iterations	44	1	1	1	1	1	1	1	1	1	1	Total #	54
	Policy Value	9.72	8.87	8	7	6	5	4	3	2	1	0	Policy Cost	10
	Execution Risk	0.108	0.112	0	0	0	0	0	0	0	0	0	Risk Taken	0.1

Figure 6-2: Comparison of iRAO* planning iterations, policy value, and execution risks against RAO* being used after each new observation along the most likely path through variations of the Colony-Bot domain.

The tables on the right in Figure 6-2 show how subsequent iRAO* iterations efficiently reuse the previous search results, resulting in faster online replanning, and ensure the overall policy from start to end does not violate the initial chance constraint. We seek to minimize the cost in this example, so the policy values are a cost-to-go metric of the remaining conditional plan at each step. The use of RAO* after each new observation requires more computation and exceeds the overall risk in both scenarios. This is a result of RAO* ignoring previously taken risk and repeatedly taking the 10% risk from the shorter icy paths without violating its definition of policy risk. The results also show the trade-off between risk and reward, as iRAO* results in longer but safer executions.

6.3 Driving Scenarios

For the qualitative autonomous driving, we focus on modeling highway scenarios that require the planner to look ahead a few steps and predict how the traffic would unfold. In doing so, we seek to highlight the model-based reasoning more than just lane-keeping and obstacle avoidance. An example highway scenario is shown in Figure 6-3 where the Ego Vehicle in white is considering the three different maneuvers shown with colored PFTs.

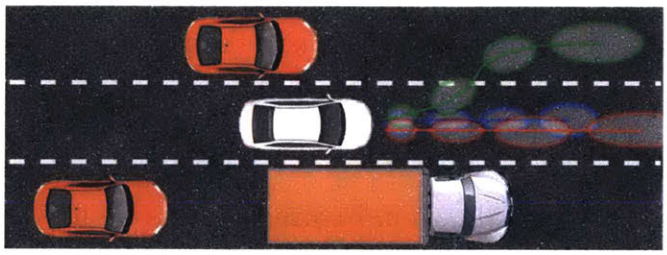


Figure 6-3: Example highway traffic scenario planning for the white Ego car with two nearby orange cars and one truck.

We present results for the highway on-ramp scenario shown in Figure 6-4, where the primary risk of collision for our Ego Vehicle comes from the orange car entering the highway and merging into our lane. There is also a truck in the passing lane that prevents the Ego Vehicle from immediately changing lanes in response to the

orange car. The truck agent is modeled with two forward manuever actions, either continuing forward (50%) or reducing speed (50%). The orange car agent is modeled with 3 forward maneuvers, modeling the on-ramp merge as a single lane, with continue forward (70%), accelerate (20%), or reducing speed (5%). There are various preconditions that affect the branching factor from each belief, including the fact that the truck won't slow down twice and the orange car won't accelerate twice, modeling their desire to maintain safe highway speeds in both cases.

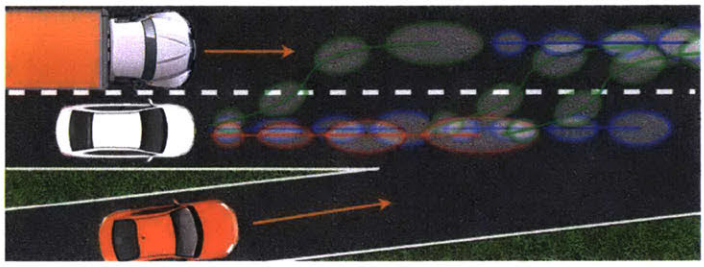


Figure 6-4: Highway on-ramp driving scenario with example PFT combinations. Planning for white Ego car with truck in the passing lane and orange car entering our current lane.

6.3.1 Synchronized Maneuvers

For the original hybrid multi-agent models with synchronized *action-models*, we condition on the possible actions of nearby vehicles at each step. The on-ramp scenario from Figure 6-4 starts in the initial state where the truck is traveling at the same veloicty as the Ego Vehicle and the and orange car is 10% slower. The available maneuvers for each Agent Vehicle are detailed above in Section 6.2.1 and the resulting conditional plan from iRAO* is shown in Figure 6-5 when we use a chance constraint of 0.01. The maneuvers available to our Ego Vehicle are shown in the top left of the Figure and the hyperedge colors in the conditional plan correspond to each action. We visualize two of the most-likely branches below the conditional plan.

The first most-likely branch of the policy is conditioned on the observations that the truck and orange car both continue forward at a constant velocity, as predicted by the prior distributions. The resulting maneuver sequence for the Ego Vehicle in this case is to slow-down twice before safely merging left behind the truck. The second

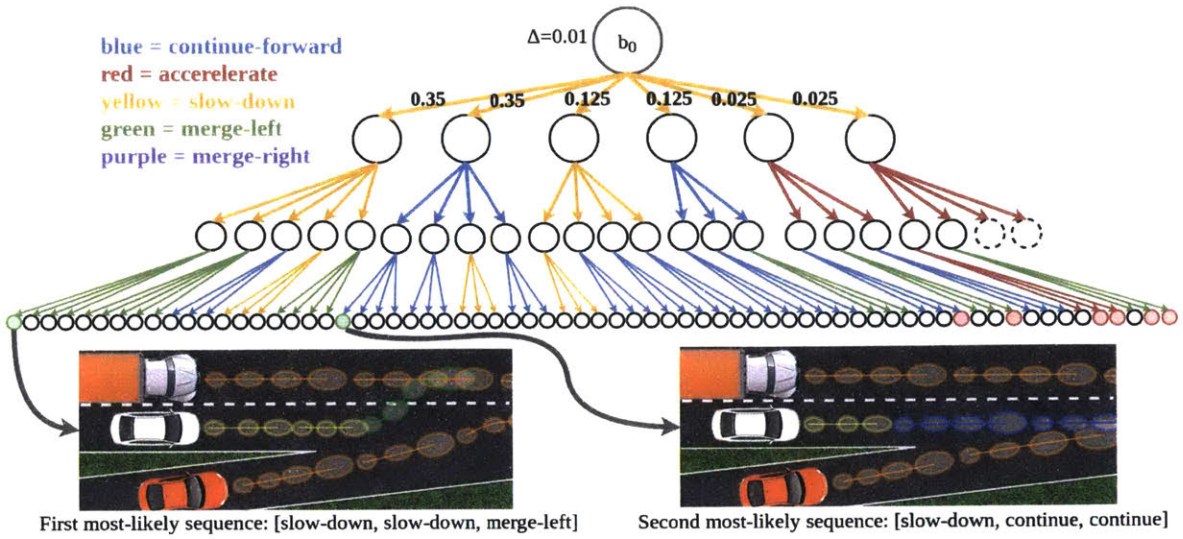


Figure 6-5: Computation time for n agent problems using RAO* with and without low-likelihood pruning.

highlighted branch of the policy is conditioned on the truck continuing forward and the orange car accelerating in its second action. The resulting maneuver sequence in this case has the Ego Vehicle slow down before continuing behind the faster moving orange car in the right lane. The conditional plans also includes risky beliefs, shown as red circles in the bottom right, where the Ego Vehicle accelerates to attempt to pass the orange car if it slows down as the first action. The policy beliefs are organized left to right by likelihood and two of the non-terminal leafs at the second depth are pruned from the search as their likelihood summed to less than $\Delta - er(b_0|\pi)$ before they were selected for expansion.

6.3.2 Durative Maneuvers

Adding durations to the *action-models*, the agents now have asynchronous decision epochs instead of branching on all actions at the same time. As discussed in Chapter 5, these durative actions add more complexity to the search space. Figure 6-6 shows results from *generate-sequences* outlined in Algorithm 5.1. The Agent Vehicle actions come from the example in Figure 5-1 and we compute the sequences for Ego Vehicle maneuver durations of 3 and 4 time units. The squares in the diagram represent decision epochs for the Agent Vehicle.

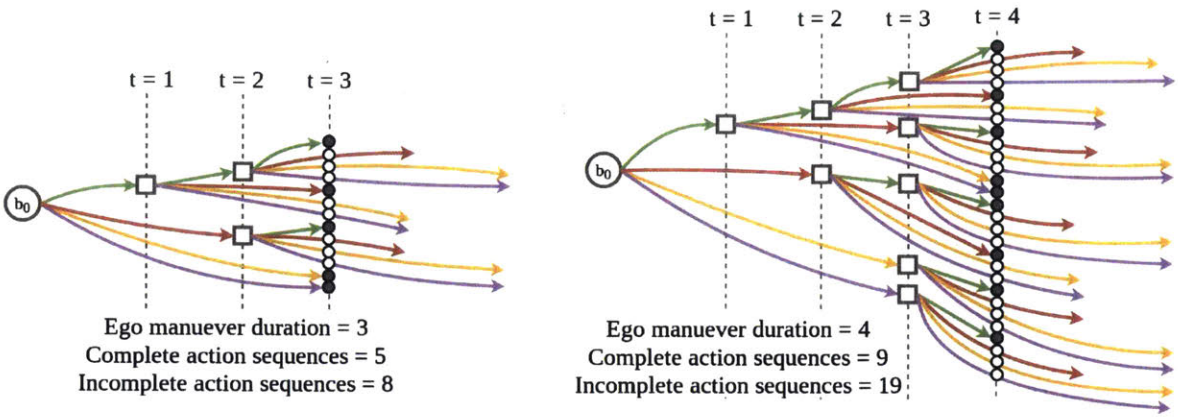


Figure 6-6: Action sequence permutations, for Ego maneuver durations of 3 and 4 times units, from the example maneuvers in Figure 5-1, where beliefs with gray circles are complete actions and white are incomplete.

In Figure 6-7, mid-permutation pruning is able to show that the Ego maneuver of merging left (with a duration of 3 time units) to go around the truck is too risky. This is a result of our model for the orange car in the passing lane having a high likelihood of accelerating during the next step, which can be due to the belief state indicating a more aggressive driver. The alternative is shown is Figure 6-6 for generating all the possible sequences for the orange Agent car during our single merging maneuver.

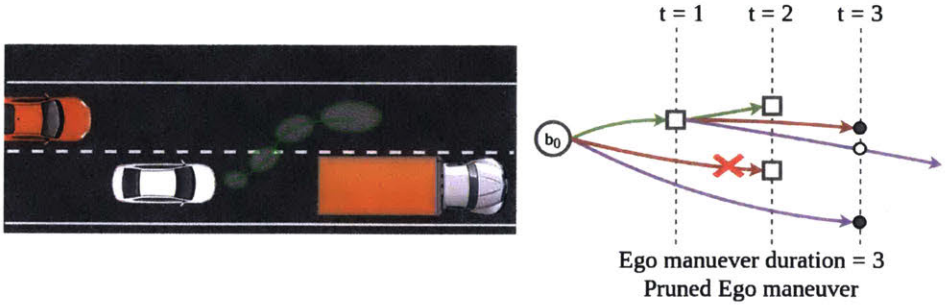


Figure 6-7: Mid-permutation pruning for the Ego maneuver to merge left.

Chapter 7

Conclusions

As a summary, this thesis defined the problem of generating online risk-bounded conditional plans for CC-POMDPs, with motivation from applications in qualitative autonomous driving. We proposed incremental RAO*, a risk-aware forward heuristic search algorithm that uses observations during execution to make incremental policy repairs and planning updates to ensure chance constraints are met online. We extended the CC-POMDP to a hybrid model for autonomous vehicles with durative actions and included approaches for reducing the exponential branching factor from multi-agent modeling. We presented results that show the benefits of iRAO* for online applications as well as representative results from autonomous vehicle scenarios. This final chapter adds insights to the results and recommends future work.

7.1 Recommended Future Work

7.1.1 Receding Horizon Backup Plan

We introduced a receding horizon variation of iRAO* in Section 3.5, but this approach can only make valid safety guarantees to the horizon at each step. As seen in [29], there is a need for a backup plan in risk-bounded receding horizon planning. The heuristic forward search has a limited look-ahead and can result in dead-ends on subsequent receding horizons with no safe options remaining. Having a backup plan,

which gets our autonomous agent to a guaranteed safe state within the horizon, might be necessary to combat dead-ends that are encountered in subsequent horizons. For the autonomous driving applications proposed in this thesis, we assume the horizon is long enough to plan and perform maneuvers into a safe state, like slowing to a stop behind traffic for example. However, having a separate conditional plan where the goal is reaching a safe state within the horizon, analogous the secondary MILP encoding in [29], will ensure that we are always prepared for dead-ends. The potential for innovation here is by reusing information from the main explicit graph, instead of doing two entirely separate conditional planning problems at each step. This is an important problem to address for any application where iRAO* is planning maneuvers of dynamic systems with a receding horizon, including autonomous vehicles.

7.1.2 Dynamic iRAO*

Another major innovation would be designing a dynamic version of iRAO*, where efficient replanning can respond to changes in the underlying models. This is motivated by the D* Lite algorithm [16], where changes to edge costs can indicate dynamic obstacles and the algorithm repairs the plan by propagating outwards from the change instead of replanning from scratch. We think there is an analogous propagation approach to the forward likelihood/risk-bound propagation done by iRAO* when the posterior belief is updated. The biggest challenge here would be guaranteeing optimality over the horizon with the changed risk model without replanning from nothing. For example, many actions are pruned by the execution risk bounds that may need to be reconsidered once the risk of the resulting states changes. Do we re-investigate all of the actions again or can we label pruned actions with useful information to help improve replanning efficiency?

Related to a dynamic approach, another improvement to address the issue of iRAO* complexity would be pruning unnecessary actions or states in the search based on value guarantees. As the policy grows, the *update-policy* procedure accounts for more computation as all children of the nodes are re-investigated. We think that there are insights from SMA*, which remains memory-bounded by pruning low value

options, that can be applied to iRAO* to manage the growing complexity [25]. Any online version of RAO*, which is a systematic conditional planner in order to make safety guarantees, needs to deal with the reality of space and time complexity.

7.1.3 Uncertain Duration Actions

While this thesis covered an extension to planning with durative actions, we only got as far as fixed durations. Actions that have uncertain durations are more powerful for modeling realistic scenarios. Instead of a fixed duration, we would represent the temporal length of each action with a distribution, preferably a Guassian with mean and variance. The branching described for multi-agent models would need to instead be able to calculate the most likely sequence of actions given duration distributions and then conditioning on the different possible orderings of events.

For the case of planning with our autonomous vehicles models, this also means further development of the PFT framework to allow for probabilistic durations, as currently the uncertainties are limited to spatial states through known time.

7.1.4 Macro Actions

The final recommendation for future research is related to using iRAO* in a qualitative autonomous driving system. The problem comes from the modeling decision of how to break up driving maneuvers into discrete actions, because there is a trade-off with modeling shorter or longer duration actions. For example, a maneuver to continue forward for 1 time unit vs 3 time units. The more primitive action at 1 time unit allows us to plan a larger variety of complex maneuver sequences within the same time. However, without looking ahead as far, there could be less risk pruning done at each small action step, forcing the planner to spend more time investigating contingencies that will end up being too risky after 3 time units regardless of subsequent actions, something that would have been seen instantly with an action of duration 3. The actions with longer durations can also be seen simply as chains of shorter actions. We can default to the shorter durations, treating them like primitive

actions, and allow the autonomous system itself to learn if chained primitive actions are useful to planning, by reasoning over previous successful executions. This includes chains of different primitive actions, for example a merge-left and 2 continue-forward actions chained together. A meta-action representing a sequence of action steps is often called a Macro Action and there are many approximate techniques for planning with Macro Actions in Decentralized-POMDPs [4].

This problem can be broken up into two steps. There is first the task of identifying promising sequences of maneuvers from past execution. There are interesting approaches for this that can use machine learning techniques on a database of previous policies and belief states to find patterns in action executions. The second task is then reasoning over the promising sequences to define the Macro Actions themselves. This means breaking down the previous execution of maneuvers and extracting the useful features of the composite states. For example, passing a large truck on the highway can be done by changing lanes and accelerating to avoid the blind spots. The system might find that this sequence is used often but would need to identify correlated features of the composite state including the truck being in front of us and the passing lane being open at the time. The ultimate goal would be to have a system that can create and apply these Learned Situational Macro Actions in order to progressively become a better driver.

Bibliography

[1] National Highway Traffic Safety Administration. *National Motor Vehicle Crash Causation Survey: Report to Congress.*

[2] National Highway Traffic Safety Administration. *Quick Facts 2016.*

[3] National Highway Traffic Safety Administration and Eun-Ha Choi. *Crash factors in intersection-related crashes: an on-scene perspective.* U.S. Dept. of Transportation, National Highway Traffic Safety Administration, 2010.

[4] Christopher Amato, George D. Konidaris, and Leslie P. Kaelbling. Planning with macro-actions in decentralized pomdps. In *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '14, pages 1273–1280, Richland, SC, 2014. International Foundation for Autonomous Agents and Multiagent Systems.

[5] Michele Bertoncello and Dominik Wee. Ten ways autonomous driving could redefine the automotive world, Jun 2015.

[6] G.E.P. Box. Robustness in the strategy of scientific model building. In ROBERT L. LAUNER and GRAHAM N. WILKINSON, editors, *Robustness in Statistics*, pages 201 – 236. Academic Press, 1979.

[7] Shuonan Dong and B. Williams. Motion learning in variable environments using probabilistic flow tubes. In *2011 IEEE International Conference on Robotics and Automation*, pages 1976–1981, May 2011.

[8] Shuonan Dong and Brian Williams. Learning and recognition of hybrid manipulation motions in variable environments using probabilistic flow tubes. *International Journal of Social Robotics*, 4(4):357–368, Nov 2012.

[9] Carlos Guestrin, Daphne Koller, and Ronald Parr. Multiagent planning with factored mdps. In *Advances in neural information processing systems*, pages 1523–1530, 2002.

[10] Nathaniel Hendren and Raj Chetty. *The Impacts of Neighborhoods on Intergenerational Mobility Childhood Exposure Effects and County - Level Estimates.* Apr 2015.

[11] Andreas Hofmann and Brian Williams. Exploiting spatial and temporal flexibility for plan execution of hybrid, under-actuated systems. In *AAAI 2006*, 2006.

[12] Joshua D Isom, Sean P Meyn, and Richard D Braatz. Piecewise linear dynamic programming for constrained pomdps. In *AAAI*, pages 291–296, 2008.

[13] Dongho Kim, Jaesong Lee, Kee-Eung Kim, and Pascal Poupart. Point-based value iteration for constrained pomdps. In *IJCAI*, pages 1968–1974, 2011.

[14] Will Knight. Networked cars will see accidents before they happen, Jul 2015.

[15] Will Knight. The Dark Secret at the Heart of AI. *MIT Technology Review*, Apr 2017.

[16] Sven Koenig and Maxim Likhachev. D*lite. In *Eighteenth National Conference on Artificial Intelligence*, pages 476–483, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence.

[17] Omid Madani, Steve Hanks, and Anne Condon. On the undecidability of probabilistic planning and infinite-horizon partially observable markov decision problems. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference Innovative Applications of Artificial Intelligence*, AAAI '99/IAAI '99, pages 541–548, Menlo Park, CA, USA, 1999. American Association for Artificial Intelligence.

[18] Mausam and Daniel S. Weld. Planning with durative actions in stochastic domains. *J. Artif. Int. Res.*, 31(1):33–82, January 2008.

[19] Nils J Nilsson. *Principles of artificial intelligence.* Morgan Kaufmann, 2014.

[20] Massachusetts Registry of Motor Vehicles. *Massachusetts Driver's Manual - RMV.*

[21] Christos Papadimitriou and John N. Tsitsiklis. The complexity of markov decision processes. *Math. Oper. Res.*, 12(3):441–450, August 1987.

[22] Pascal Poupart, Aarti Malhotra, Pei Pei, Kee-Eung Kim, Bongseok Goh, and Michael Bowling. Approximate linear programming for constrained partially observable markov decision processes. In *AAAI*, pages 3342–3348, 2015.

[23] B. Ranft and C. Stiller. The role of machine vision for intelligent vehicles. *IEEE Transactions on Intelligent Vehicles*, 1(1):8–19, March 2016.

[24] Stéphane Ross, Joelle Pineau, Sébastien Paquet, and Brahim Chaib-draa. Online planning algorithms for pomdps. *CoRR*, abs/1401.3436, 2014.

[25] Stuart Russell. Efficient memory-bounded search methods. In *Proceedings of the 10th European Conference on Artificial Intelligence*, ECAI '92, pages 1–5, New York, NY, USA, 1992. John Wiley & Sons, Inc.

[26] Stuart J. Russell and Peter Norvig. *Artificial intelligence: A Modern Approach.* Pearson Education, Inc., 2010.

[27] Pedro Santana. *Dynamic execution of temporal plans with sensing actions and bounded risk.* PhD thesis, 2016.

[28] Santana, Pedro, Sylvie Thiebaux, and Brian Williams. RAO*: an Algorithm for Chance-Constrained POMDPs. In *Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)*, Feb 2016.

[29] T. Schouwenaars, J. How, and E. Feron. Receding horizon path planning with implicit safety guarantees. In *Proceedings of the 2004 American Control Conference*, volume 6, pages 5576–5581 vol.6, June 2004.

[30] David Silver and Joel Veness. Monte-carlo planning in large pomdps. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 2164–2172. Curran Associates, Inc., 2010.

[31] Mark M Tobenkin, Ian R Manchester, and Russ Tedrake. Invariant funnels around trajectories using sum-of-squares programming. *IFAC Proceedings Volumes*, 44(1):9218–9223, 2011.

[32] Aditya Undurti and Jonathan P How. An online algorithm for constrained pomdps. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 3966–3973. IEEE, 2010.