METAREASONING FOR PLANNING AND EXECUTION IN AUTONOMOUS SYSTEMS

A Dissertation Presented

by

JUSTIN SVEGLIATO

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

February 2022

College of Information and Computer Sciences

METAREASONING FOR PLANNING AND EXECUTION IN AUTONOMOUS SYSTEMS

A Dissertation Presented

by

JUSTIN SVEGLIATO

Approved as to style and content by:

ACKNOWLEDGMENTS

It has always been clear to me that my success in graduate school has been the product of many people who have supported and believed in me over the years. I would like to take this brief opportunity to give my appreciation and gratitude to everyone who contributed to my experience in graduate school.

Shlomo Zilberstein, my advisor, has been an incredible mentor. He has always encouraged me to explore my own passions and taught me how to conduct research in an enjoyable and productive way. Under his guidance, I have developed many skills that have been and will be central to my academic career, whether it be posing research questions, mentoring students, constructing papers, or writing grants. Thank you, Shlomo, for being an endless source of wisdom, support, and opportunity.

I would like to thank my committee for being instrumental to my thesis. Joydeep Biswas offered insightful technical expertise and always encouraged me to apply my research to robots that operate in the world. Rod Grupen provided useful pointers on situating my work in the broad context of autonomous systems. Meghan Huber added a friendly, unique perspective that augmented my traditional view of robotics.

I am extremely grateful to Eileen Hamel, Leeanne Leclerc, and Michele Roberts for their constant help and unreasonable patience. The intricate maze of graduate school—whether it be degree requirements, course registration, or conference reimbursements—would have been impossible to navigate without the three of you.

Alan Labouseur and Carolyn Matheus have been instrumental to my growth as a researcher and teacher. By introducing me to research, they sparked my curiosity in going to graduate school and becoming a professor. More importantly, they have always given me constant, honest advice—both personal and professional—that has shaped me in so many ways. Thank you, Alan and Carolyn, for everything.

I will always remember the fond memories that I shared with everyone in RBR. They have listened patiently to my preliminary ideas or rough practice talks and offered candid suggestions. Moreover, our countless discussions have made me a better researcher, communicator, collaborator, and mentor. I am thankful to my past and present labmates Connor Basich, Abhinav Bhatia, Moumita Choudhury, Rick Freedman, Saaduddin Mahmud, Shuwa Miura, Pete Peterson, Luis Pineda, Sandhya Saisubramanian, and Kyle Wray as well as several excellent undergraduate and master's students Allyson Beach, Ishan Khatri, Shane Parr, and Prakhar Sharma who I had the privilege of working with on many different projects.

Throughout graduate school, I was fortunate to live at *Gray Street.* My past and present housemates Matteo Brucato, Ameya Godbole, Samer Nashed, Zeal Shah, Shanu Vashishtha, and Kyle Wray have become great friends who always offered a much needed distraction from my work when I needed it the most. Thank you, Gray Street, for watching shows like Star Trek, playing video games and board games like Rocket League and Werewolf, and indulging in huge dinners at Formosa.

Similarly, I have been surrounded by a supportive, close-knit community at CICS. I have shared many memories with Akanksha Atrey, Sam Baxter, Garrett Bernstein, Su Lin Blodgett, Lucas Chaufournier, Eddie Cunningham, Khoshrav Doctor, Cecilia Ferrando, Ben Glickenhaus, Conrad Holtsclaw, Myungha Jang, Aishwarya Kamath, Katie Keith, Tiffany Liu, Manish Motwani, Chris Nota, Jonas Pfeiffer, Sadegh Rabiee, Soha Rostaminia, Dirk Ruiken, Katherine Thai, Kevin Winner, and Sam Witty. Thank you for making my years here in Amherst so memorable.

My close friends Andrew DiNonno, Joshua Matheus, and Ryan Montano have always provided me with constant advice ranging from my career trajectory and research direction to my financial plans and personal relationships. They each have a way of keeping me honest with myself and my goals and have become a part of my family over the years. Thank you for all of our memories together.

My family has surrounded me with enduring love throughout my life. I want to thank my parents Karen and John, my siblings Jessica, John, and Kailani, my aunt Heather, my brother-in-law Matt, and my father-in-law Rich. Finally, my partner Yume has stood by me patiently and encouraged me during every challenging step of graduate school. Thank you, Yume, for your lasting love, empathy, and support. I look forward to our lives together.

ABSTRACT

METAREASONING FOR PLANNING AND EXECUTION IN AUTONOMOUS SYSTEMS

FEBRUARY 2022

JUSTIN SVEGLIATO

B.Sc., MARIST COLLEGE

M.Sc., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Shlomo Zilberstein

Metareasoning is the process by which an autonomous system optimizes, specifically monitors and controls, its own planning and execution processes in order to operate more effectively in its environment. As autonomous systems rapidly grow in sophistication and autonomy, the need for metareasoning has become critical for *efficient* and *reliable* operation in noisy, stochastic, unstructured domains for long periods of time. This is due to the uncertainty over the limitations of their reasoning capabilities and the range of their potential circumstances. However, despite considerable progress in metareasoning as a whole over the last thirty years, work on metareasoning for *planning* relies on several assumptions that diminish its accuracy and practical utility in autonomous systems that operate in the real world while work on metareasoning for *execution* has not seen much attention yet. This dissertation therefore proposes more effective metareasoning for planning while expanding the scope of metareasoning to execution to improve the efficiency of planning and the reliability of execution in autonomous systems.

In the first part of this dissertation, we propose two forms of metareasoning for *efficient planning* in autonomous systems. The first approach determines when to interrupt an anytime algorithm and act on the current solution by using online performance prediction: the meta-level control technique estimates optimal stopping of the anytime algorithm by predicting the performance of the anytime algorithm online. The second approach tunes the hyperparameters of the anytime algorithm at runtime by using deep reinforcement learning: the meta-level control technique estimates optimal hyperparameter tuning of the anytime algorithm by learning through simulation. The final result is a metareasoning framework that can determine the stopping point as well as tune the hyperparameters of anytime algorithms to achieve efficient planning in autonomous systems.

In the second part of this dissertation, we propose two forms of metareasoning for *reliable execution* in autonomous systems. The first approach recovers from exceptions that can be encountered during operation by using belief space planning: the meta-level control technique interleaves a main decision process with a set of exception handlers to detect, identify, and handle exceptions. The second approach maintains and restores safety during operation by using probabilistic planning: the meta-level control technique executes in parallel a main decision process and a set of safety processes with a conflict resolver for arbitration. The final result is a metareasoning framework that can recover from exceptions as well as maintain and restore safety to attain reliable execution in autonomous systems.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure **Page**

CHAPTER 1
INTRODUCTION

1.1 Metareasoning for Bounded Rationality

It has long been recognized that intelligent agents cannot be capable of perfect rationality due to the intractability of optimal decision making in complex domains [132, 133, 67, 123, 178]. In fact, in the early twentieth century, Herbert Simon noted the two main limitations of optimal decision making that is necessary to perfect rationality. First, perfect rationality can be impossible for autonomous systems because optimal decision making may require performing an intractable number of computations within a limited amount of time. Second, even if it were possible, perfect rationality can reduce the utility offered to autonomous systems since optimal decision making may involve using substantial computational resources, such as computation time, excessive processor usage, or memory pressure. As an example of these limitations, the optimal policy to a decision making problem that represents a navigation task can either be impossible to compute or take so long to compute that it no longer offers any utility to a mobile robot, such as an autonomous vehicle, a space exploration rover, or an unmanned aerial vehicle. Hence, motivated by the lack of operational significance of optimal decision making in complex domains, Simon concluded that intelligent agents ought to be capable of *bounded rationality* by using some criteria that evaluates whether or not a decision is *satisficing*—meaning that the decision is "satisfactory" or "good enough" in Scottish—for the situation at hand.

Simon's analysis of bounded rationality in intelligent agents has resulted in a substantial body of work within psychology and artificial intelligence. In psychology, there have been efforts to develop descriptive models of human rationality [45]. These models attempt to describe how people make decisions in the real world in the face of difficult problems with considerable uncertainty, complicated features, and strict time constraints. For example, instead of heavily relying on logical reasoning and exhaustive knowledge, there have been studies that suggest that people rely on simple—albeit roughly accurate—general-purpose heuristic methods for decision making in complex situations [46]. Moreover, in artificial intelligence, there have been efforts to develop computational approaches to bounded rationality [123]. These approaches attempt to build methods that incorporate the cost of decision making among other factors into the process of deliberation of autonomous systems [52, 66, 37, 163, 122, 175]. For instance, there have been methods that determine when to interrupt an anytime algorithm and act on the current solution by balancing the quality of a solution with the computation time required to compute that solution [61, 60, 149] and techniques that execute a portfolio of anytime algorithm in parallel by allocating different shares of processing power [109]. However, while Simon was the first to offer a comprehensive analysis of bounded rationality, he did not propose an effective computational framework for implementing bounded rationality in intelligent agents.

Specifically, in artificial intelligence, there have been three main computational approaches to bounded rationality in intelligent agents that have gradually been developed over the last fifty years. The earliest and simplest computational approach to bounded rationality has been based on *approximate reasoning*. Approximate reasoning can take on many different forms. A typical form of approximate reasoning employs algorithms that can compute an *approximate* solution to an instance of a problem. More often than not, computing an approximate solution requires significantly less computational resources than computing an optimal solution to a problem. For example, in contrast to *exact* heuristic search algorithms that are designed to compute an optimal solution to a search problem, such as A* or LAO* [62, 59], *approximate* heuristic search algorithms, such as anytime weighted A* or anytime RBFS [61, 58], can be used to calculate an approximate solution. Both exact and approximate heuristic search algorithms, however, still exploit domain knowledge to guide the search process, which improves their efficiency by reducing the amount of computation that must be performed by these algorithms. Another form of approximate reasoning employs algorithms that can compute an optimal solution to an instance of an *approximate* problem. In other words, it solves a simplified version of the original problem. By eliminating details that may be necessary to optimality in order to reduce complexity, the simplified version of the original problem can be significantly easier to solve optimally. For instance, in the field of probabilistic planning, it is possible to solve reduced models that remove either less informative state features or less likely outcomes, which may be necessary for perfect rationality but not critical for bounded rationality [112, 113, 126, 129, 128, 127]. However, regardless of its form, approximate reasoning is often complemented by other more complex computational approaches to bounded rationality.

The most ambitious computational approach to bounded rationality has been based on *bounded optimality*. Bounded optimality techniques find the most effective program that can compute a solution to an instance of a problem within the space of programs that are defined by the specific computational architecture of an autonomous system [121, 122]. More formally, a program can be said to be optimal if the overall expected utility of running the program on the specific computational architecture of an autonomous system is at least as high as all other programs that could be run. For example, on a mobile robot, if the computational architecture requires node expansions in a heuristic search algorithm to plan a route in an office building, the space of programs could be represented by different strategies for ordering the set of nodes to be expanded by the heuristic search algorithm [121]. Moreover, if the mobile robot used an anytime algorithm for route planning instead, the space of programs could be represented by different methods for deciding when to interrupt the anytime algorithm and act on the current solution [60]. It is therefore the responsibility of the designer—and surprisingly not the autonomous system itself—to not only develop the program but also prove that the program is optimal for the computational architecture of an autonomous system. Since bounded optimality techniques can be challenging to develop for many problems even given a finite space of programs, a weaker form has been proposed for *asymptotic bounded optimality* that only requires that the program performs as effectively as the optimal program by a constant factor on every instance of the problem. However, while bounded optimality offers a formal framework and exhibits many useful properties for bounded rationality, it has rarely been used in practice due to the lack of practicality and feasibility of optimizing over *programs* that solve problems rather than over *solutions* to problems.

The computational approach to bounded rationality that has seen the most use in practice has been based on *metareasoning* [178]. In general, a metareasoning technique is a process that enables an autonomous system to monitor and control its own object-level processes in order to act more effectively in its environment. When a metareasoning technique can be proven to be optimal with respect to maximizing the overall expected utility of an autonomous system, we typically refer to it as *optimal metareasoning*. As an example, one of the most popular forms of optimal metareasoning enables an autonomous system to monitor and control an anytime algorithm as its own planning process. In particular, there have been different metareasoning techniques for determining when to interrupt an anytime algorithm and act on the current solution by using a model that represents the performance of the anytime algorithm [61, 60]. At a high level, this general framework allows autonomous systems to handle any uncertainty about the range of their potential circumstances and the limitations of their reasoning capabilities. Metareasoning is therefore the focus of this thesis because it has been shown to be the most effective computational approach to bounded rationality in autonomous systems over the years.

Naturally, there has been a wide range of work on metareasoning for autonomous systems. Generally, each metareasoning technique optimizes—namely monitors and controls—a specific object-level process that is executed by the autonomous system. Here, we mention just a few typical examples of metareasoning that monitor and control different forms of planning. First, [90] offers a technique for *algorithm selection* that identifies the best algorithm to solve a problem among a set of candidate algorithms by compiling a model with a limited number of features to predict the efficiency and accuracy of each algorithm. Similarly, [109] introduces a technique for *algorithm portfolios* that executes a collection of anytime algorithms in parallel by allocating different shares of processing power to each algorithm. Moreover, [179] presents a technique for *contract sequencing* that executes an anytime algorithm for a sequence of contracts with or without stochastic information about the deadline. More recently, [63] proposes a technique for *simulation selection* that chooses the next computation, specifically the next simulation, to be performed by Monte Carlo tree search techniques by representing the decision as a Bayesian selection problem that maximizes the value of information. Finally, [5] provides a technique for *state space expansion* that interleaves unrolling the state space of a Markov decision process with solving for a partial policy by using a set of heuristic conditions. Central to these metareasoning techniques is the idea that a decision or a set of decisions must be made by the autonomous system to optimize a specific form of planning.

A large body of work on metareasoning for autonomous systems that is related to planning but is broader in scope has also been well studied. First, [41] introduces a technique for *learning management* that determines when to restart or stop learning based on its perceived skill level on the problem at hand by managing a collection of heuristics. Next, [118] proposes a technique for *filter selection* that identifies a filter from a bank of filters given the perceived effectiveness of each filter in tracking a selected object. In addition, [101] and [48] provide a technique for *domain knowledge adaptation* that adjusts the domain knowledge of an autonomous system if a decision has been deemed incorrect by using meta-knowledge that describes the current structure of that domain knowledge. Moreover, [11] offers a technique for *case-based reasoning adjustment* that monitors a case-based reasoner, determines the cause of any failures, and selects actions that adjust the case-based reasoner accordingly. More broadly, [9] and [8] develop a framework referred to as *the meta-cognitive loop* that monitors, reasons about, and adjusts the decision-making module of an autonomous system to improve its robustness to perturbations of the world. Note that it is worthwhile to mention that a collection of work on distributed metareasoning [114, 120] and metareasoning in cognitive architectures [34, 84] has seen much attention.

Generally, as autonomous systems rapidly grow in sophistication and autonomy, the need for metareasoning as an approach to bounded rationality has become critical to their design, development, and deployment. In this thesis, we argue that metareasoning is an effective framework for both *efficient* and *reliable* decision making in autonomous systems that operate in noisy, stochastic, unstructured domains for long periods of time: *efficient* in that the system has the ability to perform rapid planning and *reliable* in that the system has the ability to exhibit robust execution. This dissertation answers four main questions that are central to efficient and reliable decision making in autonomous systems that operate in the real world:

1. **Stopping.** How can an autonomous system determine when to interrupt an anytime algorithm and act on the current solution without the need for any substantial offline work?

2. **Hyperparameter Tuning.** How can an autonomous system tune the hyperparameters of an anytime algorithm at runtime to boost its overall performance on a specific problem instance and time constraint?

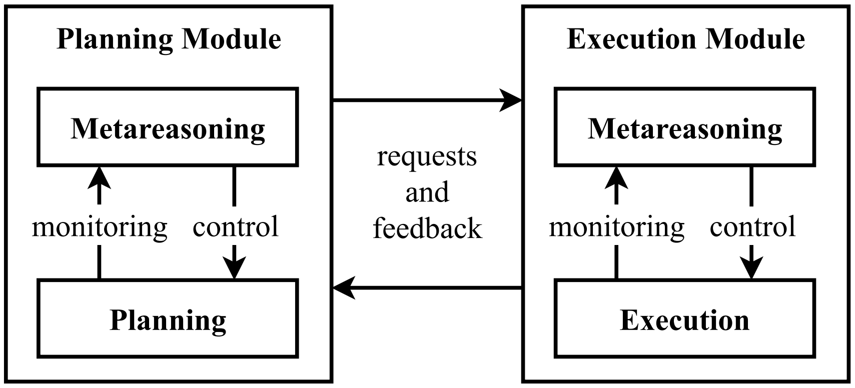


Figure 1.1: A metareasoning framework for monitoring and controlling the planning and execution processes of autonomous systems.

3. **Exception Recovery.** How can an autonomous system not only detect and identify but also handle exceptions during operation that prevent its main decision processes from completing a task?

4. **Safety.** How can an autonomous system maintain and restore safety during operation as its main decision processes are completing a task?

In order to answer these questions, we propose a novel metareasoning framework for autonomous systems with two modules shown in Figure 1.1. Suppose that an autonomous system has a *planning* process that *plans* a policy and an *execution* process that *executes* a policy. This metareasoning framework enables the autonomous system to monitor and control its planning processes for efficiency and its execution processes for reliability. At a high level, the main objective of this thesis is to develop more effective metareasoning for planning while expanding the scope of metareasoning to execution. We summarize the planning module and the execution module below.

The *planning* meta-level control module monitors and controls the *planning* process of the autonomous system in order to efficiently generate a policy to be followed by the execution module. Similar to earlier work [60], we use a planning process based on an anytime algorithm that gradually improves the quality of a policy as it runs and returns the current policy if it is interrupted. However, because meta-level control of anytime algorithms poses two well-known limitations, we propose two forms of metareasoning for the planning process. First, since existing meta-level control techniques rely on several unrealistic assumptions that can diminish the accuracy and practical utility of anytime algorithms in autonomous systems that operate in the real world, we offer metareasoning for *optimal stopping* that uses *online performance prediction* and *reinforcement learning*. Second, since existing meta-level control techniques do not tune the hyperparameters of anytime algorithms at runtime, which can slow the improvement in the quality of a policy in many domains, we offer metareasoning for *optimal hyperparameter tuning* that uses *deep reinforcement learning*.

The *execution* meta-level control module monitors and controls the *execution* process of the autonomous system in order to reliably follow a policy generated by the planning module. However, because there has been little work in applying meta-level control beyond the planning process to the execution process of an autonomous system, we propose two forms of metareasoning for the execution process. First, we offer metareasoning for *exception recovery* that interleaves a main decision process with a set of exception handlers to detect, identify, and handle exceptions during operation by using *belief space planning*. Second, we offer metareasoning for *safety* that executes in parallel a main decision process and a set of safety processes with a conflict resolver for arbitration to maintain and restore safety during operation by using *probabilistic planning*.

1.2 Thesis Contributions

We summarize the two forms of metareasoning for the planning module (i.e., Contributions 1 and 2) and the two forms of metareasoning for the execution module (i.e., Contributions 3 and 4) that compose the metareasoning framework that we propose for autonomous systems. Note that each main chapter of this thesis corresponds to a contribution. We summarize the contributions of this thesis below.

1. **Metareasoning for Stopping (Planning)** Anytime algorithms offer a trade-off between solution quality and computation time that has proven to be useful in autonomous systems for a wide range of real-time decision making problems. To optimize this trade-off, an autonomous system has to solve a challenging meta-level control problem: the autonomous system must decide when to interrupt the anytime algorithm and act on the current solution. Existing meta-level control techniques for anytime algorithms, however, rely on planning with a performance profile that must be compiled offline prior to the activation of meta-level control. This poses a number of unrealistic assumptions that reduce the accuracy and usefulness of meta-level control of anytime algorithms in the real world. Eliminating these assumptions, we therefore introduce two different approaches to meta-level control of anytime algorithms. First, we propose a novel *model-based* approach to meta-level control based on *online performance prediction* that adapts to each instance of a problem without any substantial offline preprocessing. Second, we propose a novel *model-free* approach to meta-level control based on *reinforcement learning* that adapts to each instance of a problem by learning through online simulation. Both approaches are evaluated on a set of experiments that show that they outperform existing meta-level control techniques that require substantial offline work on several common benchmark domains and a mobile robot domain. The result is nonmyopic meta-level control that improves the accuracy and usefulness of meta-level control of anytime algorithms in autonomous systems.

2. **Metareasoning for Hyperparameter Tuning (Planning)** Anytime algorithms often have hyperparameters that can be tuned at runtime to boost their overall performance in a given scenario—a specific problem instance and time constraint. However, while existing work on metareasoning has focused on

determining when to interrupt an anytime algorithm and act on the current solution, there has not been much work on tuning the hyperparameters of an anytime algorithm at runtime. We therefore offer a general, decision-theoretic metareasoning approach that optimizes both the stopping point and hyperparameters of anytime algorithms. First, we propose a generalization of an anytime algorithm called an *adjustable algorithm* that can be interrupted at any time for its current solution with hyperparameters that can be tuned at runtime. Next, we offer a meta-level control technique that monitors and controls an adjustable algorithm by using deep reinforcement learning. Finally, we show that an application of our approach boosts overall performance on a common benchmark domain that uses anytime weighted A* to solve a range of heuristic search problems and a mobile robot application that uses RRT* to solve motion planning problems compared to standard approaches that either set its hyperparameters to a static value or tune its hyperparameters heuristically. The result is nonmyopic meta-level control that uses deep reinforcement learning to boost the overall performance of adjustable algorithms in autonomous systems.

3. **Metareasoning for Exception Recovery (Execution)** Due to the complexity of the real world, autonomous systems use decision-making models that rely on simplifying assumptions to make them computationally tractable and feasible to design. However, since these limited decision-making models cannot fully capture the domain of operation, an autonomous system may encounter unanticipated scenarios that cannot be resolved effectively. Addressing this problem, we therefore introduce an exception recovery metareasoning system that uses belief space planning to detect, identify, and handle exceptions during operation by interleaving a main decision process with a set of exception handlers. We then apply an exception recovery metareasoning system to an autonomous

driving domain. Finally, we demonstrate that an exception recovery metareasoning vehicle is effective in simulation and on a fully operational prototype. The result is a belief space planning approach to robust exception recovery in autonomous systems.

4. **Metareasoning for Safety (Execution)** Maintaining and restoring safety is critical to autonomous systems that operate in the real world. While developers carefully design, develop, and test the models used by autonomous systems for decision making, it is infeasible to build monolithic decision-making models that maintain or restore safety in every possible scenario that can be encountered within the domain of operation. Due to this limitation, the need for autonomous systems to have the ability to maintain and restore safety is critical. We therefore introduce a *safety metareasoning system* that reduces the *severity* of the system's safety concerns and the *interference* to the system's task. In particular, the system executes in parallel a *task process* that completes a specified task and *safety processes* that each address a specified safety concern with a *conflict resolver* for arbitration. First, we offer a formal definition of a safety metareasoning system as well as a recommendation algorithm for a safety process and an arbitration algorithm for a conflict resolver with a theoretical analysis of the correctness and worst-case time-complexity for each algorithm. We then apply a safety metareasoning system to a planetary rover exploration domain. Finally, we demonstrate that a safety metareasoning planetary rover is effective in simulation. The result is a probabilistic planning approach to maintaining and restoring safety in autonomous systems.

1.3 Graduate Work

We list work completed during my graduate career—namely conference papers, patents, and workshop papers—that has been published in chronological order below. This work has been separated into thesis work that has been completed toward my thesis and additional work that has been completed outside of my thesis. Note that an asterisk denotes that the authors contributed equally.

1.3.1 Thesis Work

- J. Svegliato, K. Wray, and S. Zilberstein. Meta-level control of anytime algorithms with online performance prediction. *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence.* 2018.

- J. Svegliato and S. Zilberstein. Adaptive metareasoning for bounded rational agents. *Proceedings of the IJCAI Workshop on Architectures and Evaluation for Generality, Autonomy and Progress in AI.* 2018.

- J. Svegliato, K. Wray, S. Witwicki, J. Biswas, and S. Zilberstein. Belief space metareasoning for exception recovery. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems.* 2019.

- J. Svegliato, P. Sharma, and S. Zilberstein. A model-free approach to meta-level control of anytime algorithms. *Proceedings of the IEEE International Conference on Robotics and Automation.* 2020.

- J. Svegliato, S. Witwicki, K. Wray, and S. Zilberstein. Introspective autonomous vehicle operational management. *US Patent 10,649,453.* 2020.

- J. Svegliato. A metareasoning framework for planning and execution in autonomous systems. *Proceedings of the Doctoral Consortium at the Twenty-Forth European Conference on Artificial Intelligence.* 2020.

- A. Bhatia, J. Svegliato, and S. Zilberstein. On the benefits of randomly adjusting anytime weighted A*. *Proceedings of the Fourteenth Annual Symposium on Combinatorial Search*. 2021.

- A. Bhatia, J. Svegliato, and S. Zilberstein. Tuning the hyperparameters of anytime planning: A deep reinforcement learning approach. *Proceedings of the ICAPS Workshop on Heuristics and Search for Domain-Independent Planning*. 2021.

- J. Svegliato, C. Basich, S. Saisubramanian, and S. Zilberstein. Using metareasoning to maintain and restore safety for reliable autonomy. *Proceedings of the IJCAI Workshop on Robust and Reliable Autonomy in the Wild*. 2021.

1.3.2 Additional Work

- J. Svegliato, S. Witty, A. Houmansadr, and S. Zilberstein. Belief space planning for automated malware defense. *Proceedings of the IJCAI Workshop on AI for Internet of Things*. 2018.

- C. Basich, J. Svegliato, K. Wray, S. Witwicki, J. Biswas, and S. Zilberstein. Learning to optimize autonomy in competence-aware systems. *Proceedings of the Eighteenth International Conference on Autonomous Agents and Multiagent Systems*. 2020.

- J. Svegliato, S. Nashed, and S. Zilberstein. An integrated approach to moral autonomous systems. *Proceedings of the Twenty-Forth European Conference on Artificial Intelligence*. 2020.

- C. Basich, J. Svegliato, S. Zilberstein, K. Wray, and S. Witwicki. Improving competence for reliable autonomy. *Proceedings of the ECAI Workshop on Agents and Robots for Reliable Engineered Autonomy*. 2020.

- J. Svegliato, S. Nashed, and S. Zilberstein. Ethically compliant planning in moral autonomous systems. *Proceedings of the IJCAI Workshop on AI Safety*. 2020.

- S. Parr, I. Khatri, J. Svegliato, and S. Zilberstein. Agent-aware state estimation: Effective traffic light classification for autonomous vehicles. *Proceedings of the ICRA Workshop on Sensing, Estimating and Understanding the Dynamic World*. 2020.

- J. Svegliato, S. Nashed, and S. Zilberstein. Ethically compliant sequential decision making. *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence*. 2021.

- S. Nashed*, J. Svegliato*, M. Brucato, C. Basich, R. Grupen, and S. Zilberstein. Solving Markov decision processes with partial state abstractions. *Proceedings of the IEEE International Conference on Robotics and Automation*. 2021.

- S. Nashed, J. Svegliato, and S. Zilberstein. Ethically compliant planning within moral communities. *Proceedings of the Fourth AAAI/ACM Conference on Artificial Intelligence, Ethics, and Society*. 2021.

- C. Basich, J. Svegliato, A. Beach, S. Zilberstein, K. Wray, and S. Witwicki. Improving competence via iterative state space refinement. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2021.

- S. Parr, I. Khatri, J. Svegliato, and S. Zilberstein. Agent-aware state estimation for autonomous vehicles. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2021.

1.4 Thesis Organization

The rest of this thesis is organized as follows. Chapter 2 reviews the standard formal sequential decision-making models and the standard solution methods used to solve them. Chapter 3 offers a model-based metareasoning approach that uses online performance prediction and a model-free metareasoning approach that uses reinforcement learning to monitor and control anytime algorithms in autonomous systems, enabling the *planning* module to determine when to interrupt an anytime algorithm and act on the current solution. Chapter 4 offers a metareasoning approach that uses deep reinforcement learning to monitor and control adjustable algorithms in autonomous systems, allowing the *planning* module to tune the hyperparameters of an adjustable algorithm at runtime. Chapter 5 offers a metareasoning approach that uses belief space planning for exception recovery in autonomous systems, enabling the *execution* module to detect, identify, and handle exceptions during operation. Chapter 6 offers a metareasoning approach that uses probability planning for safety in autonomous systems, allowing the *execution* module to maintain and restore safety during operation. Chapter 7 concludes with a summary of the thesis and future work.

CHAPTER 2
BACKGROUND

2.1 Overview

In this chapter, we review the standard formal sequential decision-making models and the standard solution methods used to solve them. First, we review the standard definition of Markov decision processes (MDP) and their standard planning and reinforcement learning methods. This includes a discussion of infinite horizon problems, finite horizon problems, stochastic shortest path problems, and other relevant concepts as well as a discussion of planning methods, such as value iteration, policy iteration, and linear programming, and reinforcement learning methods, such as SARSA and Q-Learning. Second, we review the standard definition of partially observable Markov decision processes (POMDP) and a standard planning method. This includes a discussion of infinite horizon problems, finite horizon problems, and belief MDPs as well as a discussion of the planning method value iteration.

2.2 Markov Decision Processes

A **Markov decision process** is a formal sequential decision-making model for reasoning in fully observable, stochastic environments [20]. Intuitively, the process describes the world of the agent using four components: a set of states of the world, a set of actions of the agent, a transition function, and a reward function. At each time step, when the agent performs an action in a state, the agent receives a reward based on the reward function and transitions to a successor state based on the transition function. Most importantly, a Markov decision process satisfies a special property.

The **Markov property** holds that any transition to a successor state only depends on the current state of the world and the current action of the agent, which means that the history of states and actions before the current state and action do not matter. The goal of the agent is to maximize some notion of value over all states.

2.2.1 Formal Definition

A **Markov decision process (MDP)** is a formal sequential decision-making model for reasoning in fully observable, stochastic environments [20]. An MDP can be described as a tuple, $\langle S, A, T, R \rangle$, where

- S is a finite set of **states**,
- A is a finite set of **actions**,
- $T : S \times A \times S \to [0, 1]$ is a **transition function**, $T(s, a, s') = \Pr(s'|s, a)$, that represents the probability of reaching a state $s' \in S$ after performing an action $a \in A$ in a state $s \in S$, and
- $R : S \times A \to \mathbb{R}$ is a **reward function**, $R(s, a)$, that represents the expected immediate reward of performing an action $a \in A$ in a state $s \in S$.

Figure 2.1 illustrates the casual relationships of a single time step of an MDP. At each time step in a problem of either an **infinite horizon** $h = \infty$ or a **finite horizon** $h \in \mathbb{N}$, the agent is in a state $s \in S$. When the agent performs an action $a \in A$ in a state $s \in S$, there is a reward and a transition: the agent gains a reward of $R(s, a) \in \mathbb{R}$ and transitions to a successor state $s' \in S$ with a probability of $T(s, a, s') \in [0, 1]$. The agent repeats these steps indefinitely if the problem has an infinite horizon $h = \infty$ or until reaching a horizon $h \in \mathbb{N}$ if the problem has a finite horizon $h \in \mathbb{N}$.

A solution to an MDP is a **policy** that indicates the action that the agent should perform in each state at each time step. The policy is used in an **objective function** that describes the value of each state at each time step with respect to that policy.

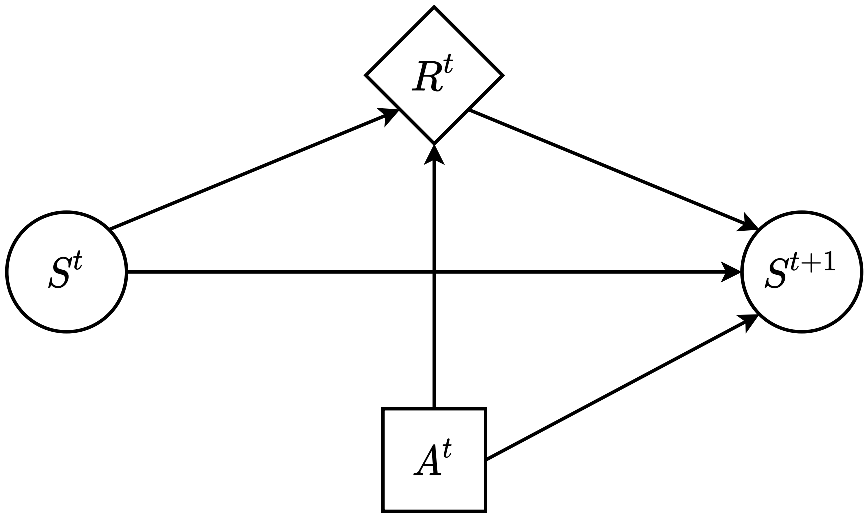


Figure 2.1: A Bayesian network for the casual relationships of an MDP.

The value of each state at each time step with respect to the policy is the expected cumulative reward that the agent would gain starting in that state and performing actions indicated by that policy until reaching the horizon. The expected cumulative reward is discounted in infinite horizon problems and undiscounted in finite horizon problems. In general, the goal of the agent is to find an **optimal policy** with an objective function that maximizes the value of each state at each time step over either an infinite horizon $h = \infty$ or a finite horizon $h \in \mathbb{N}$. We describe the objective functions for infinite horizon problems and finite horizon problems below.

2.2.1.1 Infinite Horizon Problems

In an MDP with an infinite horizon $h = \infty$ of a set of time steps $\mathcal{T} = \{1, 2, \ldots, \infty\}$ and a discount factor $\gamma \in [0, 1)$, a policy $\pi : S \to A$ indicates that the agent should perform an action $\pi(s) \in A$ in a state $s \in S$. The goal of the agent is to find an optimal policy $\pi^* : S \to A$ with an objective function that maximizes the value, or the expected discounted cumulative reward, over all states of the MDP:

$$\mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t R(s^t, \pi(s^t)) | \pi\Big], \tag{2.1}$$

where $s^t \in S$ is a random variable that represents the state at a time step $t \in \mathcal{T}$ generated by following the transition function $T : S \times A \times S \to [0, 1]$. Note that the time step does not need to be included in the representation of the policy or the value function in the case of infinite horizon problems.

For a policy $\pi : S \to A$, the state-value function $V^\pi : S \to \mathbb{R}$ is the value, or the expected discounted cumulative reward, for each state $s \in S$ of the MDP using the **Bellman equation**:

$$V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V^\pi(s'). \tag{2.2}$$

For a policy $\pi : S \to A$, the action-value function $Q^\pi : S \times A \to \mathbb{R}$ is the value, or the expected discounted cumulative reward, for each state $s \in S$ and each action $a \in A$ of the MDP:

$$Q^\pi(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^\pi(s'). \tag{2.3}$$

By using either the state-value function $V^\pi : S \to \mathbb{R}$ or the action-value function $Q^\pi : S \times A \to \mathbb{R}$ of a policy $\pi : S \to A$, it is possible to define the state-value function $V^* : S \to \mathbb{R}$ of an optimal policy $\pi^* : S \to A$ that maximizes the value, or the expected discounted cumulative reward, for each state $s \in S$ of the MDP using the **Bellman optimality equation**:

$$\begin{align} V^*(s) &= \max_{a \in A} \left[R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^*(s') \right] \tag{2.4} \\ &= \max_{a \in A} Q^*(s, a). \tag{2.5} \end{align}$$

Once the optimal action-value function $Q^* : S \times A \to \mathbb{R}$ has been calculated, the optimal policy $\pi^* : S \to A$ can be calculated by performing a one-step lookahead over each action $a \in A$ of the MDP in the following way:

$$\pi^*(s) = \arg\max_{a \in A} Q^*(s, a). \tag{2.6}$$

2.2.1.2 Finite Horizon Problems

In an MDP with a finite horizon $h \in \mathbb{N}$ of a set of time steps $\mathcal{T} = \{1, 2, \ldots, h\}$, a policy $\pi : S \times \mathcal{T} \to A$ indicates that the agent should perform an action $\pi(s, t) \in A$ in a state $s \in S$ at a time step $t \in \mathcal{T}$. The goal of the agent is to find an optimal policy $\pi^* : S \times \mathcal{T} \to A$ with an objective function that maximizes the value, or the expected undiscounted cumulative reward, over all states and all time steps of the MDP:

$$\mathbb{E}\Big[\sum_{t=0}^{h} R(s^t, \pi(s^t, t))|\pi\Big], \tag{2.7}$$

where $s^t \in S$ is a random variable that represents the state at a time step $t \in \mathcal{T}$ generated by following the transition function $T : S \times A \times S \to [0, 1]$.

For a policy $\pi : S \times \mathcal{T} \to A$, the state-value function $V^\pi : S \times \mathcal{T} \to \mathbb{R}$ is the value, or the expected undiscounted cumulative reward, for each state $s \in S$ at a time step $t \in \mathcal{T}$ of the MDP using the **Bellman equation**:

$$V^\pi(s^t, t) = R(s^t, \pi(s^t, t)) + \gamma \sum_{s^{t-1} \in S} T(s^t, \pi(s^t, t), s^{t-1}) V^\pi(s^{t-1}, t-1). \tag{2.8}$$

For a policy $\pi : S \times \mathcal{T} \to A$, the action-value function $Q^\pi : S \times A \times \mathcal{T} \to \mathbb{R}$ is the value, or the expected undiscounted cumulative reward, for each state $s^t \in S$, each action $a^t \in A$, and each time step $t \in \mathcal{T}$ of the MDP:

$$Q^\pi(s^t, a^t, t) = R(s^t, a^t) + \gamma \sum_{s^{t-1} \in S} T(s^t, a^t, s^{t-1}) V^\pi(s^{t-1}, t-1). \tag{2.9}$$

By using either the state-value function $V^\pi : S \times \mathcal{T} \to \mathbb{R}$ or the action-value function $Q^\pi : S \times A \times \mathcal{T} \to \mathbb{R}$ of a policy $\pi : S \times \mathcal{T} \to A$, it is possible to define an optimal policy $\pi^* : S \times \mathcal{T} \to A$ that maximizes value, or the expected undiscounted cumulative reward, for each state $s \in S$ and each time step $t \in \mathcal{T}$ of the MDP using the **Bellman optimality equation**:

$$V^*(s^t, t) = \max_{a \in A} \Big[R(s^t, a) + \gamma \sum_{s^{t-1} \in S} T(s^t, a, s^{t-1}) V^*(s^{t-1}, t-1) \Big] \tag{2.10}$$

$$= \max_{a \in A} Q^*(s^t, a, t). \tag{2.11}$$

Once the optimal action-value function $Q^* : S \times A \times \mathcal{T} \to \mathbb{R}$ has been calculated, the optimal policy $\pi^* : S \times \mathcal{T} \to A$ can be calculated by performing a one-step lookahead over each action $a \in A$ of the MDP in the following way:

$$\pi^*(s^t, t) = \arg\max_{a \in A} Q^*(s^t, a, t). \tag{2.12}$$

2.2.1.3 Stochastic Shortest Path Problems

Although MDPs are a standard formal sequential decision-making model that has been widely used in many applications, there is another formulation of MDPs that has been widely used as well. A **stochastic shortest path problem (SSP)** is an MDP with an initial state, a goal state, and a cost function instead of a reward function [82]. Intuitively, in an SSP, the agent must go from an initial state to a goal state in the least amount of cumulative cost. Since there is an initial state and a goal state, an SSP has an **indefinite horizon**, which means that the horizon is finite but unknown a priori. An SSP can be described as a tuple, $\langle S, A, T, C, s_0, s_g \rangle$, where

- S is a finite set of **states**,
- A is a finite set of **actions**,
- $T : S \times A \times S \to [0, 1]$ is a **transition function**, $T(s, a, s') = \Pr(s'|s, a)$, that represents the probability of reaching a state $s' \in S$ after performing an action $a \in A$ in a state $s \in S$,

- $C : S \times A \rightarrow \mathbb{R}^+$ is a **cost function**, $C(s, a)$, that represents the expected immediate cost of performing an action $a \in A$ in a state $s \in S$,
- $s_0 \in S$ is an **initial state**, and
- $s_g \in S$ is a **goal state** such that the transition probability $T(s_g, a, s_g)$ is 1 and the cost $C(s_g, a)$ is 0 for each action $a \in A$, which means that the goal state s_g is an absorbing state with nil cost.

In an SSP with an indefinite horizon $h \in \mathbb{N}$ of a set of time steps $\mathcal{T} = \{1, 2, \ldots, h\}$ where the horizon $h \in \mathbb{N}$ is finite but unknown a priori, a policy $\pi : S \rightarrow A$ indicates that the agent should perform an action $\pi(s) \in A$ in a state $s \in S$. The goal of the agent is to find an optimal policy $\pi^* : S \rightarrow A$ with an objective function that minimizes the value, or the expected undiscounted cumulative cost, over all states of the SSP:

$$\mathbb{E}\Big[\sum_{t=0}^{\infty} C(s^t, \pi(s^t))|\pi, s_0\Big], \tag{2.13}$$

where $s^t \in S$ is a random variable that represents the state at a time step $t \in \mathcal{T}$ generated by following the transition function $T : S \times A \times S \rightarrow [0, 1]$. Note that the time step does not need to be included in the representation of the policy or the value function in the case of indefinite horizon problems.

For a policy $\pi : S \rightarrow A$, the state-value function $V^\pi : S \rightarrow \mathbb{R}$ is the value, or the expected undiscounted cumulative cost, for each state $s \in S$ of the SSP using the **Bellman equation**:

$$V^\pi(s) = C(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s')V^\pi(s')]. \tag{2.14}$$

For a policy $\pi : S \rightarrow A$, the action-value function $Q^\pi : S \times A \rightarrow \mathbb{R}$ is the value, or the expected undiscounted cumulative cost, for each state $s \in S$ of the SSP:

$$Q^{\pi}(s,a) = C(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^{\pi}(s'). \tag{2.15}$$

By using either the state-value function $V^{\pi} : S \to \mathbb{R}$ or the action-value function $Q^{\pi} : S \times A \to \mathbb{R}$ of a policy $\pi : S \to A$, it is possible to define the state-value function $V^{*} : S \to \mathbb{R}$ of an optimal policy $\pi^{*} : S \to A$ that minimizes the value, or the expected undiscounted cumulative cost, for each state $s \in S$ of the SSP using the **Bellman optimality equation**:

$$\begin{aligned} V^{*}(s) &= \max_{a \in A} \Big[C(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^{*}(s') \Big] && (2.16) \\ &= \max_{a \in A} Q^{*}(s,a). && (2.17) \end{aligned}$$

Once the optimal action-value function $Q^{*} : S \times A \to \mathbb{R}$ has been calculated, the optimal policy $\pi^{*} : S \to A$ can be calculated by performing a one-step lookahead over each action $a \in A$ of the SSP in the following way:

$$\pi^{*}(s) = \arg\max_{a \in A} Q^{*}(s,a). \tag{2.18}$$

It is important to observe that a formal sequential decision-making model is an SSP if it satisfies two important conditions below.

1. There must exist a **proper policy**, which is a policy that has a probability of 1 of reaching the goal state $s_g \in S$ as the time step $t \in \mathcal{T}$ approaches ∞.

2. Every **improper policy**, which is a policy that has a probability of less than 1 of reaching the goal $s_g \in S$, must incur a state-value or action-value of ∞.

2.2.1.4 Relevant Concepts

There are two special types of states that are useful to discuss here. First, if the state in which the agent starts from is known in advance, like in an SSP, we call it an

initial state $s_0 \in S$. An initial state can be used to eliminate the need for planning over all states by only planning for states that are reachable from the initial state. This can be used to significantly reduce the computational requirements of computing an optimal policy. Second, if the state in which the agent ends at is known in advance, like in an SSP, we call it a **terminal state**. In some contexts, it is also called an **absorbing state**. A terminal state is a state in which any action performed in that state transitions back to that state: more formally, a state $s \in S$ is terminal if the transition probability $T(s, a, s)$ is 1 for each action $a \in A$.

It is possible to represent the state space S or the action space A in terms of a set of **state factors** or a set of **action factors**. For example, if we express a state space S in terms of a set of state factors $\{S_1, S_2, \ldots, S_N\}$, the state space S is equal to $S_1 \times S_2 \times \cdots \times S_N$. Similarly, if we express an action space A in terms of a set of action factors $\{A_1, A_2, \ldots, A_N\}$, the action space A is equal to $A_1 \times A_2 \times \cdots \times A_N$. Generally, while there may be specific conditions under which this does not hold given a state space or action space with certain properties, a factored representation of a state space or an action space is for convenience and not for computational tractability.

While the reward function $R(s, a)$ often uses a representation in terms of the current state $s \in S$ and the current action $a \in A$ of the agent, there are two other representations. First, when the reward gained by the agent is only a function of its current state $s \in S$, the reward function $R(s)$ can be used. Second, when the reward gained by the agent is a function of not only its current state $s \in S$ and current action $a \in A$ but also its successor state $s' \in S$, the reward function $R(s, a, s')$ can be used. An MDP that uses one representation of the reward function can easily be mapped to an MDP that uses another representation of the reward function.

2.2.2 Planning Methods

There are many planning methods that can be used to solve infinite horizon MDPs. In general, while solving infinite horizon MDPs is P-complete in the size of the problem in states and actions, it is common to have an exponential state space and action space defined by every permutation of the set of state factors and the set of action factors. We discuss several important exact planning methods for infinite horizon MDPs, such as value iteration, policy iteration, and linear programming, that have been used as a foundation for more sophisticated approximate planning methods.

2.2.2.1 Value Iteration

Value Iteration is an exact planning method for solving infinite horizon MDPs [20]. This planning method constructs a sequence of optimal finite horizon value functions $V_0^*, V_1^*, \ldots, V_t^*$. As each consecutive optimal finite horizon value function V_{t+1}^* is constructed from the current optimal finite horizon value function V_t^*, it approaches the optimal infinite horizon value function V^* as the time step $t \in \mathcal{T}$ approaches the infinite horizon $h = \infty$. That is, the difference between the optimal infinite horizon value function V^* and the optimal t-horizon value function V_t^* goes to zero as the time step $t \in \mathcal{T}$ approaches the infinite horizon $h = \infty$. For a given time step $t \in \mathcal{T}$, this difference can be calculated in the following way:

$$\lim_{t \to \infty} \max_{s \in S} |V^*(s) - V_t^*(s)| = 0,$$

where $V^*(s)$ is the optimal infinite horizon value function and $V_t^*(s)$ is the optimal t-horizon value function. For a given Bellman error ϵ, the optimal infinite horizon value function can be calculated in a finite number of optimal t-horizon value functions.

We describe the steps of value iteration for infinite horizon MDPs for a given Bellman error ϵ below.

1. **Initialize** a time step $t = 0$.

2. **Initialize** an optimal 0-horizon value function $V_0^*(s) = 0$ for each state $s \in S$.

3. **Repeat** the following steps until the termination condition $\max_{s \in S} |V_{t+1}^*(s) - V_t^*(s)| \leq \epsilon$ is satisfied.

   (a) **Calculate** the optimal $(t+1)$-horizon value function $V_{t+1}^*(s)$ for each state $s \in S$ using the Bellman optimality equation

   $$V_{t+1}^*(s) = \max_{a \in A} \Big[R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_t^*(s') \Big].$$

   (b) **Increment** the time step t.

4. **Calculate** the optimal policy $\pi^*(s)$ for each state $s \in S$ using the equation

$$\pi^*(s) = \arg\max_{a \in A} \Big[R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_t^*(s') \Big].$$

Note that value iteration results in an approximate infinite horizon value function that is within the following bound of the optimal infinite horizon value function:

$$\frac{2\epsilon\gamma}{1-\gamma}.$$

It is important to highlight that a variant of value iteration, called **asynchronous value iteration**, serves as the basis for more sophisticated approximate planning methods that solve infinite horizon MDPs and SSPs: the Bellman optimality equation can be applied to any state in any order as long as no state is starved indefinitely.

2.2.2.2 Policy Iteration

Policy iteration is an exact planning method for solving infinite horizon MDPs [20]. This planning methods begins with an initial 0-horizon policy π_0. It then alternates between a *policy evaluation step* that determines the value function V^{π_t} of the current t-horizon policy π_t and a *policy improvement step* that generates a new $(t+1)$-horizon policy π_{t+1} based on the value function V^{π_t} of the current t-horizon policy π_t.

We describe the steps of policy iteration for infinite horizon MDPs below.

1. **Initialize** an arbitrary 0-horizon policy $\pi_0(s) = a$ for each state $s \in S$ with any (random) action $a \in A$.

2. **Repeat** the following steps until the termination condition $\pi_t(s) = \pi_{t+1}(s)$ is satisfied.

    - **Evaluate** the current policy $\pi_t(s)$ by calculating the value function $V^{\pi_t}(s)$ of the t-horizon policy $\pi_t(s)$ for each state $s \in S$ using the Bellman equation

    $$V^{\pi_t}(s) = R(s, \pi_t(s)) + \gamma \sum_{s' \in S} T(s, \pi_t(s), s') V^{\pi_t}(s').$$

    - **Improve** the current policy $\pi_t(s)$ by calculating the new $(t+1)$-horizon policy π_{t+1} for each state $s \in S$ using the equation

    $$\pi_{t+1}(s) = \arg\max_{a \in A} \Big[R(s, \pi_t(s)) + \gamma \sum_{s' \in S} T(s, \pi_t(s), s') V^{\pi_t}(s') \Big].$$

It is important to note that policy iteration tends to converge in a smaller number of iterations than value iteration in practice but requires more computation in each iteration since the policy evaluation step requires solving a system of linear equations.

2.2.2.3 Linear Programming

A common approach to solving infinite horizon MDPs expresses the optimization problem as a linear program [97]. While the standard definition of an infinite horizon MDP is used here, it must be augmented with an initial state function $d : S \to [0, 1]$

that represents the probability $\Pr(s) \in [0,1]$ of starting in a state $s \in S$. In the primal form, a set of value variables V_s for the value $V(s)$ of each state $s \in S$ are minimized subject to a set of constraints that maintain consistent values using the Bellman equation. We describe the primal form of the linear program below.

$$\begin{aligned} \underset{V}{\text{minimize}} \quad & \sum_{s \in S} d(s) V_s \\ \text{subject to} \quad & V_s \geq R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_{s'} \quad \forall s \in S, a \in A \end{aligned}$$

In the dual form, a set of occupancy measures μ_a^s for the discounted number of times an action $a \in A$ is performed in a state $s \in S$ is maximized subject to a set of constraints that maintain consistent occupancy measures and nonnegative occupancy measures. We describe the dual form of the linear program below.

$$\begin{aligned} \underset{\mu}{\text{maximize}} \quad & \sum_{s \in S} \sum_{a \in A} R(s,a) \mu_a^s \\ \text{subject to} \quad & \sum_{a' \in A} \mu_{a'}^{s'} = d(s') + \gamma \sum_{s \in S} \sum_{a \in A} T(s,a,s') \mu_a^s \quad \forall s' \in S \\ & \mu_a^s \geq 0 \qquad \forall s \in S, a \in A \end{aligned}$$

It is easy to calculate an optimal policy from an optimal solution to the primal form or the dual form of the linear program by using the set of value variables V_s or the set of occupancy measures μ_a^s. Note that linear programming has served as the basis of recent solution methods [103].

2.2.3 Reinforcement Learning Methods

There are many reinforcement learning methods that can be used to solve infinite and finite horizon MDPs. To maximize a notion of value, a reinforcement learning agent learns a policy by taking actions in its environment and observing rewards from its environment. Similar to planning, the goal of reinforcement learning is to maximize expected discounted cumulative reward over time.

$$\mathbb{E}\big[\sum_{t=0}^{\infty}\gamma^t R(s^t,\pi(s^t))|\pi\big].$$

TD learning, a central approach to reinforcement learning, enables an agent to learn an optimal policy without a model of the transition function or the reward function. In particular, the agent can learn in TD learning in the following way. Suppose that an agent is operates in an environment. At each time step, the agent performs an action $a_t \in A$ in a state $s_t \in S$ at time step t, gains a reward signal r_t, transitions to a state s_{t+1}, and performs an action $a_{t+1} \in A$ in a state $s_{t+1} \in S$. By using this information, the agent can learn the optimal value function over time. We discuss two standard reinforcement learning methods for MDPs that are based on TD learning, specifically SARSA and Q-Learning, that have been used as the basis for more sophisticated approximate reinforcement learning methods.

2.2.3.1 SARSA

SARSA is the on-policy form of TD learning. In on-policy control, we learn the action-value function of the policy that is currently being followed by the agent. At each time step $t \in \mathcal{T}$, given a state $s \in S$, an action $a \in A$, a reward signal $r \in \mathbb{R}$, a successor state $s' \in S$, and an action $a' \in A$, the agent updates its action-value function using the SARSA update rule:

$$Q(s,a) \to Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)], \tag{2.19}$$

where α is a learning rate that changes the amount that the action-value function $Q(s,a)$ is adjusted in each update.

We describe the steps of the standard reinforcement learning method that uses the SARSA update rule below.

1. **Initialize** an arbitrary action-value function $Q(s, a)$ for each state $s \in S$ and each action $a \in A$.

2. **Repeat** the following steps for each episode $1, 2, \dots, N$.

    (a) **Set** the current state $s \in S$.

    (b) **Repeat** for each step of the episode until the state $s \in S$ is terminal or the horizon $h \in \mathbb{R}$ is reached.

        i. **Select** an action $a \in A$ for the current state $s \in S$ using a policy $\pi(s)$ calculated from the action-value function $Q(s, a)$ and an exploration strategy, such as softmax action selection or ϵ-greedy action selection.

        ii. **Perform** the action $a \in A$.

        iii. **Observe** a reward $r \in \mathbb{R}$.

        iv. **Transition** to a successor state $s' \in S$.

        v. **Update** the action-value function $Q(s, a)$ using the SARSA update rule as follows:

$$Q(s, a) \rightarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)].$$

        vi. **Set** the state $s \in S$ to the successor state $s' \in S$.

2.2.3.2 Q-Learning

Q-Learning is an **off-policy** form of TD learning. In off-policy control, we learn the action-value function of the policy that differs from the policy currently being followed by the agent. At each time step $t \in \mathcal{T}$, given a state $s \in S$, an action $a \in A$, a reward signal $r \in \mathbb{R}$, and a successor state $s' \in S$, the agent updates its action-value function using the Q-Learning update rule:

$$Q(s,a) \to Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a') - Q(s,a)], \tag{2.20}$$

where α is a learning rate that changes the amount that the action-value function $Q(s,a)$ is adjusted in each update.

We describe the steps of the standard reinforcement learning method that uses the Q-Learning update rule below.

1. **Initialize** an arbitrary action-value function $Q(s,a)$ for each state $s \in S$ and each action $a \in A$.

2. **Repeat** the following steps for each episode $1, 2, \ldots, N$.

   (a) **Set** the current state $s \in S$.

   (b) **Repeat** for each step of the episode until the state $s \in S$ is terminal or the horizon $h \in \mathbb{R}$ is reached.

      i. **Select** an action $a \in A$ for the current state $s \in S$ using a policy $\pi(s)$ calculated from the action-value function $Q(s,a)$ and an exploration strategy, such as softmax action selection or ϵ-greedy action selection.

      ii. **Perform** the action $a \in A$.

      iii. **Observe** a reward $r \in \mathbb{R}$.

      iv. **Transition** to a successor state $s' \in S$.

      v. **Update** the action-value function $Q(s,a)$ using the Q-Learning update rule as follows:

      $$Q(s,a) \to Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a') - Q(s,a)].$$

      vi. **Set** the state $s \in S$ to the successor state $s' \in S$.

2.3 Partially Observable Markov Decision Processes

A **partially observable Markov decision process** is a formal sequential decision-making model for reasoning in partially observable, stochastic environments [134, 76]. A partially observe Markov decision process is an MDP with two extra components: a set of observations made by the agent and an observation function. At each time step, when the agent performs an action in a state, the agent not only receives a reward based on the reward function and transitions to a successor state based on the transition function but also makes an observation based on the observation function. Similarly, the goal of the agent is to maximize some notion of value over all states.

2.3.1 Formal Definition

A **partially observable Markov decision process (POMDP)** can be described as a tuple, $\langle S, A, T, R, \Omega, O \rangle$, where

- S is a finite set of **states**,
- A is a finite set of **actions**,
- $T : S \times A \times S \to [0, 1]$ is a **transition function**, $T(s, a, s') = \Pr(s'|s, a)$, that represents the probability of reaching a state $s' \in S$ after performing an action $a \in A$ in a state $s \in S$,
- $R : S \times A \to \mathbb{R}$ is a **reward function**, $R(s, a)$, that represents the expected immediate reward of performing an action $a \in A$ in a state $s \in S$,
- Ω is a finite set of **observations**, and
- $O : S \times A \times \Omega \to [0, 1]$ is an **observation function**, $O(s, a, \omega) = \Pr(\omega|s', a)$, that represents the probability of making an observation $\omega \in \Omega$ after performing an action $a \in A$ and reaching a state $s' \in S$.

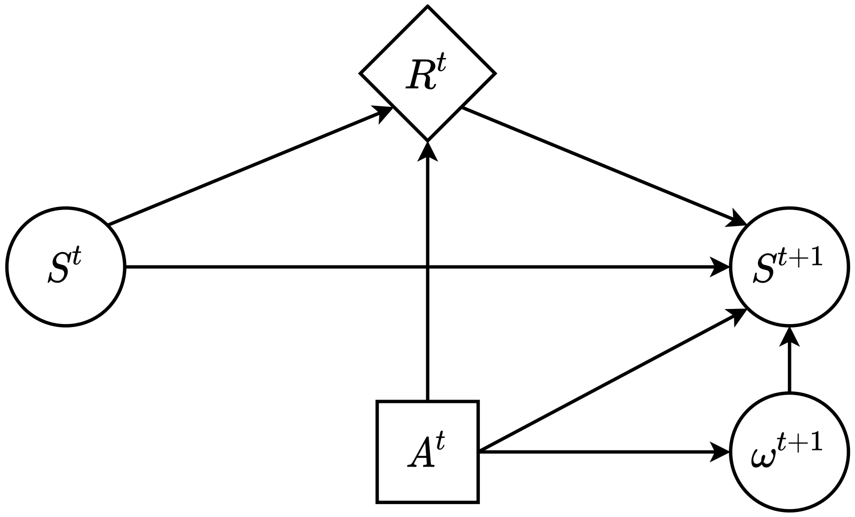


Figure 2.2: A Bayesian network for the casual relationships of a POMDP.

Figure 2.2 illustrates the casual relationships of a single time step of a POMDP. At each time step in a problem of either an **infinite horizon** $h = \infty$ or a **finite horizon** $h \in \mathbb{N}$, the agent is in a state $s \in S$. When the agent performs an action $a \in A$ in a state $s \in S$, there is a reward and a transition: the agent gains a reward of $R(s, a) \in \mathbb{R}$ and transitions to a successor state $s' \in S$ with a probability of $T(s, a, s') \in [0, 1]$. Once the agent has transitioned to a successor state $s' \in S$ after performing action $a \in A$, the agent makes an observation $\omega \in \Omega$ with a probability of $O(s, a, \omega) \in [0, 1]$. The agent repeats these steps indefinitely if the problem has an infinite horizon $h = \infty$ or until reaching a horizon $h \in \mathbb{N}$ if the problem has a finite horizon $h \in \mathbb{N}$.

It is important to mention that the agent does not necessarily know its current state $s \in S$. Instead, the agent makes noisy observations $\omega \in \Omega$ that reflect its current state $s \in S$ and current action $a \in A$. More formally, in order to represent its uncertainty, the agent maintains a belief state $b \in B$, a probability distribution $\Pr(s|b)$ over each state $s \in S$, where $B = \Delta^{|S|}$ is the space of all belief states. In general, since the belief state summarizes the entire history of states, actions, and observations that are encountered by the agent in a partially observable stochastic environment, the belief state satisfies the Markov property. Initially, the agent begins with an initial belief state $b_0 \in B$. After performing an action $a \in A$ and making an observation $\omega \in \Omega$, the agent updates its current belief state $b \in B$ to a new belief state $b' \in B$ using the **belief state update equation**:

$$b'(s'|b, a, \omega) = \frac{O(a, s', \omega) \sum_{s \in S} T(s, a, s') b(s)}{\Pr(\omega|b, s')}, \tag{2.21}$$

where $\Pr(\omega|b, s')$ is the probability that the agent observes an observation $\omega \in \Omega$ given that the agent has a belief state $b \in B$ in a state $s' \in S$, which serves as a normalization constant. Note that the notation $b'_{a,\omega}$ is used to represent the successor belief state generated by the belief state update equation for a belief state $b \in B$, an action $a \in A$, and an observation $\omega \in \Omega$.

A solution to a POMDP is a **policy** that indicates the action that the agent should perform in each belief state at each time step. That is, instead of using a policy based on its current state, the agent uses a policy based on its current belief state. Like an MDP, however, the policy is used in an **objective function** that describes the value of each state at each time step with respect to that policy. The value of each state at each time step with respect to the policy is the expected cumulative reward that the agent would gain starting in that belief state and performing actions indicated by that policy until reaching the horizon. The expected cumulative reward is discounted in infinite horizon problems and undiscounted in finite horizon problems. In general, the goal of the agent is to find an **optimal policy** with an objective function that maximizes the value of each belief state at each time step over either an infinite horizon $h = \infty$ or a finite horizon $h \in \mathbb{N}$. We describe the objective functions for infinite horizon problems and finite horizon problems below.

2.3.1.1 Infinite Horizons Problems

In a POMDP with an infinite horizon $h = \infty$ of a set of time steps $\mathcal{T} = \{1, 2, \ldots, \infty\}$ and a discount factor $\gamma \in [0, 1)$, a policy $\pi : B \rightarrow A$ indicates that the agent should perform an action $\pi(s) \in A$ in a belief state $b \in B$. Given an initial belief $b^0 \in B$, the goal of the agent is to find an optimal policy $\pi : B \rightarrow A$ with an objective function that maximizes the value, or the expected discounted cumulative reward, over all belief states of the POMDP:

$$\mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t R(b^t, \pi(b^t)) | \pi, b^0\Big], \tag{2.22}$$

where $b^t \in B$ is a random variable that represents the belief state at a time step $t \in \mathcal{T}$ generated by following the transition function $T : S \times A \times S \rightarrow [0, 1]$ and the observation function $O : S \times A \times \Omega \rightarrow [0, 1]$. Note that the time step does not need to be included in the representation of the policy or the value function in the case of infinite horizon problems.

For a policy $\pi : B \rightarrow A$, the state-value function $V^\pi : B \rightarrow \mathbb{R}$ is the value, or the expected discounted cumulative reward, for each belief state $b \in B$ of the POMDP using the **Bellman equation**:

$$V^\pi(b) = \sum_{s \in S} b(s) R(s, \pi(b)) + \gamma \sum_{\omega \in \Omega} \Pr(\omega | b, \pi(b)) V^\pi(b'_{\pi(b),\omega}). \tag{2.23}$$

For a policy $\pi : B \rightarrow A$, the action-value function $Q^\pi : B \times A \rightarrow \mathbb{R}$ is the value, or the expected discounted cumulative reward, for each belief state $b \in B$ and each action $a \in A$ of the POMDP:

$$Q^\pi(b, a) = \sum_{s \in S} b(s) R(s, \pi(b)) + \gamma \sum_{\omega \in \Omega} \Pr(\omega | b, a) V^\pi(b'_{a,\omega}). \tag{2.24}$$

By using either the state-value function $V^\pi : B \rightarrow \mathbb{R}$ or the action-value function $Q^\pi : B \times A \rightarrow \mathbb{R}$ of a policy $\pi : B \rightarrow A$, it is possible to define the state-value function $V^* : B \to \mathbb{R}$ of an optimal policy $\pi^* : B \to A$ that maximizes the value, or the expected discounted cumulative reward, for each belief state $b \in B$ of the POMDP using the **Bellman optimality equation**:

$$V^*(b) = \max_{a \in A} \Big[\sum_{s \in S} b(s) R(s, a) + \gamma \sum_{\omega \in \Omega} \Pr(\omega | b, \pi(b)) V^\pi(b'_{\pi(b)\omega}) \Big] \tag{2.25}$$

$$= \max_{a \in A} Q^*(b, a). \tag{2.26}$$

Once the optimal action-value function $Q^* : B \times A \to \mathbb{R}$ has been calculated, the optimal policy $\pi^* : B \to A$ can be calculated by performing a one-step lookahead over each action $a \in A$ of the MDP in the following way:

$$\pi^*(b) = \arg\max_{a \in A} Q^*(b, a). \tag{2.27}$$

It is important to highlight that, while infinite horizon POMDPs are an interesting framework for theoretical analysis, they are undecidable for most problems because there can be an infinite space of belief states with or without an initial belief state.

2.3.1.2 Finite Horizon Problems

In a POMDP with a finite horizon $h \in \mathbb{N}$ of a set of time steps $\mathcal{T} = \{1, 2, \ldots, h\}$ and a discount factor $\gamma \in [0, 1)$, a policy $\pi : B \to A$ indicates that the agent should perform an action $\pi(b) \in A$ in a belief state $b \in B$. Given an initial belief $b^0 \in B$, the goal of the agent is to find an optimal policy $\pi : B \to A$ with an objective function that maximizes the value, or the expected discounted cumulative reward, over all belief states and all time steps of the POMDP:

$$\mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t R(b^t, \pi(b^t)) | \pi, b^0 \Big], \tag{2.28}$$

where $b^t \in B$ is a random variable that represents the belief state at a time step $t \in \mathcal{T}$ generated by following the transition function $T : S \times A \times S \to [0, 1]$ and the observation function $O : S \times A \times \Omega \to [0, 1]$.

However, instead of using a policy $\pi : B \to A$, we use another representation in the interest of clarity for this discussion. At a time step $t \in \mathcal{T}$, a **policy tree** $\sigma_t \in \Gamma_t$ is a tuple, $\langle a, \upsilon_t \rangle$, where $a \in A$ is an action that should be performed by the agent and $\upsilon_t : \Omega \to \Gamma_{t-1}$ is an observation strategy that indicates a policy tree $\sigma_{t-1} \in \Gamma_{t-1}$ that should be followed after making an observation $\omega \in \Omega$. Given a set of observation strategies Γ_{t-1}, the set of policy trees Γ_t available to the agent with t time steps remaining in the problem is defined by the set below:

$$\Gamma_t = \{\langle a_t, \upsilon_t \rangle \mid a_t \in A, \upsilon_t \in \Gamma_{t-1}\}.$$

Note that a policy tree is only represented by an action $a \in A$ without an observation strategy when there are no time steps remaining in the problem since the agent does not need to make an observation $\omega \in \Omega$ to select a policy tree at the final time step.

By representing a policy $\pi : B \to A$ as a policy tree $\sigma_t = \langle a, \upsilon_t \rangle \in \Gamma_t$ with an action $a \in A$ and an observation strategy $\upsilon_t : \Omega \to \Gamma_{t-1}$ at each time step $t \in \mathcal{T}$, the state-value function for each state $s \in S$ can be represented like so:

$$V_t^\pi(s) = V_t^{\sigma_t}(s) = R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') \sum_{\omega \in \Omega} O(s, a, \omega) V^{\upsilon_t(o)}(s'). \tag{2.29}$$

Given the state-value function $V_t^\pi(s) = V_t^{\sigma_t}(s)$ and a policy tree $\sigma_t = \langle a, \upsilon_t \rangle \in \Gamma_t$ with an action $a \in A$ and an observation strategy $\upsilon_t : \Omega \to \Gamma_{t-1}$ at each time step $t \in \mathcal{T}$, it is possible to define a value function for each belief state $b \in B$ in the following way:

$$V_t^\pi(b) = V_t^{\sigma_t}(b) = \sum_{s \in S} b(s) V_t^{\sigma_t}(s). \tag{2.30}$$

Using a state-value function $V_t^{\pi}(s) = V_t^{\sigma_t}(s)$ for each state $s \in S$, the optimal value function for each belief state $b \in B$ can be calculated as follows:

$$V_t^*(b) = \max_{\sigma \in \Gamma_t} \sum_{s \in S} b(s) V_t^{\sigma_t(s)}(s). \tag{2.31}$$

The optimal value function $V_t^*(b)$ for each belief state $b \in B$ can be viewed as the upper surface of all of the state-value functions $V_t^{\sigma_t(s)}(s)$ that represent each policy tree $\sigma_t = \langle a, \upsilon_t \rangle \in \Gamma_t$ with an action $a \in A$ and an observation strategy $\upsilon_t : \Omega \to \Gamma_{t-1}$ at a time step $t \in \mathcal{T}$. The optimal value function $V_t^*(b)$ is therefore piecewise linear and convex. It is therefore possible to write the optimal value function $V_t^*(b)$ in another representation. Let us first define an α-vector α^{σ} as a vector that contains the optimal value of each state $s \in S$ with respect to the policy tree $\sigma \in \Gamma$:

$$\alpha^{\sigma} = [V^{\sigma}(s_0), V^{\sigma}(s_1), \dots, V^{\sigma}(s_n)].$$

By representing the value function $V_t^{\sigma}(b)$ as a set of α-vector α^{σ} for a policy tree $\sigma \in \Gamma$, we can rewrite the optimal value function $V_t^*(b)$ in the following way:

$$V_t^*(b) = \max_{\sigma \in \Gamma_t} \sum_{s \in S} b(s) \alpha^{\sigma}(s) = \max_{\alpha \in \mathcal{V}_t} \sum_{s \in S} b(s) \alpha(s), \tag{2.32}$$

where $\mathcal{V}_t$ is the set of all α-vectors such that each α-vector uniquely represents a policy tree $\sigma \in \Gamma_t$ at a time step $t \in \mathcal{T}$.

2.3.1.3 Belief Markov Decision Processes

An infinite horizon POMDP or a finite horizon POMDP can be cast as a continuous space MDP with a state space that is the set of belief states and an action space that is the set of actions from the original POMDP. This is typically called a **belief MDP**. A belief MDP can be described as a tuple, $\langle B, A, \tau, \rho \rangle$, where

1. $B = \Delta^{|S|}$ is the set of **belief states** from the original POMDP,

2. A is the set of **actions** from the original POMDP,

3. $\tau : B \times A \times B \to [0, 1]$ is the **belief transition function**, $\tau(b, a, b') = \Pr(b'|b, a)$, that represents the probability of reaching a belief state $b' \in B$ after performing an action $a \in A$ in a belief state $B \in B$, and

4. $\rho : B \times A \to \mathbb{R}$ is the **belief reward function**, $\rho(b, a)$, that represents the expected immediate reward of performing an action $a \in A$ in a belief state $b \in B$

It is possible to represent the belief transition function $\tau(b, a, b')$ and the belief reward function $\rho(b, a)$ in terms of the original formulation of the POMDP. The belief transition function $\tau(b, a, b')$ can be represented in the following way:

$$\tau(b, a, b') = \sum_{\omega \in \Omega} \Pr(b'|a, b, \omega) \sum_{s' \in S} O(s, a, \omega) \sum_{s \in S} T(s, a, s') b(s), \tag{2.33}$$

where $\Pr(b'|a, b, \omega)$ can be defined as $[b_{a,\omega} = b']$ with Iverson bracket notation $[\cdot]$. The belief reward function $\rho(b, a)$ can be represented in the following way:

$$\rho(b, a) = \sum_{s \in S} b(s) R(s, a). \tag{2.34}$$

In a belief MDP, the optimal state-value function $V^* : B \to \mathbb{R}$ for a belief $b \in B$ that can be used to calculate the optimal policy $\pi^* : B \to A$ is as follows:

$$V^*(b) = \max_{a \in A} \Big[\rho(b, a) + \gamma \sum_{\omega \in \Omega} \Pr(\omega|b, a) V^*(b'_{a,\omega}) \Big]. \tag{2.35}$$

In a belief MDP, the optimal action-value function $Q^* : B \times A \rightarrow \mathbb{R}$ for a belief $b \in B$ that can be used to calculate the optimal policy $\pi^* : B \rightarrow A$ is as follows:

$$Q^*(b, a) = \rho(b, a) + \gamma \sum_{\omega \in \Omega} \Pr(\omega | b, a) V^*(b'_{a,\omega}). \tag{2.36}$$

Finally, we can calculate the optimal policy $\pi^* : B \rightarrow A$ by performing a one-stop look-ahead over the optimal action-value function $Q^* : B \times A \rightarrow \mathbb{R}$:

$$\pi^*(b) = \arg\max_{a \in A} Q^*(b, a).$$

2.3.2 Planning Methods

There are many planning methods that can be used to solve infinite horizon POMDPs. We discuss an important exact planning method for infinite horizon POMDPs, value iteration, that has been used as a foundation for more sophisticated approximate planning methods [94, 32, 110, 135].

2.3.2.1 Value Iteration

Value Iteration is a standard exact planning method for solving infinite horizon POMDPs [20]. This planning method constructs a sequence of optimal finite horizon value functions $V_0^*, V_1^*, \ldots, V_t^*$. As each consecutive optimal finite horizon value function V_{t+1}^* is constructed from the current optimal finite horizon value function V_t^*, it approaches the optimal infinite horizon value function V^* as the time step $t \in \mathcal{T}$ approaches the infinite horizon $h = \infty$. That is, the difference between the optimal infinite horizon value function V^* and the optimal t-horizon value function V_t^* goes to zero as the time step $t \in \mathcal{T}$ approaches the infinite horizon $h = \infty$. For a given time step $t \in \mathcal{T}$, this difference can be calculated in the following way:

$$\lim_{t \rightarrow \infty} \max_{b \in B} |V^*(b) - V_t^*(b)| = 0,$$

where $V^*(b)$ is the optimal infinite horizon value function and $V_t^*(b)$ is the optimal t-horizon value function. For a given Bellman error ϵ, the optimal infinite horizon value function can be calculated in a finite number of optimal t-horizon value functions.

We describe the steps of value iteration for infinite horizon POMDPs in terms of a belief MDP $\langle B, A, \tau, \rho \rangle$ for a given Bellman error ϵ below.

1. **Initialize** a time step $t = 0$.

2. **Initialize** an optimal 0-horizon value function $V_0^*(b) = 0$ for each belief state $b \in B$.

3. **Repeat** the following steps until the termination condition $\max_{b \in B} |V_{t+1}^*(b) - V_t^*(b)| \leq \epsilon$ is satisfied.

   (a) **Calculate** the optimal $(t+1)$-horizon value function $V_{t+1}^*(b)$ for each belief state $b \in B$ using the Bellman optimality equation

   $$V_{t+1}^*(b) = \max_{a \in A} \Big[\rho(b, a) + \gamma \sum_{\omega \in \Omega} \Pr(\omega | b, a) V^*(b'_{a,\omega}) \Big].$$

   (b) **Increment** the time step t.

4. **Calculate** the optimal policy $\pi^*(b)$ for each belief state $b \in B$ using the equation

$$\pi(b) = \arg\max_{a \in A} \Big[\rho(b, a) + \gamma \sum_{\omega \in \Omega} \Pr(\omega | b, a) V^*(b'_{a,\omega}) \Big].$$

Note that value iteration results in an approximate infinite horizon value function that is within the following bound of the optimal infinite horizon value function:

$$\frac{2\epsilon\gamma}{1 - \gamma}.$$

CHAPTER 3
METAREASONING FOR STOPPING

3.1 Introduction

Starting with metareasoning for planning, we offer two metareasoning approaches that enable autonomous systems to determine the optimal stopping point of anytime algorithms. Anytime algorithms have been developed for a wide range of real-time decision-making problems, such as belief-space planning [110, 135, 86], probabilistic inference [115], heuristic search [61, 116, 105, 30, 35], motion planning [19, 157, 95, 77], constraint satisfaction [161], and object detection [181, 176, 78, 117]. Simply put, an *anytime algorithm* is an algorithm that gradually improves the quality of the current solution as it runs and returns the current solution if it is interrupted. This behavior offers a trade-off between solution quality and computation time that has proven to be useful in autonomous systems that need to produce effective action in a timely manner. However, in order to optimize this trade-off, an autonomous system has to solve a challenging meta-level control problem: the autonomous system must decide when to interrupt the anytime algorithm and act on the current solution.

Generally, there have been two main approaches to meta-level control of anytime algorithms in autonomous systems. The earliest approach that was proposed executes the anytime algorithm until a stopping point determined prior to runtime [66, 27]. Because the stopping point is not adjusted once the anytime algorithm starts, this approach is called *fixed allocation*. While fixed allocation is effective given negligible uncertainty in the performance of the anytime algorithm or the urgency for the solution, there is often substantial uncertainty in either or both these variables in real-time decision-making problems [108]. In response, a more sophisticated approach that was developed tracks the performance of the anytime algorithm and estimates a stopping point at runtime periodically [67, 182, 60, 92]. Since the stopping point is adjusted as the anytime algorithm is running, this approach is called *monitoring and control*. Overall, monitoring and control has proven to be a substantially more effective approach to meta-level control of anytime algorithms than fixed allocation.

Current meta-level control techniques that monitor and control anytime algorithms have traditionally relied on planning with a model, called a *performance profile*, that describes the performance of a given anytime algorithm solving a specific problem on a particular system [177, 60]. This model must be compiled offline through simulation before the activation of meta-level control by using the anytime algorithm to solve thousands of instances of the problem on the system. Planning with a performance profile, however, imposes many assumptions that are often violated by autonomous systems that operate in the real world [149, 150]:

- *There must be enough time for offline compilation of the performance profile of the anytime algorithm prior to the activation of meta-level control.*
- *The parameters of the anytime algorithm across every problem instance must not change over time to avoid invalidating the performance profile.*
- *The distribution of problem instances solved by the anytime algorithm must be known and fixed to compile and avoid invalidating the performance profile.*
- *The processor and memory conditions of the system that executes the anytime algorithm must be static to avoid invalidating the performance profile.*

In short, by using a performance profile that must be compiled offline and cannot be updated online, existing meta-level control techniques make a number of assumptions that reduce their accuracy and usefulness in the real world.

3.1.1 Contributions

In this chapter, we make the following contributions. First, we propose a novel *model-based* approach to meta-level control based on *online performance prediction* by offering these contributions:

1. a online performance prediction framework that can be used by a meta-level control technique to predict the performance of an anytime algorithm, and

2. a model-based meta-level control technique that determines when to interrupt an anytime algorithm and act on the current solution by using the online performance prediction framework.

Second, we propose a novel *model-free* approach to meta-level control based on *reinforcement learning* by offering these contributions:

3. a formal MDP representation of the meta-level control problem for anytime algorithms that can be used to learn the optimal stopping point of an anytime algorithm with reinforcement learning, and

4. a model-free meta-level control technique that learns when to interrupt an anytime algorithm and act on the current solution by using the formal MDP representation of the meta-level control problem for anytime algorithms.

Our work is evaluated on a set of experiments that show that they outperform existing meta-level control techniques that require substantial offline work on several common benchmark domains and a mobile robot domain.

3.2 Meta-Level Control Problem

We begin by reviewing the meta-level control problem for anytime algorithms. This problem requires a model that describes the utility of a solution computed by an anytime algorithm. Naturally, in real-time decision-making problems, a solution

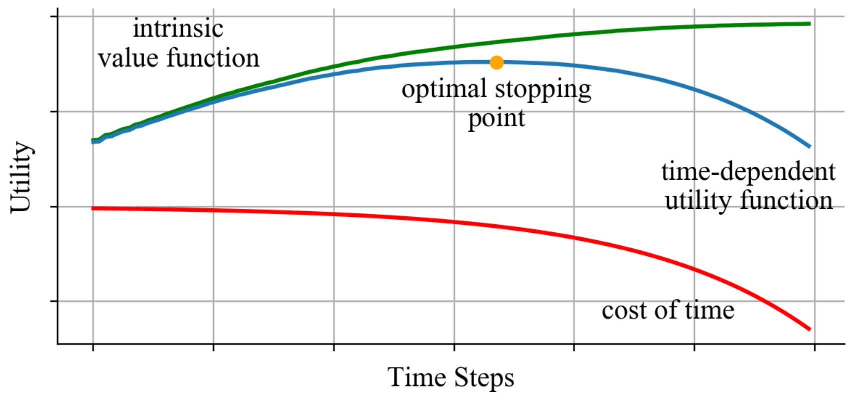


Figure 3.1: An example of the meta-level control problem.

of a specific quality calculated in a minute has lower utility than a solution of the same quality calculated in a second. At a minimum, this implies that the utility of a solution is likely a function of not only its quality but also its computation time [65, 27]. Following this line of intuition, we define the utility of a solution below.

Definition 1. *A* ***time-dependent utility function****, $U : \Phi \times \Psi \to \mathbb{R}$, represents the utility of a solution of quality $q \in \Phi$ at time step $t \in \Psi$.*

A time-dependent utility function can often be simplified by expressing it as the difference between two functions typically referred to as *object-level utility* and *inference-level utility* [64]. Object-level utility represents the utility of a solution by considering its quality but disregarding its computation time while inference-level utility represents the utility of a solution by taking into account its computation time but ignoring its quality. Therefore, adopting more recent terminology [121], we define this property as follows [60].

Definition 2. *A time-dependent utility function, $U : \Phi \times \Psi \to \mathbb{R}$, is* ***time-separable*** *if the utility of a solution of quality $q \in \Phi$ at time step $t \in \Psi$ can be expressed as the difference between two functions, $U(q,t) = U_I(q) - U_C(t)$, where $U_I : \Phi \to \mathbb{R}^+$ is the* ***intrinsic value function*** *and $U_C : \Psi \to \mathbb{R}^+$ is the* ***cost of time****.*

Given a time-dependent utility function, the meta-level control problem is the problem in which an autonomous system must decide when to interrupt an anytime algorithm and act on the current solution. Figure 3.1 provides an illustration of the meta-level control problem [177]. In this illustration, the algorithm should ideally be interrupted at the optimal stopping point because this is the maximum point of the time-dependent utility function. In practice, however, the optimal stopping point can rarely be determined as a result of considerable uncertainty over two variables: the performance of the algorithm and the urgency for the solution. The optimal stopping condition must therefore be approximated using an approach that models either or both of these variables. Note that we assume that there is only uncertainty about the performance of the algorithm throughout this chapter following earlier work [60].

3.3 Model-Based Metareasoning with Online Performance Prediction

We first propose a novel *model-based* meta-level control approach based on *online performance prediction*. This section offers an online performance prediction framework that can be used by a meta-level control technique to predict the performance of an anytime algorithm. Existing meta-level control techniques instead use a performance profile as a function in terms of a state of computation that represents the computation time and the quality of the current solution to predict the expected quality of the next solution [37, 67, 60]. Relying on a performance profile, however, has several drawbacks because it requires significant preprocessing that can decrease the accuracy and usefulness of meta-level control in real world domains.

We define a pair of vectors that jointly represent the performance of an anytime algorithm in place of a performance profile. The first vector represents the *past* performance of the anytime algorithm as it currently solves a specific instance of a problem. Past performance can be expressed as a vector of solution qualities from the *initial* solution to the *current* solution observed over the *elapsed* time of the anytime algorithm. More formally, we describe past performance in the following way.

Definition 3. *A* ***performance history****, $\vec{h}$, represents the past performance of an anytime algorithm on a given instance of a problem as a vector of solution qualities, $\vec{h} = [q_0\ q_1\ \ldots\ q_t]$, observed from the start time step 0 to the current time step t at fixed intervals of a duration Δ.*

The second vector represents the *future* performance of an anytime algorithm as it currently solves a specific instance of a problem. Future performance can be expressed as a vector of solution qualities from the *current* solution to the *final* solution projected over the *remaining* time of the anytime algorithm. More formally, we describe future performance in the following way.

Definition 4. *A* ***performance projection****, $\vec{p}$, represents the future performance of an anytime algorithm on a given instance of a problem as a vector of solution qualities, $\vec{p} = [q_{t+1}\ q_{t+2}\ \ldots\ q_T]$, projected from the time step $t+1$ to the final time step T at fixed intervals of a duration Δ such that the final time step T is an upper bound on the computation time allocated to the anytime algorithm.*

We use the *past* performance of the anytime algorithm on the specific instance of the problem currently being solved to predict its *future* performance. This can be viewed as a function that computes a *performance projection* from a *performance history*. It is possible for this function to be implemented in many different ways. In most cases, a simple method, such as nonlinear regression, can compute a sensible performance projection from a performance history. In other cases, a more complex method that uses a rich model, such as neural network or a regression tree, that includes features that describe the problem, the anytime algorithm, or the underlying system can be used instead [15, 68, 172]. However, a rich model must be adapted to an online context to avoid the drawbacks of a performance profile. It is also possible for this function to compute performance projections from a weighted performance history with a bias toward recent solution qualities. Here, without committing to a specific implementation, we provide a general definition of this function below.

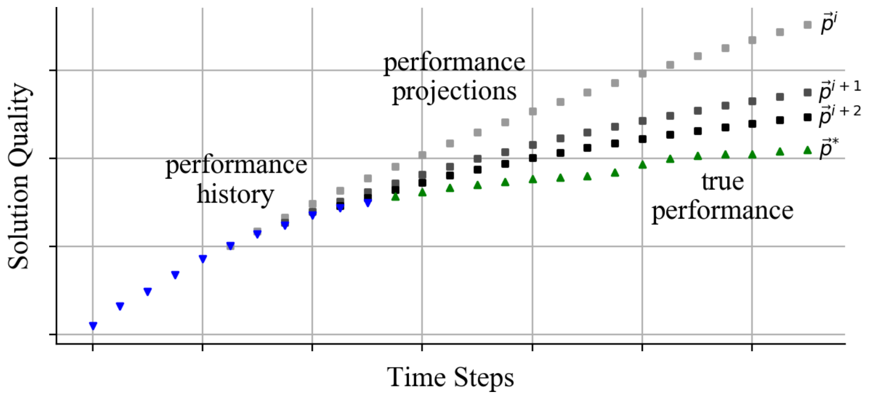


Figure 3.2: An illustration of online performance prediction.

Definition 5. *A **performance predictor**, $\Xi(\vec{h}) = \vec{p}$, generates a performance projection $\vec{p}$ from a performance history $\vec{h}$.*

Figure 3.2 offers an intuitive depiction of a performance predictor. Recall that the *performance history* is a sequence of solution qualities observed from the start time step to the current time step of the anytime algorithm while each *performance projection* is a sequence of solution qualities projected after the current time step to the final time step of the anytime algorithm. Ideally, the performance predictor computes performance projections that approach the true performance of an anytime algorithm as the size of the performance history increases. For instance, at the ith time step, the performance projection $\vec{p}^i$ does not closely approximate the true performance $\vec{p}^*$: in fact, it appears to be overly optimistic. However, at the $(i+1)$th time step, the next performance projection $\vec{p}^{i+1}$ draws closer to the true performance $\vec{p}^*$. Intuitively, as the performance predictor exploits more information from the solution qualities in the performance history, we observe that the performance projections approach the true performance of an anytime algorithm in practice in our experiments.

3.4 Model-Based Meta-Level Control Technique

This section offers a model-based meta-level control technique that determines when to interrupt an anytime algorithm and act on the current solution by using

Algorithm 1: A general approach to model-based meta-level control that *determines* when to interrupt an anytime algorithm and act on the current solution by using online performance prediction.

```
   Input: An anytime algorithm $\Lambda$, a performance predictor $\Xi$, a projected
          stopping condition $C$, and a duration $\Delta$
   Output: A solution $\alpha$
1  $t \leftarrow 0$
2  $\vec{h} \leftarrow [\,]$
3  $\Lambda.\mathit{Start}()$
4  while $\Lambda.\mathit{Running}()$ do
5      $\alpha \leftarrow \Lambda.\mathit{CurrentSolution}()$
6      $q \leftarrow \alpha.\mathit{Quality}()$
7      $\vec{h} \leftarrow \vec{h} \| q$
8      $\vec{p} = \Xi(\vec{h})$
9      if $C(\vec{p})$ then
10         $\Lambda.\mathit{Stop}()$
11         return $\alpha$
12     $t \leftarrow t + \Delta$
13     $\mathit{Sleep}(\Delta)$
14 return $\alpha$
```

the online performance prediction framework. Similar to earlier work, our meta-level control technique monitors the performance of the anytime algorithm and estimates the stopping point at runtime [67, 28, 182, 60]. However, unlike existing meta-level control techniques that rely on planning with a performance profile that must be compiled offline before the activation of meta-level control, our meta-level control technique uses the online performance prediction framework: at each monitoring step, it computes a performance projection from a growing performance history using a performance predictor. Our meta-level control technique therefore avoids relying on significant preprocessing that can decrease the accuracy and usefulness of meta-level control of anytime algorithms.

Algorithm 1 outlines the general form of our meta-level control technique. First, the current time step is initialized, the performance history is initialized to an empty vector, and the anytime algorithm is started (Lines 1-3). Next, the performance of the algorithm is monitored at fixed intervals until interrupted early or terminated naturally (Line 4). During each monitoring step, there are several steps. The quality of the current solution is appended to the performance history (Lines 5-7) and a performance projection is computed from that performance history using the performance predictor (Line 8). If the performance projection meets the stopping condition, which we describe in detail later in the chapter, the anytime algorithm is interrupted and the current solution is returned (Lines 9-11). Otherwise, the anytime algorithm continues to run (Lines 12-13). Finally, the current solution is returned if the anytime algorithm is terminated naturally but not interrupted early (Line 14).

In general, any meta-level control technique uses a stopping condition to determine whether or not an anytime algorithm should be interrupted. If the stopping condition evaluates to true, the meta-level control technique interrupts the anytime algorithm. Otherwise, it lets the anytime algorithm continue to run. An optimal stopping condition interrupts the anytime algorithm when the *expected value of computation* (EVC) is no longer positive, where the EVC can be expressed as the expected improvement of the time-dependent utility of the current solution [67]. However, to calculate the EVC, the meta-level control technique must consider the entire sequence of remaining decisions about whether to continue or interrupt the algorithm. Hence, because such a calculation is often intractable, meta-level control uses an estimate of the EVC in practice. Given this line of reasoning, our meta-level control technique uses a stopping condition that depends on the projected performance of an anytime algorithm. We call this a *projected stopping condition* and denote it as $C(\vec{p})$ in Algorithm 1. We develop two projected stopping conditions below.

3.4.1 Myopic Projected Stopping Condition

The first projected stopping condition uses the projected one-step improvement of the time-dependent utility of the current solution to determine whether or not the anytime algorithm should be interrupted. More formally, this myopic improvement can be expressed as the difference between two time-dependent utilities: the time-dependent utility of the projected *next* solution and the time-dependent utility of the *current* solution. We define myopic improvement in the following way.

Definition 6. *Suppose that an anytime algorithm computes a solution of quality $q \in \Phi$ at time step $t \in \Psi$. The **myopic projected value of computation** (MPVC) is*

$$MPVC(q, t, \Delta) = U(p_{t+\Delta}, t + \Delta) - U(q, t)$$

for an additional duration Δ given the current performance projection $\vec{p}$.

Our meta-level control technique allows the anytime algorithm to continue to run while the $MPVC$ is positive: simply put, the anytime algorithm executes as long as the projected one-step improvement of the current solution is positive. We define our myopic stopping condition below.

Definition 7. *The meta-level control technique with the **myopic stopping condition** lets an anytime algorithm continue as long as $MPVC(q, t, \Delta) > 0$.*

We call this version of our technique the *myopic* meta-level control technique.

If the performance of the anytime algorithm is strictly concave, our myopic meta-level control technique will interrupt the algorithm at the optimal stopping point near the global maximum of the time-dependent utility. Intuitively, when the performance of the anytime algorithm is concave, the benefit of continuing the anytime algorithm diminishes over time. Thus, if our myopic meta-level control technique decides to interrupt the anytime algorithm, that decision will remain optimal at any later time step. However, the performance of the anytime algorithm often includes steps with little or no improvement. In this more likely case, since our myopic meta-level control technique only considers the very next solution, it may interrupt the anytime algorithm too early near a local maximum of the time-dependent utility. Hence, we define a more accurate but more computationally demanding projected stopping condition to relax the assumption that the performance of the anytime algorithm is concave.

3.4.2 Nonmyopic Projected Stopping Condition

The second projected stopping condition improves upon the first condition by considering the projected *best* solution instead of the projected *next* solution. More formally, this nonmyopic improvement can be expressed as the difference between two time-dependent utilities: the time-dependent utility of the projected *best* solution and the time-dependent utility of the *current* solution. We define nonmyopic improvement in the following way.

Definition 8. *Suppose that an anytime algorithm computes a solution of quality $q \in \Phi$ at time step $t \in \Psi$. The* ***nonmyopic projected value of computation*** *(NPVC) is*

$$NPVC(q,t) = \max_{t' \in \Psi} U(p_{t'}, t') - U(q,t)$$

given the current performance projection $\vec{p}$.

Our meta-level control technique allows the anytime algorithm to continue to run while the $NPVC$ is positive: intuitively, even if the projected one-step improvement of the current solution is zero or negative, the anytime algorithm executes as long as the projected future improvement of the current solution is positive. We define our nonmyopic stopping condition below.

Definition 9. *The meta-level control technique with the* ***nonmyopic stopping condition*** *lets an anytime algorithm continue as long as $NPVC(q,t) > 0$.*

We call this version of our technique the *nonmyopic* meta-level control technique.

Our nonmyopic meta-level control technique is not as shortsighted as our myopic meta-level control technique. Simply put, because the nonmyopic stopping condition uses the projected best solution in place of the projected next solution, our nonmyopic meta-level control technique is less likely to interrupt the anytime algorithm too early near a local maximum of the time-dependent utility. As a result, even when the performance of the anytime algorithm includes steps with little or no improvement, our nonmyopic meta-level control technique will still interrupt the anytime algorithm closer to the optimal stopping point near the global maximum of the time-dependent utility. This results in more effective meta-level control of anytime algorithms.

3.5 Model-Free Metareasoning with Reinforcement Learning

We now turn to a novel *model-free* approach to meta-level control based on *reinforcement learning*. This section offers a formal MDP representation of the meta-level control problem for anytime algorithms that can be used to learn the optimal stopping point of an anytime algorithm with reinforcement learning. Reinforcement learning has led to a variety of methods [140] that have been effective across a range of applications from game playing [152] to helicopter control [81]. In order to maximize some notion of value, a reinforcement learning agent can learn a policy by performing actions and observing rewards in its environment both *online* and *incrementally*. This is critical to meta-level control of anytime algorithms for two reasons. First, because there is often not enough time before the activation of meta-level control in real-time environments, the policy must be compiled *online*. Second, since the parameters of meta-level control often change over time in dynamic environments, the policy must be updated *incrementally*. Hence, reinforcement learning is a natural approach to meta-level control of anytime algorithms.

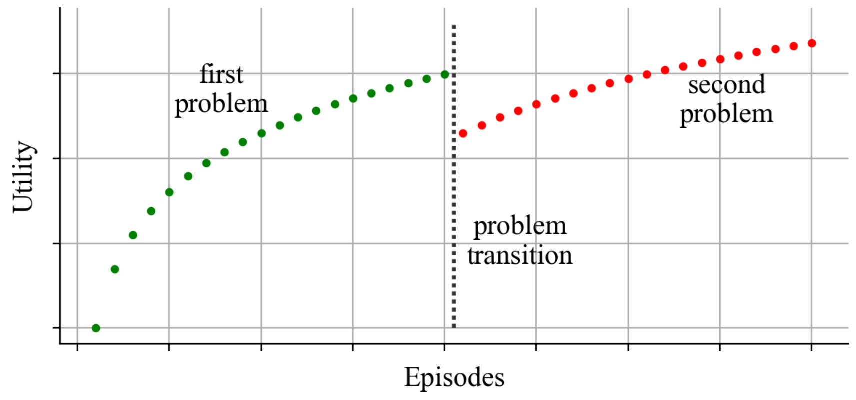


Figure 3.3: A depiction of model-free meta-level control.

Figure 3.3 illustrates how a meta-level control technique that uses a reinforcement learning method learns its policy for a problem with parameters that change gradually over time. Suppose the meta-level control technique compiles its policy for the problem online (the *green* section of the *first* problem). When the parameters of the problem change (the *problem transition*), the policy of the meta-level control technique may degrade in performance. In response, the meta-level control technique updates its policy for the problem incrementally (the *red* section of the *second* problem). Generally, if there is insufficient time before the activation of meta-level control or the parameters of meta-level control change over time, the meta-level control technique can learn its policy on the fly from scratch.

Although we are unaware of any model-free approach to meta-level control based on reinforcement learning, our approach, which is compatible with work on managing the execution of different planning models and methods [148, 147], is an especially good fit for several reasons. First, while the transition dynamics given the performance of an anytime algorithm may be unknown, reinforcement learning can learn an effective policy by balancing exploitation with exploration. Next, by ignoring large regions of the state space unlikely to be reached in practice, reinforcement learning can reduce the overhead of learning an effective policy by learning a *partial* policy that covers only reachable regions of the state space instead of a *universal* policy that covers the entire state space. Finally, while the transition dynamics given the performance of an anytime algorithm may be nonstationary, reinforcement learning can maintain an effective policy by making minor adjustments with negligible overhead. In short, meta-level control shares many properties with problems that have traditionally been solved by reinforcement learning.

Our model-free approach to meta-level control based on reinforcement learning expresses the meta-level control problem as an MDP. We now provide a formal description of the meta-level control problem by representing it as an MDP below.

Definition 10. *The* ***meta-level control problem for monitoring and controlling an anytime algorithm****, Λ, is represented by an MDP $\langle \Phi, \Psi, S, A, T, R, s_0 \rangle$ given a time-dependent utility function $U : \Phi \times \Psi \rightarrow \mathbb{R}$:*

- $\Phi = \{q_0, q_1, \ldots, q_{N_\Phi}\}$ *is a set of qualities for the current solution of the anytime algorithm.*
- $\Psi = \{t_0, t_1, \ldots, t_{N_\Psi}\}$ *is a set of time steps for the current solution of the anytime algorithm.*
- $S = \Phi \times \Psi$ *is a set of states of anytime computation: each state $s \in S$ indicates that the anytime algorithm has a solution of quality $q \in \Phi$ at time step $t \in \Psi$.*
- $A = \{\text{Stop}, \text{Continue}\}$ *is a set of actions of anytime computation: the* Stop *action interrupts the anytime algorithm and the* Continue *action executes the anytime algorithm for another time step of duration Δ.*
- $T : S \times A \times S \rightarrow [0, 1]$ *is a transition function of anytime computation that is unknown and possibly nonstationary that represents the probability of reaching a state $s' = (q', t') \in S$ after performing an action $a \in A$ in a state $s = (q, t) \in S$.*
- $R : S \times A \times S \rightarrow \mathbb{R}$ *is a* ***reward function of anytime computation*** *that we define later in this chapter that represents the expected immediate reward of*

reaching a state $s' = (q', t') \in S$ after performing an action $a \in A$ in a state $s = (q, t) \in S$.

- *$s_0 \in S$ is an optional start state that is typically $s_0 = (q_0, t_0)$ that indicates that the anytime algorithm has a solution of quality $q_0 \in \Phi$ at time step $t_0 \in \Psi$.*

Note that the discount factor γ is set to 1 because the meta-level control problem has an indefinite horizon since an anytime algorithm must terminate eventually [57].

The meta-level control problem has a **reward function of anytime computation** that describes the reward that is generated for each solution computed by the anytime algorithm. This can be represented as a piecewise function of two components. First, if the action is to execute the anytime algorithm for another time step, the reward is the difference between the utility of the current solution and the utility of the previous solution. Second, if the action is to interrupt the anytime algorithm immediately, the reward is nil. We define the reward function in the following way.

Definition 11. *Given a state of anytime computation $s = (q, t) \in S$, an action of anytime computation $a \in A$, and a successor state of anytime computation $s' = (q', t') \in S$, the **reward function of anytime computation** can be represented by the piecewise function*

$$R(s, a, s') = \begin{cases} U(q', t') - U(q, t), & \text{if } a = \textsc{Continue}, \\ 0, & \textit{otherwise}, \end{cases}$$

where $U : \Phi \times \Psi \to \mathbb{R}$ is a time-dependent utility function.

Note that it is easy to verify that the reward of anytime computation is consistent with the objective of optimizing time-dependent utility: running the anytime algorithm until a solution of quality $q \in \Phi$ at time step $t \in \Psi$ results in a cumulative reward of anytime computation equal to the time-dependent utility $U(q, t)$.

The main objective of the meta-level control problem is to generate an optimal policy that performs optimal meta-level control of the anytime algorithm under a set of assumptions. Intuitively, given the assumption that the quality and computation time of the current solution determines the quality and computation time of the successor solution without any need for the history of solutions up until the current solution, the meta-level control problem generates an optimal policy that produces optimal meta-level control of the anytime algorithm. We formalize this notion below.

Remark 1. *If the quality $q_t \in \Phi$ and time step $t \in \Psi$ of the current solution determines the quality $q_{t+1} \in \Phi$ and time step $(t+1) \in \Psi$ of the successor solution without any need for the history of solutions $[(q_0, 0), (q_1, 1), \ldots, (q_{t-1}, t-1)]$ up until the current solution, the optimal policy $\pi^* : S \to A$ of the meta-level control problem determines the optimal stopping point t^* of the anytime algorithm.*

Proof Sketch. This follows directly from the Markov assumption of an MDP: the transition dynamics over a successor state of computation $s' = (q', t') \in S$ only depends on the current state of computation $s = (q, t) \in S$ given an action of anytime computation $a \in A$. $\square$

Although many approaches to meta-level control have traditionally represented the state of computation as the quality and time step of the current solution, such a representation may not be sufficient since the improvement in the quality of the solution given its current quality and time step may not be Markovian. This representation could therefore benefit from additional features that describe the state of computation. In particular, it could include features that summarize the internal state of the algorithm, such as the size of the open list of anytime A* [58], the instance of the problem, such as the cluster distance of a TSP [75], or the performance of the system, such as the memory pressure of the system. While our model-free approach exhibits near optimal performance and fast convergence with a simple state of computation in our experiments, it can naturally augment the state of computation by using reinforcement learning. To this end, we describe in detail the use of a more sophisticated state of computation in the following chapter.

3.6 Model-Free Meta-Level Control Technique

This section offers a model-free meta-level control technique that learns when to interrupt an anytime algorithm and act on the current solution by using the formal MDP representation of the meta-level control problem for anytime algorithms. Similar to earlier work and the model-based meta-level control technique proposed earlier in the chapter, our meta-level control technique monitors the performance of the anytime algorithm and estimates the stopping point at runtime [67, 28, 182, 60]. However, unlike existing meta-level control techniques that rely on planning with a performance profile that must be compiled offline before the activation of meta-level control, our meta-level control technique uses reinforcement learning to learn the policy online and incrementally instead: it builds its policy gradually using the reward of anytime computation each time the anytime algorithm updates its solution to the instance of the problem at hand. By replacing offline compilation with online learning, our meta-level control technique eliminates the unrealistic assumptions of existing meta-level control techniques that reduce their accuracy and usefulness in the real world.

Algorithm 2 outlines the general form of our meta-level control technique. First, the state is initialized using the initial quality and time step, the action is initialized using the policy induced by the initial action-value function and the exploration strategy, and the anytime algorithm is started for a fixed duration (Lines 1-4). Next, the performance of the anytime algorithm is monitored at fixed intervals at interrupted early or terminated naturally (Line 5). During each monitoring step, there are several steps. The current solution is first retrieved from the anytime algorithm (Line 6). The successor state is then built using the new quality and time step (Line 7). The

Algorithm 2: A general approach to model-free meta-level control that *learns* when to interrupt an anytime algorithm and act on the current solution by using reinforcement learning.

```
Input: An anytime algorithm $\Lambda$, an action-value function $Q$, an update rule
       $\rho$, an exploration strategy $\xi$, and a duration $\Delta t$
Output: A solution $\sigma$
1  $s = (q, t) \leftarrow s_0$
2  $a \leftarrow \pi_\xi^Q(s)$
3  $\Lambda.\mathit{Start}()$
4  $\mathit{Sleep}(\Delta t)$
5  while $\Lambda.\mathit{Running}()$ do
6      $\sigma \leftarrow \Lambda.\mathit{CurrentSolution}()$
7      $s' = (q', t') \leftarrow (\sigma.\mathit{Quality}(), t + \Delta t)$
8      $r \leftarrow R(s, a, s')$
9      $\rho(Q, r, \alpha)$
10     $a \leftarrow \pi_\xi^Q(s')$
11     if $a = $ STOP then
12         $\Lambda.\mathit{Stop}()$
13         return $\sigma$
14     $s \leftarrow s'$
15     $\mathit{Sleep}(\Delta t)$
16 return $\sigma$
```

reward of anytime computation is subsequently calculated using the state, the action, and the successor state (Line 8). The action-value function is in turn updated using the update rule based on the reward of anytime computation and the learning rate (Line 9). An action is once again selected from the policy induced by the updated action-value function and the exploration strategy (Line 10). If the action indicates to stop, the anytime algorithm is interrupted and the current solution is returned (Lines 11-13). Otherwise, the state is set to the successor state and the anytime algorithm continues to run for a fixed duration (Lines 14-15). Finally, the current solution is returned if the anytime algorithm is terminated naturally but not interrupted early (Line 16). Note that the action-value function can easily be represented by a table [140] or approximated by a linear [83] or nonlinear function [100].

Our meta-level control technique has been generalized to support many reinforcement learning methods. It can use on-policy and off-policy temporal difference (TD) learning methods like TD(λ) and SARSA(λ) [139, 33] as well as exploration strategies like ϵ-greedy and softmax action selection [140]. While we do not commit to a specific reinforcement learning method, we describe our meta-level control technique using ϵ-*greedy Q-learning* [162] as an example because it has been analyzed extensively and proven effective across many applications [137, 151, 96]. We discuss the update rules and the exploration strategies of our meta-level control technique below.

3.6.1 Update Rules

A reinforcement learning agent can update its action-value function by following an *update rule* that uses a reward signal emitted by the environment. In Algorithm 2, when a new solution is computed in each monitoring step, our meta-level control technique updates the action-value function using the update rule based on the reward of anytime computation and the learning rate. However, when the anytime algorithm is interrupted, our meta-level control technique does not update the action-value function because there is no change in the solution.

3.6.1.1 ϵ-greedy Q-learning Example

Given an action-value function Q, a reward of anytime computation $r = R(s, a, s')$, and a learning rate α, the update rule $\rho(Q, r, \alpha)$ is below:

$$Q(s, a) \overset{+}{\leftarrow} \alpha[r + \max_{a' \in A} Q(s', a') - Q(s, a)],$$

where the current state is $s = (q, t) \in S$, the current action is $a \in A$, and the successor state is $s' = (q', t') \in S$.

3.6.2 Exploration Strategies

A reinforcement learning agent can balance exploitation with exploration by following an *exploration strategy.* In Algorithm 2, when the action-value function is updated in each monitoring step, our meta-level control technique technique updates the policy using the action-value function and the exploration strategy.

3.6.2.1 ϵ-greedy Q-learning Example

The greedy policy must first be calculated. This policy can be built by performing a one-step lookahead over every action available at the current state. Given an action-value function Q, the greedy policy $\pi^Q(s)$ is calculated below:

$$\pi^Q(s) \leftarrow \arg\max_{a \in A} Q(s, a),$$

where the current state is $s = (q, t) \in S$.

Finally, once the greedy policy has been calculated, it can be modified to follow ϵ-greedy exploration by introducing randomness. Given an exploration probability ϵ and a greedy policy π^Q, the ϵ-greedy policy $\pi^Q_\xi(s)$ is calculated below:

$$\pi^Q_\xi(s) = \begin{cases} \pi^Q(s), & \textit{with probability 1 - } \epsilon, \\ \text{random}(A), & \textit{otherwise}, \end{cases}$$

where the current state is $s = (q, t) \in S$.

3.7 Experiments

In this section, we evaluate our model-based and model-free approach to meta-level control of anytime algorithms on several common benchmark domains and a mobile robot domain. In each common benchmark domain, an autonomous system solves the meta-level control problem for a given anytime algorithm on a specific problem: it must decide when to interrupt an anytime algorithm and act on the current solution. To do this, each trial runs two processes in parallel. The *object-level process* uses an anytime algorithm to solve an instance of the problem. At the same time, the *meta-level process* uses a meta-level control technique to monitor and control the anytime algorithm at fixed intervals. The trial is over once the anytime algorithm is either interrupted early or terminated naturally. All meta-level control techniques monitor approximately every tenth of a second.

Any meta-level control problem requires a time-dependent utility function. Similar to earlier work [60], given a solution of quality $q \in \Phi$ at time step $t \in \Psi$, the time-dependent utility can be defined as the function $U(q, t) = \alpha q - e^{\beta t}$, where the rates α and β are selected in practice based on the value of a solution (the intrinsic value function) and the urgency for a solution (the cost of time). These rates are selected deliberately to avoid trivializing the problem by making the urgency for a solution so low that the anytime algorithm runs to completion or so high that it is interrupted immediately. Note that the first term and the second term of the time-dependent utility function represents the intrinsic value function and the cost of time.

3.7.1 Domains

We discuss each common benchmark domain and the mobile robot domain below. Ideally, the quality of a solution can be defined as the approximation ratio, $q = c^*/c$, where c^* is the cost of the optimal solution and c is the cost of the current solution. However, since computing the cost of the optimal solution for complex problems is often infeasible, we estimate the quality of a solution as the approximation ratio, $q = \ell/c$, where ℓ is a problem-dependent lower bound on the cost of the optimal solution and c is the cost of the current solution like earlier work [60].

3.7.1.1 Lin-Kernighan Heuristic Domain

The first domain uses the *Lin-Kernighan heuristic* to solve travelling salesman problems (TSP). A TSP has a set of cities that must be visited using the shortest possible route where a distance is given for each pair of cities. The Lin-Kernighan heuristic is a standard tour improvement algorithm that starts with an initial tour and gradually improves that tour by swapping specific subtours until convergence [93]. Solution (tour) quality is approximated using the length of the minimum spanning tree of the TSP as the lower bound ℓ_{tsp}

3.7.1.2 Genetic Algorithm Domain

The second domain uses a *genetic algorithm* to solve job-shop problems (JSP). A JSP has a set of jobs composed of a sequence of tasks that must be scheduled on a set of machines. The genetic algorithm is a standard open-source Python implementation, *jsp-ga*, based on swap mutation and generalized order crossover used to solve JSPs approximately [36, 102, 24, 39]. Solution (schedule) quality is approximated using the time required to complete the longest job as the lower bound ℓ_{jsp}.

3.7.1.3 Simulated Annealing Domain

The third domain uses *simulated annealing* to solve quadratic assignment problems (QAP). A QAP has a set of facilities that must be assigned to a set of locations where a distance is given for each pair of locations and a flow is given for each pair of facilities. The simulating annealing algorithm is a standard open-source Fortran implementation, *QAPLIB*, used to solve QAPs approximately [164, 29, 99]. Solution (assignment) quality is approximated using the Gilmore-Lawler bound, the optimal cost of a linearized QAP [47], as the lower bound ℓ_{qap}.

3.7.1.4 Mobile Robot Domain

The fourth domain uses a path planning algorithm to solve path planning problems. Each path planning problem involves finding a path between a start location and a goal location that maximizes a measure of safety based on the probability of collision in a map with many obstacles and walls. The path planning algorithm is a standard open-source robotics C++ implementation, *epic* [168], based on harmonic function path planning in log-space that is designed for an iClebo Kobuki. Solution (path plan) quality is approximated using a measure of safety based on the probability of collision. Note that conducting experiments on an actual mobile robot ensures that our approaches produce meaningful behavior that is suitable for use on real systems.

3.7.2 Model-Based Evaluation

We first evaluate our model-based metareasoning approach by comparing it to the standard myopic and nonmyopic general-purpose offline planning meta-level control techniques that can be used with any anytime algorithm [60]:

- a myopic monitor that interrupts an anytime algorithm once an estimate of the EVC is no longer positive, and
- a nonmyopic monitor that interrupts an anytime algorithm once instructed to by a monitoring policy.

Note that, as discussed earlier, both general-purpose offline planning meta-level control techniques rely on a performance profile that must be compiled offline prior to the activation of meta-level control.

In contrast to existing meta-level control techniques that require substantial offline work, our approach only requires a simple performance predictor. We use a performance predictor based on nonlinear least squares regression with the model, $f(x; \vec{\theta}) = \theta_1 g(x + \theta_2) + \theta_3$, where the vector $\vec{\theta}$ contains the parameters of the model and the function g represents a nonlinear function, since it is a simple implementation

Type	Prediction	50-TSP	60-TSP	70-TSP	80-TSP	90-TSP
Nonmyopic	**Online**	**93.67**	**89.64**	**91.23**	**91.09**	**94.41**
	Offline	92.98	86.53	89.75	90.51	92.26
Myopic	**Online**	**80.47**	**70.29**	**63.07**	**59.61**	**59.91**
	Offline	60.98	43.66	37.88	39.29	42.53

Table 3.1: The average time-dependent *utility* for the best tour computed by the Lin-Kernighan heuristic on five TSPs with our model-based metareasoning approach.

Type	Prediction	20-JSP	40-JSP	60-JSP
Nonmyopic	**Online**	**119.19**	**102.62**	**101.23**
	Offline	115.64	95.57	94.05
Myopic	**Online**	**114.08**	**97.49**	**96.34**
	Offline	101.20	91.28	92.83

Table 3.2: The average time-dependent *utility* for the best schedule computed by the genetic algorithm on three JSPs with our model-based metareasoning approach.

Type	Prediction	100-QAP	150-QAP	200-QAP
Nonmyopic	**Online**	**165.55**	**167.29**	**164.78**
	Offline	162.78	163.91	162.84
Myopic	**Online**	**162.20**	**161.53**	**160.77**
	Offline	159.55	159.82	159.32

Table 3.3: The average time-dependent *utility* for the best assignment computed by simulated annealing on three QAPs with our model-based metareasoning approach.

of a performance predictor that we have observed to work well with many anytime algorithms. Moreover, because anytime algorithms generally exhibit the *diminishing returns* property [177], many logarithmic and sigmoidal functions work effectively. Here, for the performance predictor, we use the sigmoidal function $g(x) = \arctan(x)$ but have observed empirically that $g(x) = \log(x)$ is effective as well since many anytime algorithms exhibits diminishing returns. Note that, unlike a performance profile that must be built using a lengthy program, a performance predictor can be implemented in a few lines of code using the open-source Python library *SciPy*.

Type	Prediction	Office	Mine-S	Mine-L
Nonmyopic	**Online**	**88.43**	**87.90**	**86.74**
	Offline	79.12	56.02	70.83
Myopic	**Online**	**85.60**	**86.72**	**84.62**
	Offline	44.12	52.91	65.76

Table 3.4: The average time-dependent *utility* for the best path computed by the path planning algorithm on three maps with our model-based metareasoning approach.

On the common benchmark domains, all versions of our meta-level control technique are evaluated along their degree of optimality. Tables 3.1, 3.2, and 3.3 show the average time-dependent *utility* of the final solution for each version of our meta-level control techniques across 100 instances of all common benchmark problems. This means that *higher* average time-dependent utility signifies *better* performance. The results of the myopic and nonmyopic versions of the meta-level control techniques have been separated to ensure a fair comparison.

In the mobile robot domain, all versions of our meta-level control technique are evaluated along their degree of optimality. Table 3.4 shows the average time-dependent *utility* of the final solution for each of our meta-level control techniques across 100 instances of three path planning problems. Each problem uses a different map. The first map, Office, is a domain of an office in which the goal is impeded by many boxes, furniture, and walls. The other maps, Mine-S and Mine-L, are standard domains of coal mines generated with a mapping procedure [155]. Every instance of a problem has a random start position but the same goal position.

Figure 3.4 depicts the results of the experiments on an actual mobile robot. In this case, we run our nonmyopic meta-level control technique on the Office map. We only consider our nonmyopic meta-level control technique given its dominant performance in the mobile robot simulation. The four scenarios are associated with an infinite, high, low, and nil cost of time but the same start and goal position.

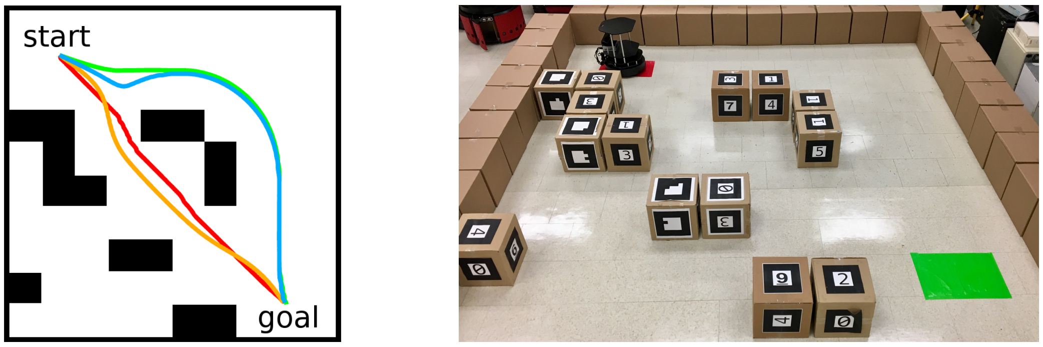


Figure 3.4: The Office map (*left*) with the riskiest path (*red*), a very risky path (*yellow*), a very safe path (*blue*), and the safest path (*green*) in addition to the environment of the mobile robot (*right*).

3.7.2.1 Discussion

On all common benchmark domains, not only does our meta-level control technique avoid offline work, but it also outperforms the state-of-the-art meta-level control techniques. Given similar results across every domain in Tables 3.1, 3.2, and 3.3, we focus our analysis on the Lin-Kernighan heuristic domain in Table 3.1. In the nonmyopic case, our meta-level control technique incurs a loss under 2% on every problem. As the size of the problem increases, the loss of our nonmyopic meta-level control technique remains steady while the existing meta-level control technique varies. In the myopic case, the difference between our meta-level control technique and the meta-level control existing technique is even larger. As the size of the problem increases, the myopic meta-level control technique degrades more slowly than the existing meta-level control technique as well. While all meta-level control techniques may be improved with a richer model using problem-specific instance features [75], it is very encouraging that we observe near optimal results given only computation time and solution quality as the state of computation.

Figure 3.5 illustrates the preprocessing time required to compile a performance profile and a monitoring policy for the state-of-the-art meta-level control techniques on the Lin-Kernighan heuristic domain. This shows that preprocessing time grows rapidly with the size of the problem. In fact, even for modest problems, compiling

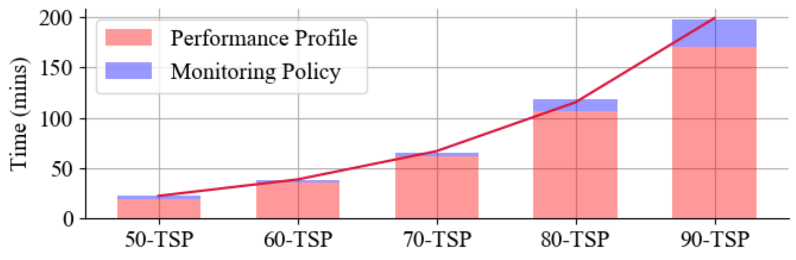


Figure 3.5: The preprocessing time of prevailing planning approaches.

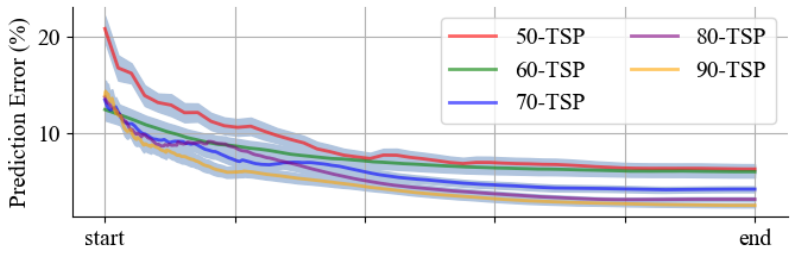


Figure 3.6: The change in the prediction error of our approach.

a performance profile and a monitoring policy can take hours of offline work. Furthermore, Figure 3.6 illustrates the improvement in the prediction error over time for our nonmyopic technique on the Lin-Kernighan heuristic domain. This shows that prediction error falls quickly with the size of the performance history. In particular, for all problems, the prediction error starts at less than 21% and ends at less than 7%. Note that the prediction error is expressed as the maximum difference between the current performance projection and the true performance of the anytime algorithm.

On the mobile robot domain, our meta-level control technique substantially outperforms the state-of-the-art meta-level control techniques. In simulation in Table 3.4, since the performance of the path planning algorithm is concave [168], our myopic meta-level control technique performs nearly as well as our nonmyopic meta-level control technique. In fact, our myopic meta-level control technique even outperforms the existing nonmyopic meta-level control technique due to large variation across instances of the problem, which is not captured by a performance profile. Crucially, on a mobile robot in Figure 3.4, our nonmyopic meta-level control technique effectively trades computation time with path safety. Given a high cost of time, the robot dangerously traverses through the boxes to the goal. However, given a low cost of time, the robot safely avoids the boxes entirely.

It may seem counterintuitive that our meta-level control technique outperforms state-of-the-art meta-level control techniques shown to be optimal [60]. The key idea is that existing meta-level control techniques assume that a performance profile is an *exact* model of the behavior of an anytime algorithm. There are a number of reasons, however, why such a model may not be adequate. First, existing meta-level control techniques assume that the model is accurate across different instances of a problem while our meta-level control technique adapts to each instance by using online performance prediction. Next, when existing meta-level control techniques are deployed, they do not perform as well because the model was compiled using some predicted distribution instead of the true but unknown distribution. Moreover, the model loses information about the performance of the algorithm as solution quality and computation time must be discretized into a small number of bins. Finally, since the model is compiled under specific CPU and memory conditions, existing meta-level control techniques become less accurate given variance in these conditions. Even without any offline work, our meta-level control technique avoids these pitfalls.

3.7.3 Model-Free Evaluation

We now evaluate our model-free metareasoning approach by comparing it to the prevailing nonmyopic general-purpose planning technique that can be used with any anytime algorithm [60]:

- a nonmyopic monitor that interrupts an anytime algorithm once instructed to by a monitoring policy.

Each version of our meta-level control technique uses a different reinforcement learning method with some function representation following ϵ-greedy action selection. In particular, we evaluate the following versions of our meta-level control technique:

- tabular SARSA,
- tabular Q-learning,
- Fourier basis SARSA, and
- Fourier basis Q-Learning.

Note that we experiment with tabular functions and linear approximations with Fourier basis for our meta-level control techniques because they are often the first to be tried by researchers and practitioners in reinforcement learning [83].

All of our meta-level control techniques begin with a randomized initial policy that is equally likely to stop or continue the anytime algorithm. This policy is updated as the meta-level control technique learns from 5000 random problem instances. The exploration probability ϵ is set to 0.1 with a decay of 0.999 while the learning rate α is set to 0.1 for our tabular meta-level control techniques and 0.00001 for our Fourier basis meta-level control techniques. It is also possible to design an initial policy that exploits the form of the time-dependent utility function in safety-critical domains. The planning meta-level control technique, however, uses a static policy that cannot be updated over time. This policy is calculated by applying dynamic programming to a performance profile compiled from 2000 random problem instances solved to *completion* prior to the activation of meta-level control.

On the common benchmark domains, all versions of our meta-level control technique are evaluated along three important dimensions: the degree of optimality, the rate of convergence, and the rate of adaptation. First, for the degree of optimality, Tables 3.5, 3.6, and 3.7 show the average time-dependent *utility loss* of the final solution for each version of our meta-level control techniques across 100 instances of all benchmark problems. This means that *lower* average time-dependent utility losses signify *better* performance. Next, for the rate of convergence, Figure 3.7 shows the change in the time-dependent utility of the policy for each of our meta-level con-

Method	50-TSP	60-TSP	70-TSP	80-TSP	90-TSP
Planning	11.40	15.27	8.95	10.68	11.96
SARSA(Table)	13.94	16.73	13.61	24.49	21.46
Q-learning(Table)	15.66	18.61	20.93	23.34	26.44
SARSA(Fourier)	**2.27**	6.77	3.75	**3.81**	5.68
Q-learning(Fourier)	2.69	**6.33**	**2.51**	5.64	**5.25**

Table 3.5: The average time-dependent *utility loss* for the best tour computed by the Lin-Kernighan heuristic on five TSPs with our model-free metareasoning approach.

Method	20-JSP	40-JSP	60-JSP
Planning	2.85	5.54	2.52
SARSA(Table)	18.26	17.23	15.33
Q-learning(Table)	18.17	16.96	14.43
SARSA(Fourier)	**2.11**	2.37	**1.38**
Q-learning(Fourier)	2.77	**1.88**	2.22

Table 3.6: The average time-dependent *utility loss* for the best schedule computed by the genetic algorithm on two JSPs with our model-free metareasoning approach.

Method	100-QAP	150-QAP	200-QAP
Planning	4.33	6.52	7.13
SARSA(Table)	4.13	3.97	4.39
Q-learning(Table)	3.36	3.95	3.52
SARSA(Fourier)	0.69	**0.51**	**1.12**
Q-learning(Fourier)	**0.36**	0.53	1.17

Table 3.7: The average time-dependent *utility loss* for the best assignment computed by simulated annealing on two QAPs with our model-free metareasoning approach.

trol techniques on select benchmark problems. Finally, for the rate of adaptation, Figure 3.8 shows the number of episodes required by each of our meta-level control techniques to adapt to a change in the parameters of select benchmark problems.

In the mobile robot domain, all versions of our meta-level control technique are evaluated along their degree of optimality. Table 3.8 shows the average time-dependent utility loss of the final solution for each of our meta-level control techniques

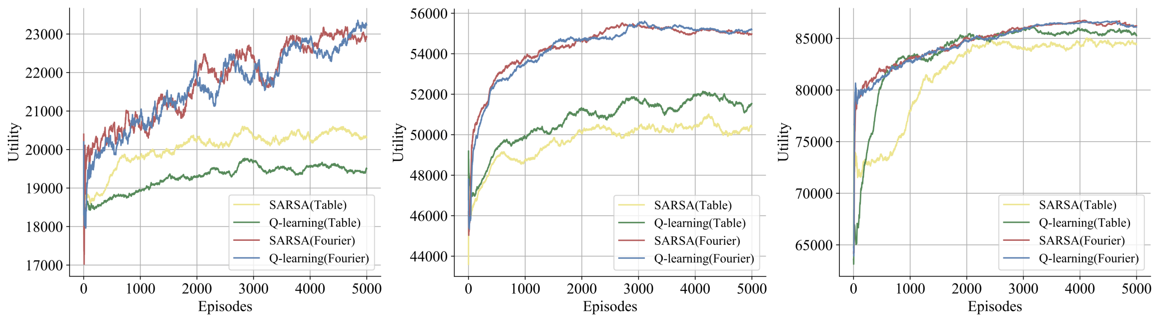


Figure 3.7: The learning curves for each of our meta-level control techniques on the 60-Tsp, 40-Jsp, and 150-Qap benchmark problems.

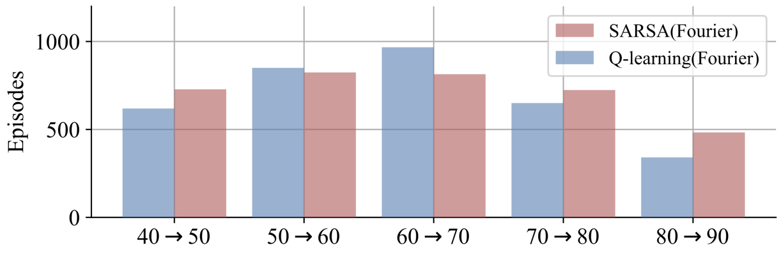


Figure 3.8: The adaptation period for each of our Fourier basis meta-level control techniques on all TSP benchmark problems.

across 100 instances of the three path planning problems. Recall that the first map, Office, is a domain of an office in which the goal is impeded by many boxes, furniture, and walls while the other maps, Mine-S and Mine-L, are standard domains of coal mines generated with a mapping procedure [155]. Again, every instance of a problem has a random start position but the same goal position.

Figure 3.9 shows a simulation of the mobile robot domain. An autonomous system can use a model-free approach to meta-level control based on reinforcement learning to learn when to interrupt a path planning algorithm and act on the current path plan. As the number of episodes increases, the utility of the current path plan approaches the utility of the final path plan that optimizes the trade-off between solution quality and computation time. It is important to highlight that the path plan at convergence requires minutes while the final path plan only involves seconds of deliberation.

Method	Office	Mine-S	Mine-L
Planning	12.02	10.64	11.02
SARSA(Table)	5.52	6.72	5.66
Q-learning(Table)	3.59	6.01	4.08
SARSA(Fourier)	2.95	3.37	**2.34**
Q-learning(Fourier)	**2.75**	**3.15**	3.13

Table 3.8: The average time-dependent *utility loss* for the best path computed by the path planning algorithm on three maps with our model-free metareasoning approach.

3.7.3.1 Discussion

Our model-free approach to meta-level control based on reinforcement learning outperforms the planning meta-level control technique across every domain. Given near optimal performance in Tables 3.5, 3.6, 3.7, and 3.8 and fast convergence in Figure 3.7, we focus on our meta-level control techniques that use a linear approximation with Fourier basis. Our meta-level control techniques incur a loss lower than 3% on most problems with an upper limit of 7% while the planning meta-level control technique incurs a loss higher than 10% on most problems with an upper limit of 16%. Our meta-level control techniques also have less variance compared to the planning meta-level control technique. Overall, although our approach can be improved in several ways, it is encouraging that it exhibits near optimal performance and fast convergence using standard reinforcement learning methods, simple function approximations, and naive exploration strategies.

Our approach also adapts to meta-level control problems with parameters that change over time. In Figure 3.8, our meta-level control techniques update their policies in under 1000 random instances to adapt to a change in the size of each TSP benchmark problem. The planning meta-level control technique, however, requires substantial offline work because it has to compile a completely new policy by applying dynamic programming to a performance profile prior to meta-level control.

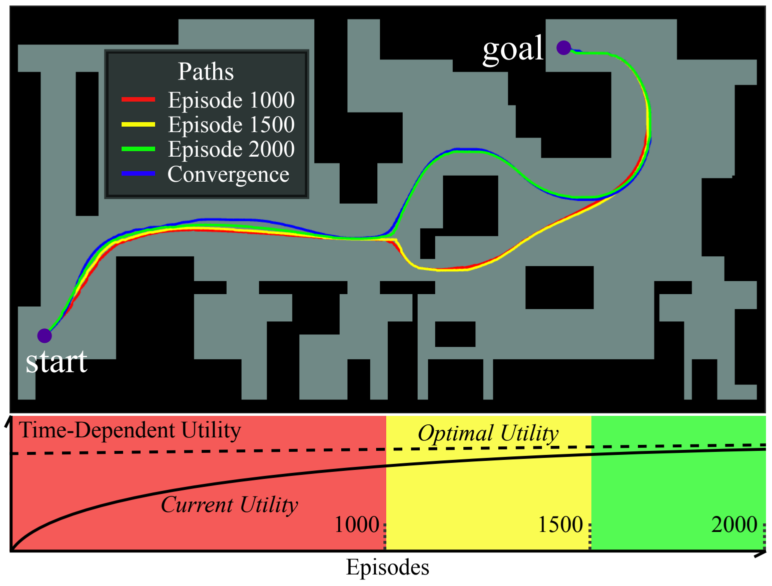


Figure 3.9: A simulation of the mobile robot domain.

Using reinforcement learning for model-free meta-level control offers a number of advantages over the traditional planning paradigm. First, when the parameters of meta-level control change over time, our approach can update its policy incrementally on the fly. This is critical since the settings of the anytime algorithm, the distribution of problem instances, and the CPU and memory conditions of the system often shift in practice. Moreover, when there is not enough time before the activation of meta-level control, our approach can compile its policy online from scratch. Most importantly, even if the parameters of meta-level control do *not* change over time and there *is* enough time before the activation of meta-level control, our approach still outperforms the planning paradigm by learning a significantly *more effective* policy in substantially *less time*. This is because reinforcement learning focuses on meaningful regions of the state space of the meta-level control problem in contrast to planning.

3.8 Summary

This chapter introduces two metareasoning approaches to stopping of anytime algorithms: a model-based approach that uses online performance prediction and a model-free approach that uses reinforcement learning. By learning the performance of the anytime algorithm using either a model-based or model-free approach, it is possible to avoid relying on significant preprocessing that can decrease the accuracy and usefulness of existing metareasoning methods for anytime algorithms in the real world. In our experiments, we show that both approaches outperform existing meta-level control techniques that require substantial offline work on several common benchmark domains and a mobile robot domain.

CHAPTER 4
METAREASONING FOR HYPERPARAMETER TUNING

4.1 Introduction

Building on our work in the previous chapter, we develop a metareasoning approach that enables autonomous systems to not only determine the optimal stopping point but also tune the hyperparameters of anytime algorithms at runtime. Naturally, anytime algorithms often have *hyperparameters* that can be tuned at runtime to boost their overall performance in a specific scenario—given a certain problem instance and a time constraint. As we described in detail earlier, the central property of an anytime algorithm is that it can be interrupted at any time to provide a solution that is gradually improved upon at runtime [177], which offers an important trade-off between the quality and computation time of a solution that has proven to be useful in a variety of real-time decision-making problems. Currently, to manage this trade-off, existing work on metareasoning has focused on determining when to interrupt an anytime algorithm and act on the current solution. However, the scope of metareasoning can ideally be expanded to tune the hyperparameters of an anytime algorithm at runtime in order to boost its overall performance in a specific scenario.

There has been a substantial body of work on developing metareasoning techniques that determine when to interrupt an anytime algorithm and act on the current solution as we discussed earlier. Generally, these methods monitor and control an anytime algorithm by tracking its performance and estimating its stopping point at runtime. For example, an early approach models optimal stopping as a sequential decision problem and derives a meta-level control policy using dynamic programming techniques [60]. Moreover, in the previous chapter, we offered a model-based approach and a model-free approach that estimate the optimal stopping point of an anytime algorithm using online performance prediction and reinforcement learning. All of these methods, however, cannot tune the hyperparameters of the anytime algorithm at runtime to improve its overall performance in a specific scenario.

Nevertheless, formal techniques for tuning the hyperparameters of an anytime algorithm at runtime have largely been designed for specific anytime algorithms. For instance, there have been methods for heuristically tuning the weight of an anytime heuristic search algorithm called *anytime weighted A** [58, 138, 153, 22] and methods for heuristically tuning the growth factor and area of focus of an anytime motion planning algorithm called *RRT** [156, 4, 80]. However, these methods have several drawbacks as they lack formal analysis or generality and require expertise in the implementation of the anytime algorithm. Most importantly, they do not fully utilize that an anytime algorithm continually computes solutions of a well-defined utility as it runs.

We therefore introduce a general, decision-theoretic metareasoning approach to both optimal stopping and optimal hyperparameter tuning for a generalization of an anytime algorithm that we call an *adjustable algorithm*. Our approach models the problem of monitoring and controlling an adjustable algorithm as a deep reinforcement learning problem, specifically an MDP, with two main attributes. Its *states* represent the quality and computation time of the current solution and any other features needed to summarize the internal state of the algorithm, the instance of the problem, or the performance of the system. Its *actions* represent either interrupting the algorithm or executing the algorithm for another time step while tuning its internal hyperparameters. Given this MDP, our meta-level control technique uses *deep reinforcement learning* to learn a policy for both optimal stopping and optimal hyperparameter tuning of the adjustable algorithm: it performs a series of episodes that each use the adjustable algorithm to solve a generated instance of a specific problem. Our experiments on distinct search paradigms, particularly anytime weighted A* for heuristic search and RRT* for motion planning, highlight that deep reinforcement learning is an effective approach to metareasoning for adjustable algorithms given the abundance of simulations that are generated readily.

4.1.1 Contributions

In this chapter, we make the following contributions: (1) a generalization of an anytime algorithm called an adjustable algorithm that can be interrupted at any time for its current solution and has hyperparameters that can be tuned at runtime, (2) a meta-level control technique that learns optimal stopping and optimal hyperparameter tuning of an adjustable algorithm by using deep reinforcement learning, (3) an example of our approach on anytime weighted A*, and (4) a set of experiments that show that our approach boosts overall performance on a common benchmark domain that uses anytime weighted A* to solve a range of heuristic search problems and a mobile robot application that uses RRT* to solve motion planning problems.

4.2 Related Work

There are two broad approaches to automatic hyperparameter tuning for general algorithms. *Model-based* approaches typically interleave fitting a model with selecting hyperparameters based on that model. Notably, building on earlier work in sequential model-based optimization [14, 72, 71], the *SMAC* method uses a model represented as a random forest to select the hyperparameters of an algorithm [70]. However, while *model-free* approaches do not use any model, they have still been shown to be effective across a range of applications. Limited to numerical hyperparameters, the *CALIBRA* method uses experimental designs to find initial hyperparameters followed by local search to improve those hyperparameters [1] while the *F-Race* method leverages racing algorithms from machine learning [25, 26] to select the hyperparameters of an algorithm. Extending to categorical hyperparameters, the *GGA* method employs parallel gender-based genetic algorithms [10] while the *ParamILS* method performs iterated local search [74, 73] to select the hyperparameters of an algorithm.

We highlight that these methods are largely designed for general algorithms and typically do not exploit anytime algorithms. By applying deep reinforcement learning to anytime algorithms in particular, our approach avoids many drawbacks often imposed by current methods. First, unlike methods that only support numerical hyperparameters and deterministic algorithms, our approach also supports both categorical hyperparameters and stochastic algorithms. Next, unlike methods that only optimize the hyperparameters of an algorithm on a *single* problem instance, our approach optimizes the hyperparameters of an algorithm on *multiple* problem instances. In fact, our approach adjusts the hyperparameters of an algorithm on a specific instance of a problem at runtime. Finally, unlike methods that always execute an algorithm until completion, our approach terminates an algorithm early if necessary.

An orthogonal line of work that focuses on using a portfolio of algorithms to solve different instances of hard computational problems has seen recent attention as well. This research recognizes that different algorithms tend to dominate each other on different instances of a problem because there is often no single best algorithm [89]. This has resulted in methods that can use *portfolios of algorithms* for satisfiability [50], *ensemble methods* in machine learning [40, 42], and *multiple methods* in real-time problem solving [160, 180]. Notably, SATzilla [172], an efficient solver that manages a portfolio of algorithms to solve difficult satisfaction problems, has won multiple competitions and has continued to dominate the field.

In the previous chapter, we discussed the body of work for optimal stopping of anytime algorithms that has grown over the last thirty years. In particular, early approaches based on *fixed allocation* execute the anytime algorithm until a stopping point determined prior to runtime [66, 27] in comparison to recent approaches based on *monitoring and control* that track the performance of the anytime algorithm and estimates a stopping point at runtime periodically [67, 182, 60, 92, 149, 146]. Our approach not only estimates the stopping point but also tunes the hyperparameters of an anytime algorithm at runtime to boost its overall performance.

Most recently, in the field of heuristic search, a couple techniques have been proposed for dynamically selecting the heuristic function by using deep reinforcement learning [136] and dynamically adjusting the search strategy in classical planning by using evolutionary strategies [51]. Moreover, in the field of motion planning, a technique has been proposed for learning the parameters of a planner from demonstration [171]. However, our approach focuses more broadly on anytime algorithms instead of just heuristic search as we will see in our experiments.

4.3 Adjustable Algorithms

We begin by proposing a generalization of an anytime algorithm called an *adjustable algorithm*. An adjustable algorithm has both *internal state* and *internal hyperparameters*. Simply put, a metareasoner can monitor the internal state of the algorithm and control the internal hyperparameters to either interrupt the algorithm or execute the algorithm for another time step while adjusting its internal operation. This results in a new meta-level control problem that involves both stopping and hyperparameter tuning of an adjustable algorithm. At a high level, we will see that our approach monitors and controls an adjustable algorithm by expressing this new meta-level control problem as a deep reinforcement learning problem. We define an adjustable algorithm is defined as follows.

Definition 12. *An* ***adjustable algorithm****,* Λ*, has internal state that can be monitored and internal hyperparameters* $\{\Theta_0, \Theta_1, \ldots, \Theta_{\ell_\Theta}\}$ *that can be controlled such that the internal hyperparameter* $\Theta_0 = \{\textsc{Stop}, \textsc{Continue}\}$ *either interrupts the algorithm*

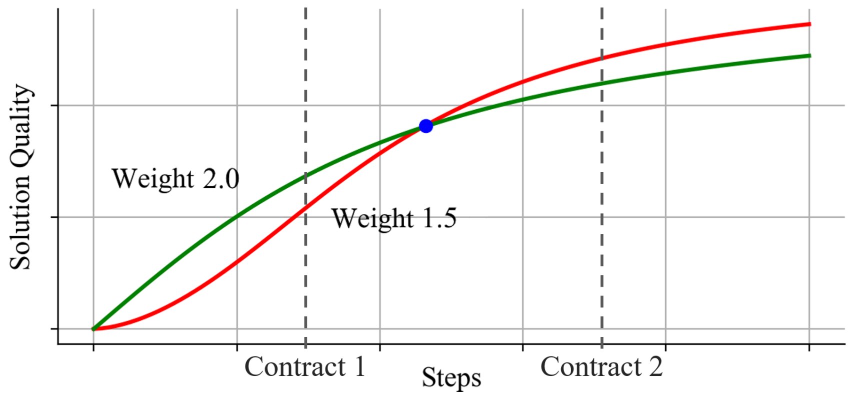


Figure 4.1: An example of two executions of anytime weighted A*.

or executes the algorithm for another time step and the internal hyperparameters $\{\Theta_1, \ldots, \Theta_{\ell_\Theta}\}$ *adjust the internal operation of the algorithm.*

We use *anytime weighted A** to illustrate our approach throughout the chapter. Anytime weighted A*, an anytime algorithm that extends the popular A* heuristic search algorithm [61, 91, 3, 58, 154], is an example of an adjustable algorithm. This algorithm (1) uses an inadmissible heuristic to quickly find suboptimal solutions, (2) continues the search after each suboptimal solution is found, (3) provides an error bound on each suboptimal solution when it is interrupted, and (4) guarantees an optimal solution once the open list is empty. Notably, the standard evaluation function $f(n) = g(n) + h(n)$ that is used to select the next node for expansion from the open list is replaced with a weighted evaluation function $f_w(n) = g(n) + w \cdot h(n)$, where the path cost function $g(n)$ is the cost of the path from the start node to a node n and the heuristic function $h(n)$ is the estimated cost from a node n to the goal node given a weight $w \geq 1$. Intuitively, by weighting the heuristic function $h(n)$ more heavily than the path cost function $g(n)$ for any weight $w > 1$, the algorithm prioritizes expanding nodes that appear closer to reaching any solution instead of nodes that lead to the optimal solution. This causes the algorithm to speed up computation time at the expense of solution quality.

Figure 4.1 shows typical performance curves of two runs of anytime weighted A* with different weights that each solve a given instance of a problem. With *deadlines*, a weight of 2.0 leads to better quality at *Contract 1* while a weight of 1.5 results in better quality at *Contract 2*. Without *deadlines*, a weight of 2.0 leads to better quality in the short term but worse quality in the long term while a weight of 1.5 results in worse quality in the short term but better quality in the long term. This poses an important question: is it possible to develop a metareasoning approach that tunes the hyperparameters of an adjustable algorithm at runtime to boost its overall performance *with* or *without* deadlines? In this chapter, we answer this question by offering a simple metareasoning framework for learning optimal stopping and optimal hyperparameter tuning of adjustable algorithms with deep reinforcement learning.

Our approach to monitoring and controlling adjustable algorithms based on deep reinforcement learning expresses the meta-level control problem as an MDP. This meta-level control problem, which can be viewed as an extension of the meta-level control problem for monitoring and controlling anytime algorithms defined in the previous chapter, is an MDP that has two central attributes. First, the the set of *states* has *state factors* that reflect the quality and computation time of the current solution along with *state factors* that reflect the internal state of the algorithm, the instance of the problem, or the performance of the system. Second, the set of *actions* has an *action factor* that reflects the internal hyperparameter that either interrupts the algorithm or executes the algorithm for another time step along with *action factors* that reflect the internal hyperparameters that adjust the internal operation of the algorithm. We now provide a formal description of the meta-level control problem by representing it as an MDP below.

Definition 13. *The* ***meta-level control problem for monitoring and controlling an adjustable algorithm****,* Λ*, is represented by an MDP* $\langle \Phi, \Psi, F, S, A, T, R, s_0 \rangle$ *given a time-dependent utility function* $U : \Phi \times \Psi \to \mathbb{R}$*:*

- $\Phi = \{q_0, q_1, \ldots, q_{N_\Phi}\}$ *is a set of qualities for the current solution of the adjustable algorithm.*

- $\Psi = \{t_0, t_1, \ldots, t_{N_\Psi}\}$ *is a set of time steps for the current solution of the adjustable algorithm.*

- $F = F_0 \times F_1 \times \cdots \times F_{N_F}$ *is a set of features that summarize the internal state of the algorithm, the instance of the problem, or the performance of the system.*

- $S = \Phi \times \Psi \times F$ *is a set of states of adjustable computation: each state* $s \in S$ *indicates that the adjustable algorithm has a solution of quality* $q \in \Phi$ *at time step* $t \in \Psi$ *with a feature* $f \in F$*.*

- $A = \Theta_0 \times \Theta_1 \times \cdots \times \Theta_{N_\Theta}$ *is a set of actions of adjustable computation: the internal hyperparameter* $\Theta_0 = \{\textsc{Stop}, \textsc{Continue}\}$ *either interrupts the adjustable algorithm or executes the adjustable algorithm for another time step of duration* Δ *while the internal hyperparameters* $\Theta_1, \ldots, \Theta_{N_\Theta}$ *adjust the internal operation of the adjustable algorithm.*

- $T : S \times A \times S \to [0, 1]$ *is a transition function of adjustable computation that is unknown and possibly nonstationary that represents the probability of reaching a state* $s' = (q', t', f') \in S$ *after performing an action* $a \in A$ *in a state* $s = (q, t, f) \in S$*.*

- $R : S \times A \times S \to \mathbb{R}$ *is a reward function of adjustable computation that represents the expected immediate reward,* $R(s, a, s') = U(q', t') - U(q, t)$*, of reaching a state* $s' = (q', t', f') \in S$ *after performing an action* $a \in A$ *in a state* $s = (q, t, f) \in S$*.*

- $s_0 \in S$ *is an optional start state that is typically* $s_0 = (q_0, t_0, f_0) \in S$ *that indicates that the adjustable algorithm has a solution of quality* $q_0 \in \Phi$ *at time step* $t_0 \in \Psi$ *with a feature* $f_0 \in F$*.*

Observe that, following the previous chapter, the discount factor γ of the MDP is set to 1 because the meta-level control problem has an indefinite horizon since an adjustable algorithm must terminate eventually [57]. Moreover, it is easy to verify that the reward of adjustable computation is consistent with the objective of optimizing time-dependent utility: running the adjustable algorithm until a solution of quality $q \in \Phi$ at time step $t \in \Psi$ with a feature $f \in F$ results in a cumulative reward of adjustable computation equal to the time-dependent utility $U(q, t)$.

Similar to the previous chapter, the meta-level control problem here recognizes that the state of computation as only the quality and computation time of the current solution [182] will likely not be Markovian. Consequently, it is clear that the state of computation could be augmented with features that summarize the internal state of algorithm, the instance of the problem, or the performance of the system. As an example, in a domain that uses anytime weighted A* to solve an instance of a TSP, there could be features for the mean of the g- and h-values on the open list of anytime weighted A*, the number of cities in the instance of the TSP [75], or the processor usage of the system. Our approach to adjustable algorithms can use a complex representation with a range of features for the state of computation to better approximate the Markov property by using deep reinforcement learning.

4.4 Metareasoning with Deep Reinforcement Learning

We now offer a meta-level control technique that uses deep reinforcement learning to monitor and control adjustable algorithms. Here, our meta-level control technique uses deep reinforcement learning to learn both optimal stopping and optimal hyperparameter tuning of an adjustable algorithm: it performs a series of episodes that each use the adjustable algorithm to solve a generated instance of a specific problem.

Deep reinforcement learning has been effective across a variety of applications, including Atari [100], chess [131], and StarCraft [159]. A deep reinforcement learn-

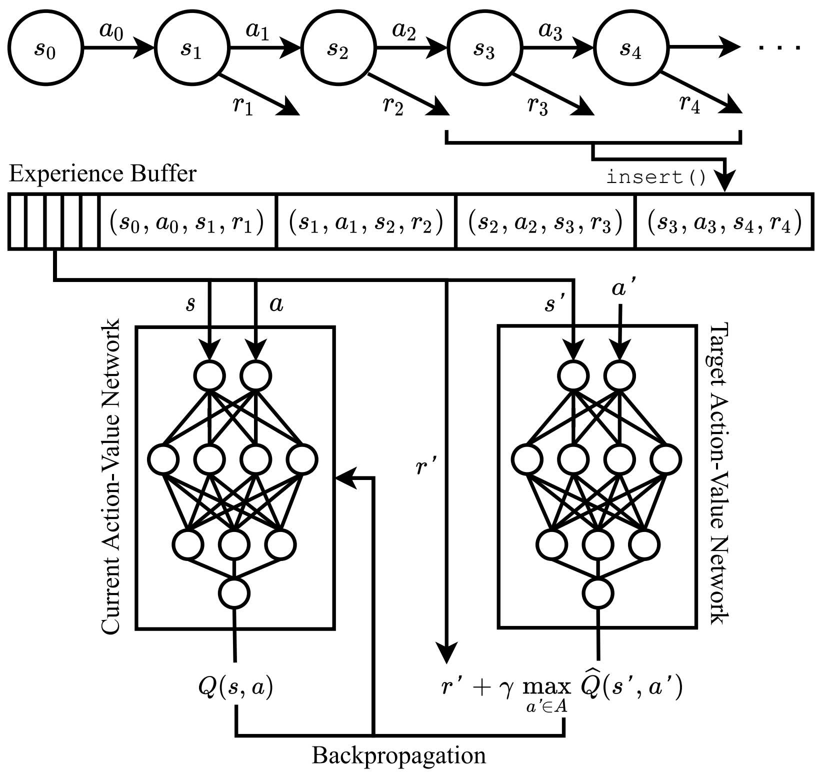


Figure 4.2: A diagram of our meta-level control technique.

ing agent can learn a policy expressed as a neural network by performing actions and observing rewards in its environment. This makes it a natural approach to the meta-level control problem for monitoring and controlling adjustable algorithms for three reasons [140]. First, by balancing exploitation with exploration, it can learn a policy that tunes the internal hyperparameters of the algorithm without knowing the transition function. Next, by ignoring large regions of the state space that are not reached in practice, it can reduce the overhead of learning a policy that tunes the internal hyperparameters of the algorithm. Finally, by using a neural network that can learn complex relationships between large input and output spaces, it can learn the effect of the internal hyperparameters on the internal state of the algorithm.

Algorithm 3 shows our metareasoning technique for monitoring and controlling adjustable algorithms using *deep Q-learning* [100]. Each episode (Line 4) starts by executing the adjustable algorithm for a time step on a generated instance of a specific problem (Lines 5-11). For each time step as the adjustable algorithm is executing (Line 12), there are several steps. First, the experience buffer is updated with the current state of computation, the current action of computation, the current reward, and the next state of computation (Lines 13-15) as shown in Figure 4.2. Next, if the size of the experience buffer exceeds the initialization period, we sample a minibatch (Lines 16-17). With that minibatch, the temporal-difference error is used to update the action-value network via backpropagation and then the target action-value network is updated via a moving average (Lines 17-21). Thereafter, the adjustable algorithm is either interrupted or executed for another time step while adjusting its internal hyperparameters by following the policy computed from the action-value network and the exploration strategy (Lines 22-28). Finally, the action-value function is returned (Line 29). We now walk through each line of Algorithm 3 below.

Reinforcement Learning Episode Loop The experience buffer is initialized to a capacity (Line 1). The current action-value function is initialized to an action-value network (Line 2). The target action-value function is initialized to the current action-value function (Line 3). An episode loop iterates from 1 to the number of episodes (Line 4). For each episode, the following phases are performed for setup (Lines 5-11), monitor (Lines 12-15), update (Lines 16-21), and control (Lines 22-28). The action-value function is returned (Line 29).

Episode Setup Phase An instance is sampled from the problem distribution and the adjustable algorithm is set up to solve that problem instance (Lines 5-6). The current time is initialized to zero, the current state is initialized to the quality and computation time of the initial solution along with any extra features, and the current action is initialized to the policy calculated from the current action-value function and the exploration strategy at the current state (Lines 7-9). The adjustable algorithm starts to solve the problem instance with the hyperparameters following the current action (Lines 10-11).

Algorithm 3: Our meta-level control technique that uses deep reinforcement learning, specifically deep Q-learning, to learn both optimal stopping and optimal hyperparameter tuning of an adjustable algorithm.

```
Input: An adjustable algorithm $\Lambda$, an action-value network $\mathcal{N}$, a step size $\alpha_1$, a
       target action-value network step size $\alpha_2$, an exploration strategy $\mathcal{E}$, an
       experience buffer capacity $\ell_1$, a number of episodes $\ell_2$, an initialization
       period $\ell_3$, a minibatch size $\ell_4$, and a duration $\Delta$
Output: An action-value function $Q$
1  $B \leftarrow \textsc{ExperienceBuffer}(\ell_1)$
2  $Q \leftarrow \textsc{NeuralNetwork}(\mathcal{N})$
3  $\hat{Q} \leftarrow Q$
4  for $i = 1, 2, \ldots, \ell_2$ do
5      $P \leftarrow \textsc{SampleProblemDistribution}()$
6      $\Lambda.\textsc{Setup}(P)$
7      $t \leftarrow 0$
8      $s_t \leftarrow (\Lambda.\textsc{Get}\Phi(), \Lambda.\textsc{Get}\Psi(), \Lambda.\textsc{Get}F())$
9      $a_t \leftarrow \pi_{\mathcal{E}}^{Q}(s_t)$
10     $\Lambda.\textsc{Start}(a_t.\Theta_1, \ldots, a_t.\Theta_{\ell_\Theta})$
11     $\textsc{Sleep}(\Delta)$
12     while $\Lambda.\textsc{Running}()$ do
13         $s_{t+1} \leftarrow (\Lambda.\textsc{Get}\Phi(), \Lambda.\textsc{Get}\Psi(), \Lambda.\textsc{Get}F())$
14         $r_t \leftarrow R(s_t, a_t, s_{t+1})$
15         $B.\textsc{Append}((s_t, a_t, r_t, s_{t+1}))$
16         if $B.\textsc{Size}() \geq \ell_3$ then
17             $M \leftarrow B.\textsc{SampleMinibatch}(\ell_4)$
18             $\hat{\mathcal{L}}(r, s') := r + \gamma \max_{a' \in A} \hat{Q}(s', a')$
19             $\mathcal{L}(s, a, r, s') := [\hat{\mathcal{L}}(r, s') - Q(s, a)]^2$
20             $Q.\textsc{Backpropagate}(M, \mathcal{L}, \alpha_1)$
21             $\hat{Q} \leftarrow (1 - \alpha_2) \cdot \hat{Q} + \alpha_2 \cdot Q$
22         $t \leftarrow t + 1$
23         $a_t \leftarrow \pi_{\mathcal{E}}^{Q}(s_t)$
24         if $a_t.\Theta_0 = \textsc{Stop}$ then
25             $\Lambda.\textsc{Stop}()$
26             break
27         $\Lambda.\textsc{Continue}(a_t.\Theta_1, \ldots, a_t.\Theta_{\ell_\Theta})$
28         $\textsc{Sleep}(\Delta)$
29     return $Q$
```

Episode Monitor Phase A loop runs until the adjustable algorithm is interrupted early or terminated naturally (Line 12). The successor state is set to the quality and computation time of the new solution and any extra features while the current reward is calculated from the current state, the current action, and the successor state (Lines 13-14). The experience buffer is appended with the current state, the current action, the current reward, and the successor state (Line 15).

Episode Update Phase The phase occurs if the size of the experience buffer is greater than the initialization period (Line 16). A minibatch is sampled from the experience buffer (Line 17). The loss function is defined as the square of the temporal-difference error (Lines 18-19). The current action-value function as a neural network is updated via backpropagation from the minibatch, the loss function, and the step size and the target action-value function is updated using the current action-value function (Lines 20-21).

Episode Control Phase The current time is incremented and the new current action is set to the policy calculated from the current action-value function and an exploration strategy at the new current state (Lines 22-23). If the action indicates to stop the adjustable algorithm, the adjustable algorithm is interrupted and the loop iterates to the next episode (Lines 24-26). Otherwise, the adjustable algorithm continues execution after tuning its internal hyperparameters (Lines 27-28).

4.5 Anytime Weighted A* Example

We turn to an application of our approach to anytime weighted A*. Recall that anytime weighted A* is an anytime heuristic search algorithm that uses an inadmissible heuristic function to find suboptimal solutions, continues searching after each suboptimal solution is found, exhibits an error bound on a suboptimal solution when it is interrupted, and guarantees an optimal solution once the open list is empty. Importantly, anytime weighted A* uses a weighted evaluation function $f_w(n) = g(n) + w \cdot h(n)$ instead of the standard evaluation function $f(n) = g(n) + h(n)$ with a path cost function $g(n)$ and a heuristic function $h(n)$ for a given node n. Meta-level control therefore involves not only optimal stopping and but also optimal hyperparameter tuning of the weight of anytime weighted A*.

Recent work on anytime weighted A* has focused on selecting the best static weight for a specific problem [58], choosing the best static weight for a specific instance of a problem [138], and adjusting the weight at runtime heuristically [153]. However, there is also work that shows that anytime weighted A* can be improved through random restarting when a solution is found [116] and even work that analyzes the failure conditions of anytime weighted A* with respect to its weight for specific problems [166]. Overall, recent work has shown that the problem of adjusting the weight of anytime weighted A* at runtime is challenging.

Intuitively, the meta-level control problem for monitoring and controlling anytime weighted A* is an MDP with two important attributes. The set of states reflects the quality and computation time of the current solution and extra features that summarize the nodes in the open list, the instance of the problem, and the performance of the system. The set of actions reflects an internal hyperparameter that either interrupts the algorithm or executes anytime weighted A* for another time step and an internal hyperparameter that adjusts the weight of anytime weighted A*. This MDP is used by our meta-level control technique learn how to monitor and control anytime weighted A* effectively using deep reinforcement learning.

More formally, the meta-level control problem for monitoring and controlling anytime weighted A*, Λ, is represented by an MDP $\langle \Phi, \Psi, F, S, A, T, R, s_0 \rangle$ given a time-dependent utility function $U : \Phi \times \Psi \rightarrow \mathbb{R}$. $\Phi = [0, 1]$ is the set of qualities. $\Psi = [0, \tau]$ is the set of time steps with a deadline τ. F is the set of features such that the feature $w \in W$ is the current weight, the features $\mu_g \in \mathbb{R}$ and $\mu_h \in \mathbb{R}$ are the mean of the

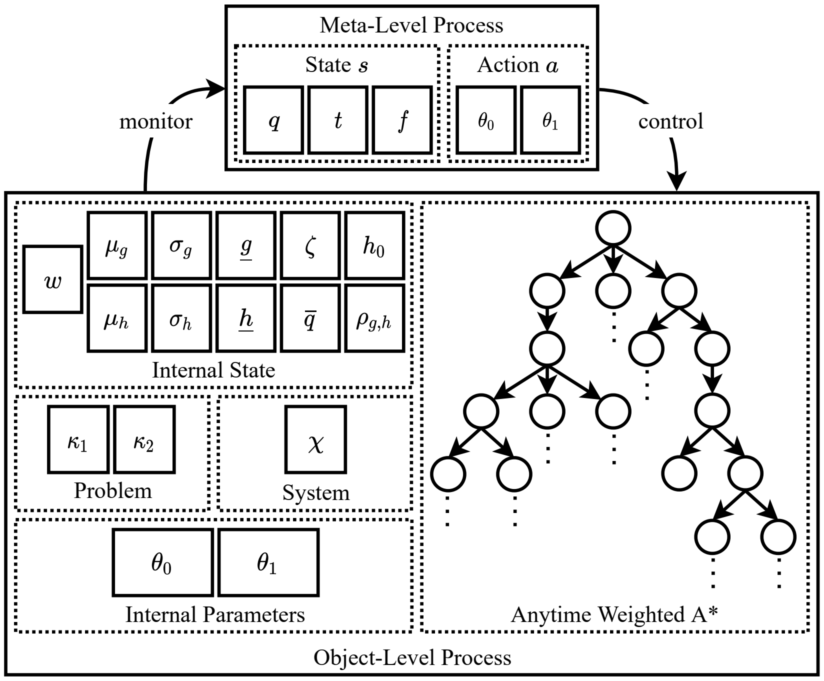


Figure 4.3: An example of a metareasoning architecture for anytime weighted A* that has a meta-level process and an object-level process.

g- and h-values on the open list, the features $\sigma_g \in \mathbb{R}$ and $\sigma_h \in \mathbb{R}$ are the standard deviation of the g- and h-values on the open list, the features $\underline{g} \in \mathbb{R}$ and $\underline{h} \in \mathbb{R}$ are the minimum g- and h-values on the open list, the feature $\zeta \in \mathbb{R}$ is the value $\log(n)$ for the number of nodes n on the open list, the feature $\bar{q} \in \mathbb{R}$ is the h-value of the initial state divided by the minimum f-value on the open list, the feature h_0 is the h-value of the initial state, the feature $\rho_{g,h} \in [-1, 1]$ is the correlation between the g- and h-values on the open list, the feature $\kappa \in K$ is a set of settings for the instance of the problem, and the feature $\chi \in [0, 1]$ is the processor usage of the system. A is the set of actions of computation: the internal hyperparameter $\Theta_0 \in \{\textsc{Stop}, \textsc{Continue}\}$ interrupts the algorithm or executes the algorithm for another time step while the internal hyperparameter $\Theta_1 = \{\leftarrow, \rightarrow\}$ adjusts the weight $w \in W$ by shifting its pointer to the set of weights $W = \{1, 1.5, 2, 3, 4, 5\}$ either left or right. Note that S, A, T, R, and s_0 follow from the meta-level control problem for adjustable algorithms.

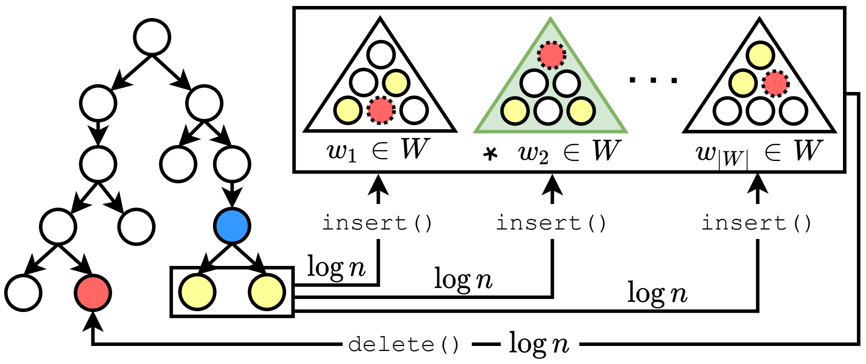


Figure 4.4: A modified implementation of anytime weighted A* that manages multiple open lists each associated with a specific weight.

The metareasoning architecture as shown in Figure 4.3 has a *meta-level process* that monitors and controls an *object-level process* that executes anytime weighted A*.

Anytime weighted A* involves a simple modification to allow its weight to be adjusted at runtime. Instead of inserting/deleting a node into/from a single open list for a static weight, the algorithm inserts/deletes this node into/from $|W|$ open lists each ordered by the f_w-value of a weight $w \in \mathcal{W}$ as illustrated in Figure 4.4 such that each open list has a different ordering of the same exact nodes. The worst-case time complexity for inserting/deleting a node across all open lists of size n sequentially is $\mathcal{O}(|W| \log n)$, which are two operations that can be parallelized for each open list.

4.6 Experiments

In this section, we evaluate our approach on a common benchmark domain that uses anytime weighted A* to solve a range of heuristic search problems and a mobile robot application that uses RRT* to solve motion planning problems.

In our experiments, our approach and each standard approach solve 1000 randomized instances of a problem using the adjustable algorithm from both domains. For each randomized instance, we record the final solution quality produced by the adjustable algorithm for all approaches. Ideally, for any instance of a problem, recall that we define solution quality as the approximation ratio, $q = c^*/c$, where c^*

is the cost of the optimal solution and c is the cost of the current solution. However, since computing the cost of the optimal solution for complex problems is often infeasible, we estimate solution quality as the approximation ratio, $q = \hat{c}^*/c$, where $\hat{c}^*$ is a problem-dependent lower bound on the cost of the optimal solution and c is the cost of the current solution following existing work on anytime algorithms [60]. Intuitively, a solution quality $q = 0$ means no solution was computed while a solution quality $q = 1$ means an optimal solution was computed.

The meta-level control problem for monitoring and controlling adjustable algorithms involves a time-dependent utility function. We consider a *contract* setting for our experiments: the algorithm must terminate before a deadline of τ sec to avoid a severe utility penalty Υ. This is common in planning and robotics where a system has a fixed duration for planning before execution. Formally, given a solution of quality $q \in \Phi$ at time step $t \in \Psi$, the time-dependent utility function is $U(q, t) = [t \leq \tau] \cdot U_I(q) - [t > \tau] \cdot \Upsilon$, where $U_I(q) = \iota q$ is the utility of a solution of quality $q \in \Phi$ for an intrinsic value multiplier ι. It is important to note that our approach can support any given well-behaved time-dependent utility function.

Our approach is trained on randomized instances of every problem for both domains. Each problem only involves a few hours of training. Algorithm 3 uses typical settings for deep Q-learning. The action-value network $\mathcal{N}$ is a fully connected neural network with two hidden layers of 64 and 32 nodes with ReLU activation and a linear output layer of 5 nodes. The step size α_1 is 0.0001. The target action-value network step size α_2 is 0.001. The exploration strategy $\mathcal{E}$ is ϵ-greedy action selection with an exploration probability ϵ that is annealed from 1 to 0.1 over 1000 episodes. The experience buffer capacity ℓ_1 is ∞. The number of episodes ℓ_2 is 15000 and 30000 for the anytime weighted A* and RRT* domains. The initialization period ℓ_3 is 1000. The minibatch size ℓ_4 is 64. The duration Δ is 1/20th of the contract. Our experiments were run on an AMD Ryzen 3900X processor with 32 GB of RAM.

Our open source Julia library, *Metareasoning.jl*, was used in our experiments: it offers our approach and the RL environments for anytime weighted A* and RRT*.

4.6.1 Common Benchmark Domain

We begin by considering the common benchmark domain that uses anytime weighted A* to solve a range of heuristic search problems. Here, we compare our approach to standard approaches that run anytime weighted A* with either a static weight, a dynamic weight that increases from the lowest weight after each solution, or a dynamic weight that decreases from the highest weight after each solution given a set of commonly used weights of 1, 1.5, 2, 3, 4, and 5.

We provide a brief description of each benchmark domain below. Each heuristic search problem is selected to reflect problems that require different static weights and problems for which counterintuitive behavior of anytime weighted A* was reported [166]. The parameters are selected to avoid trivializing the problem by either not having enough time so that no approach finds any solution or having too much time so that every approach finds the optimal solution within the contract. The duration τ corresponds to 6000, 6000, 3000, and 2400 node expansions for Sp, Isp, Tsp, and Cnp. We enforce a node expansion limit instead of a duration for reproducibility.

4.6.1.1 Sliding Puzzle

An Sp instance has $J = j^2 - 1$ tiles with each tile i labeled from 1 to J in a $j \times j$ grid. Every tile must be moved from an initial position to a desired position given a *unit* cost $c(i) = 1$ for moving a tile i. The sum of the Manhattan distances from the current position of each tile to its desired position is used as an admissible and consistent heuristic function h. The number of tiles J is 15. The difficulty of an instance, as measured by the h-value of the initial configuration of all tiles, is chosen randomly between 35 and 45. The meta-level control MDP has a set of settings K that represent the difficulty for the instance of the problem.

4.6.1.2 Inverse Sliding Puzzle

An Isp instance is the same as an Sp instance except that there is an *inverse* cost $c(i) = 1/i$ for moving a tile i. This means that the sum of the Manhattan distances from the current position of each tile to its desired position, weighted by the cost of moving each tile, is used as an admissible and consistent heuristic function h.

4.6.1.3 Traveling Salesman Problem

A Tsp instance has J cities that must be visited along an optimal route given a cost for each edge between a pair of cities. A percentage of the edges have an infinite cost to control its sparsity. The total cost of a minimum spanning tree across the unvisited cities with an infinite cost for no feasible tour is used as an admissible and consistent heuristic function h. The number of cities J is chosen randomly between 25 and 35. The percentage of edges with an infinite cost is chosen randomly between 0% and 30%. The cost for each edge between a pair of cities is chosen randomly. The meta-level control MDP has a set of settings K that represent the number of cities and the percentage of edges with an infinite cost for the instance of the problem.

4.6.1.4 City Navigation Problem

A Cnp instance simulates navigating between two locations in different cities [166]. There are J cities scattered randomly on a $j \times j$ square such that each city is connected by a random tour and to its nearest n_J cities. Each city contains I locations scattered randomly throughout the city that is an $i \times i$ square such that each location in the city is connected by a random tour and to its nearest n_I locations. The edge between a pair of cities costs the Euclidean distance plus an offset β_1. The edge between a pair of locations within a city costs the Euclidean distance scaled by a random number sampled between 1 and a maximum β_2. The goal is to find an optimal path from a randomly selected location in one city to a randomly selected location in another city. The Euclidean distance from the current location to the target location is used as an

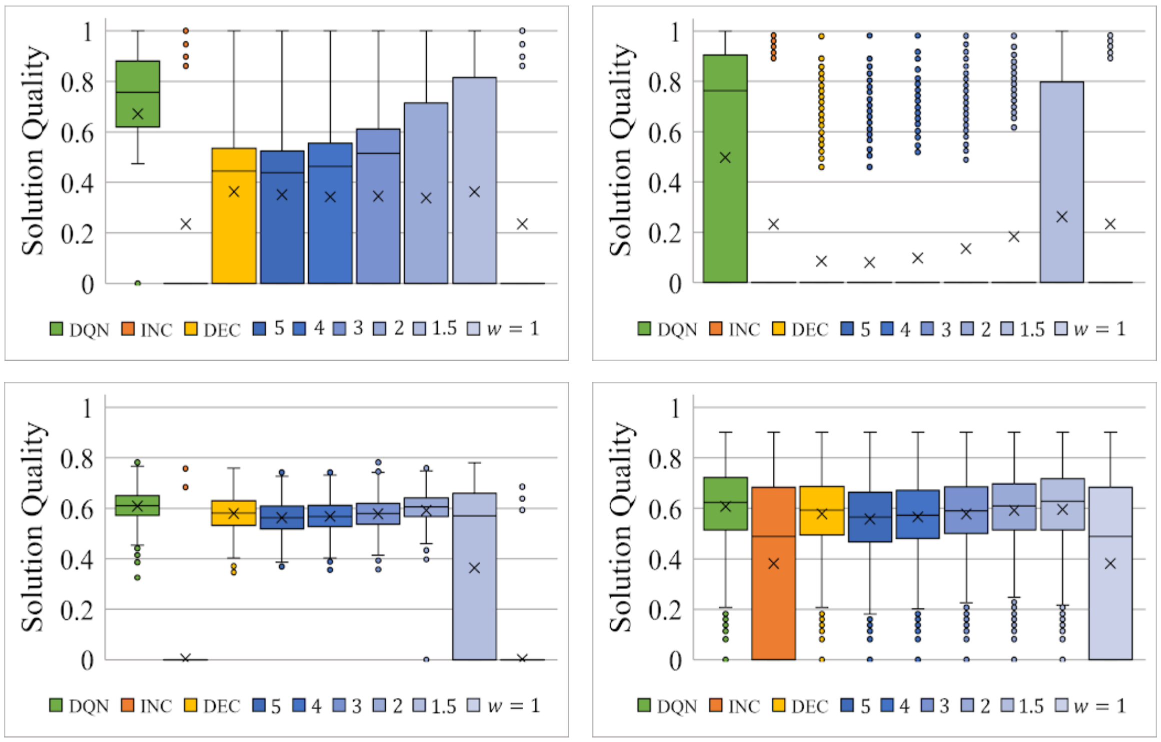


Figure 4.5: The box plots of the final solution qualities produced by anytime weighted A* for each approach over all instances of the Sp (*top-left*), Isp (*top-right*), Tsp (*bottom-left*), and Cnp (*bottom-right*) heuristic search problems.

admissible and consistent heuristic h. The parameters are chosen such that $J = 150$, $j = 100$, $n_J = 3$, $I = 150$, $i = 1$, $n_I = 3$, $\alpha = 2$, and $\beta = 1.1$. The meta-level control MDP does not have any additional set of settings K for the instance of the problem.

4.6.1.5 Discussion

Figure 4.5 shows the results for the common benchmark domain. Note that the *crosses* denote the mean, the *bullets* denote the outliers, and the *median* and *upper quartiles* can be zero for some approaches. Generally, our approach tends to solve more problem instances with higher solution quality than the best standard approach: it exhibits a better mean solution quality than *DEC* for Sp and $w = 1.5$ for Isp and also a comparable mean solution quality to $w = 2$ for Tsp and $w = 1.5$ for Cnp. Overall, our approach is better than or comparable to the baselines approaches without any need to tune the weight manually.

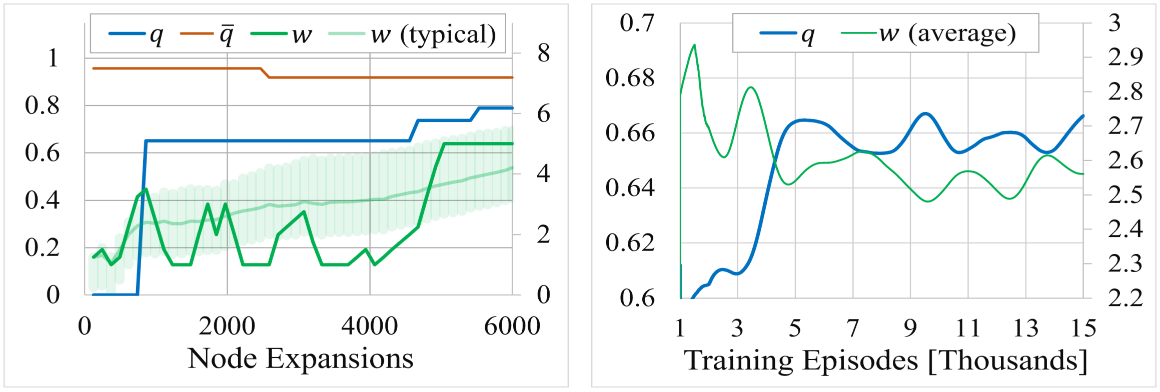


Figure 4.6: The first pair of analyses for the SP heuristic search problem.

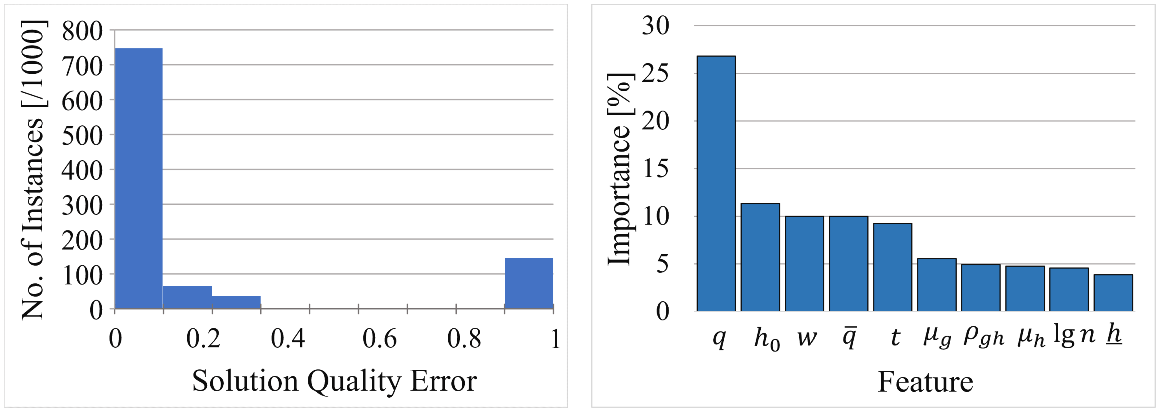


Figure 4.7: The second pair of analyses for the SP heuristic search problem.

Figure 4.6 and 4.7 offers a set of analyses of our approach on the SP heuristic search problem. Let us begin by examining the first pair of analyses in Figure 4.6. First, Figure 4.6(a) is a line chart that shows how the solution quality (*left y-axis*), the solution quality upper bound (*left y-axis*), and the weight (*right y-axis*) change with the node expansions for a select instance where the faded line for the typical weight in the shaded region represents the mean weight and its standard deviation over all instances (*right y-axis*). In this figure, our approach adjusts the weight based on solution quality and other features that are not shown. Generally, the mean weight increases as the node expansions increase to ensure generating at least one solution. Second, Figure 4.6(b) is a line chart that shows how the mean final solution quality (*left y-axis*) and the mean weight (*right y-axis*) change with the training episodes. In this figure, our approach improves its final solution quality with each training episode by learning gradually. In fact, the mean weight initially increases but then decreases to generate higher quality solutions.

Let us turn to examining the second pair of analyses in Figure 4.7. Figure 4.7(a) is a histogram that shows the distribution over all instances for the solution quality error of our approach where *solution quality error* is the normalized difference between the final solution quality of our approach and the final solution quality of the best approach. In this figure, our approach exhibits a solution quality error of 0 for over 700 instances. While roughly 100 instances have a solution quality error of 1, this is still better than the standard approaches. Figure 4.7(b) is a bar chart that shows the importance as a percentage of the top 10 features learned by our approach where *importance* is the mean absolute weight of a feature in the input layer of the neural network. In this figure, it is clear that our approach uses the other features, such as the current weight, the upper bound on solution quality, and the initial heuristic value in addition to solution quality and computation time.

4.6.2 Mobile Robot Application

We now consider the mobile robot application that uses RRT* to solve motion planning problems. RRT* is a popular algorithm that computes an optimal motion plan from a start state to a goal state by rapidly expanding a tree via sampling the map randomly [77]. Typically, RRT* has two hyperparameters: a growth factor limiting how much the tree grows for each sample and an area of focus biasing where each sample is drawn. Recent work on RRT* focuses on heuristically tuning the growth factor and area of focus [156, 4, 80].

Here, we compare our approach that runs RRT* with an adjustable growth factor and an adjustable area of focus to a standard approach that runs RRT* with a small and large static growth factor and a uniform area of focus that spans the entire map.

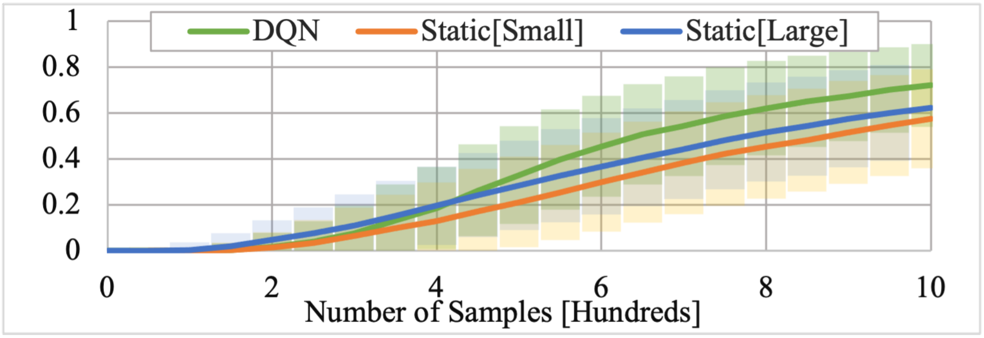


Figure 4.8: The performance of our approach and the standard approach to RRT* with a small and large growth factor over all instances of the motion planning problem.

Note that our approach can increase or decrease the growth factor between the small and large growth factors of the standard approach and move a small square as the area of focus either north, east, south, or west. Each motion planning problem is a map with a high density of obstacles where the start and goal states are in the bottom-left and top-right corners. The duration τ corresponds to 1000 samples. We enforce a sample limit instead of a duration for reproducibility.

At a high level, the meta-level control problem for monitoring and controlling RRT* is similar to anytime weighted A* but with a different set of features and actions of computation. The set of features includes the growth factor, the position of an area of focus, the percentage of samples that have expanded the tree, a lower bound on the remaining distance to the goal state, an estimate of the free space of the map, an estimated average size of the obstacles of the map, and a score for each possible area of focus that considers an estimated probability of improving the current path, the fraction of the current path within that area of focus, the fraction of the tree within that area of focus, and the average curvature of the current path within that area of focus. The actions of computation includes the internal hyperparameter that interrupts the algorithm or executes the algorithm for another time step, the internal hyperparameter that increases or decreases the growth factor, and the internal hyperparameter that moves the position of the area of focus either north, east,

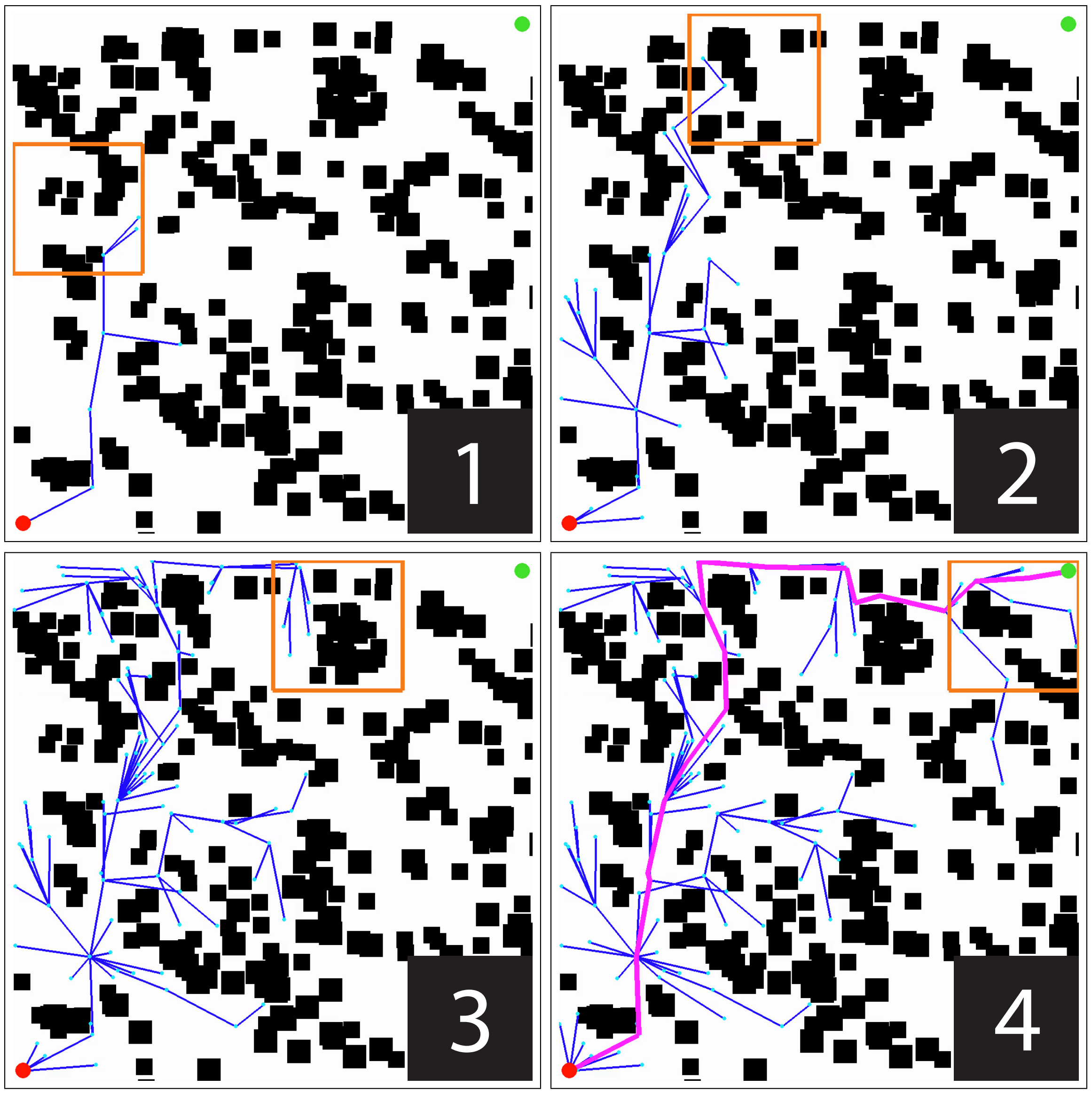


Figure 4.9: The evolution of RRT* over the number of samples for our approach on a select instance of the motion planning problem from Checkpoint 1 to 4.

south, or west. Note that the other attributes of the MDP follow directly from the meta-level control problem for adjustable algorithms.

4.6.2.1 Discussion

Figure 4.8 shows how the mean solution quality changes with the number of samples for each approach. In general, our approach outperforms the standard approach to RRT*: it produces a higher mean solution quality than the standard approach by adjusting the growth factor and the area of focus. In particular, our approach reaches a mean solution quality of roughly 0.72 while the standard approach to RRT* reaches a mean solution quality of 0.62 for the large growth factor and a mean solution quality of 0.59 for the small growth factor. It is important to highlight that our approach is comparable to or higher than the standard approach to RRT* over any number of samples, which indicates that it is sampling in a more efficient way that results in better motion plans. Given that we did not modify any of the settings of our deep reinforcement learning architecture that were originally used for anytime weighted A*, this result is encouraging. It is likely that our results would be even stronger had we modified the setting of our deep reinforcement learning architecture.

Figure 4.9 illustrates how our approach adjusts RRT* in practice. Note that the *black* shapes are the obstacles, the *red* and *green* circles are the start and goal states, the *blue* lines are the current tree, the *cyan* circles are the samples, the *purple* line is the current path, and the *orange* box is the current area of focus. In general, our approach guides the tree from the start state to the goal state of the map by focusing on the frontier that has been less explored to compute an initial path quickly. In particular, for all checkpoints, we observe that the area of focus is always placed at the frontier of the existing tree of RRT*. Intuitively, our approach shifts the area of focus to the frontier in order to compute an initial motion plan as quickly as possible. Once our approach has computed an initial motion plan, it tends to favor the center of the map near this motion plan in order to improve it as quickly as possible. Overall, our approach learns how to shift the area of focus to specific areas of the map, which outperforms the standard approach that simply samples from the entire map, by not only reducing how long it takes to compute an initial motion plan but also reducing how long it takes to improve that initial motion plan.

4.7 Summary

This chapter introduces a metareasoning approach to hyperparameter tuning of adjustable algorithms. By using deep reinforcement learning, our approach not only boosts the performance of an adjustable algorithm by tuning its hyperparameters at runtime on a specific instance of a problem, but it also eliminates any need for manual hyperparameter exploration. In our experiments, we show that our approach boosts overall performance on a common benchmark domain that uses anytime weighted A* to solve a range of heuristic search problems and a mobile robot application that uses RRT* to solve motion planning problems.

CHAPTER 5
METAREASONING FOR EXCEPTION RECOVERY

5.1 Introduction

Shifting to metareasoning for execution, we present a metareasoning approach that enables autonomous systems to detect, identify, and handle exceptions during operation. Autonomous systems have been deployed across many applications, such as autonomous driving [167, 169, 107], space exploration [183, 165], search and rescue [31, 53, 111], and energy conservation in smart buildings [87]. At a high level, these autonomous systems use decision-making models that have inherent limitations. For example, a self-driving car may be designed to drive along a route but not capable of passing different obstacles that block its route. Hence, in order to guarantee reliable operation under normal conditions, some restricting assumptions must be satisfied. This simplifies the complexity of designing, developing, and evaluating methods for efficient planning and plan execution [43, 54]. However, as a result of relying on incomplete decision-making models, the autonomous system may encounter a wide range of unanticipated scenarios that cannot be resolved during normal operation.

A simple approach to ensuring that the necessary conditions of normal operation are satisfied is to place the entire responsibility on the researcher or practitioner deploying the autonomous system. However, although relying on human judgment can improve performance [173], it is desirable to limit human involvement when the conditions of normal operation are violated. In fact, most of this responsibility should ideally be delegated to the autonomous system itself. Therefore, in this chapter, we propose a metareasoning approach that can pause a primary decision-making

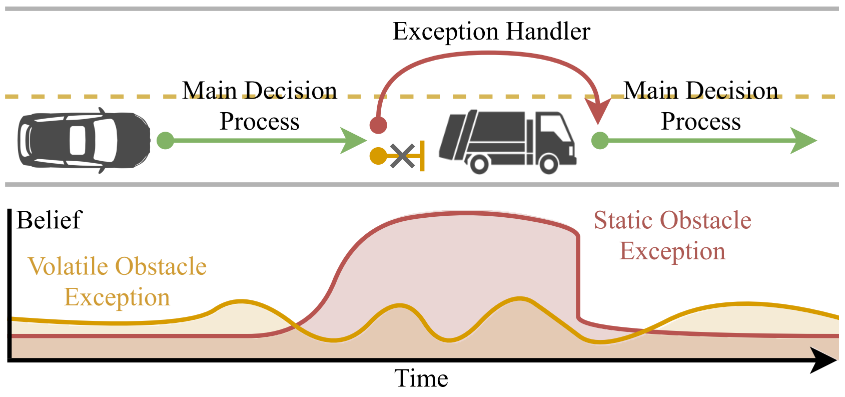


Figure 5.1: An example of an exception recovery metareasoning system.

model designed for normal operation and activate a secondary decision-making model designed to restore normal operation—with or without human involvement—given any violation of the necessary conditions of normal operation.

Despite tremendous progress in metareasoning centered on monitoring and controlling anytime algorithms [60], there have been few attempts to build autonomous systems that use metareasoning to recover from exceptional situations effectively. Such a system poses many challenges. First, because an unanticipated scenario is not captured by a decision-making model by definition, the model does not have the information needed to resolve that exception. Next, while a decision-making model can be extended to capture a set of unanticipated scenarios, a naïve approach will exponentially grow the complexity of the model with the number of exceptions. This is often infeasible for complex exceptions in real world applications. Finally, since a decision-making model cannot capture every unanticipated scenario, there will always be exceptions that cannot be resolved properly. In short, building autonomous systems that recover from exceptional situations effectively can be challenging.

There has been recent work in exception recovery that has focused on fault diagnosis—particularly detecting and identifying faults—during normal operation. For instance, many approaches diagnose faults by exploiting methods that use particle filters [38, 158, 98] or multiple model estimation with neural networks [119, 49].

While these approaches can *detect* and *identify* exceptions reliably, they do not offer a comprehensive framework that can also *handle* exceptions without human assistance. In particular, after an exception has been detected and identified by the autonomous system, these approaches often cease operation to request human assistance. In this chapter, building on recent work in fault diagnosis, our goal is to propose an exception recovery framework that detects, identifies, and handles exceptions effectively.

Hence, we propose an approach for building *exception recovery metareasoning systems* that use belief space planning for exception recovery. In Figure 5.1, this approach makes decisions by interleaving two types of decision processes: a main decision process designed for normal operation and a set of exception handlers designed for exceptional operation. As the system completes its task, it activates its decision processes based on a belief over potential exceptions. If its belief suggests normal operation, it executes its main decision process. Otherwise, if its belief indicates exceptional operation, it suspends its main decision process and executes an exception handler. It can also gather information or, if necessary, transfer control to a human operator given any uncertainty in its belief. At a high level, exception recovery metareasoning systems interleave a central decision process with a set of exceptions handlers by using a belief over a set of potential exceptions.

5.1.1 Contributions

In this chapter, we make the following contributions: (1) a formal definition of an exception recovery metareasoning system and its key properties, (2) a framework for profiling decision processes, (3) an application of an exception recovery metareasoning system to an autonomous driving domain, and (4) a demonstration that our approach is effective in simulation and on a fully operational prototype.

5.2 Exception Recovery

Given the complexity of the real world, autonomous systems have traditionally relied on limited decision-making models that depend on a number of simplifying assumptions to support planning and execution [43]. However, due to these limitations, autonomous systems can encounter unanticipated scenarios that cannot be resolved effectively. For instance, an autonomous vehicle can encounter different types of obstacles along its route. Achieving the complete potential of autonomous systems therefore requires the capability to recover from exceptional situations [6].

5.2.1 Exception Recovery Metareasoning Systems

In order to recover from exceptions, an *exception recovery metareasoning system* maintains a *belief* over a set of *potential exceptions*. The system uses this belief to reason about how to interleave a set of *decision processes*. Naturally, the set of decision processes includes the *regular process*, which makes decisions using a model designed for a particular task. If the system believes that there is not an exception, suggesting *normal operation*, it executes the regular process. The set of decision processes also includes a set of *exception handlers*, which make decisions based on a model designed for a specific exception. If the system believes that there is an exception, indicating *exceptional operation*, it executes an exception handler. It is also possible for the system to transfer control to a human operator given any uncertainty over many different exceptions. As an example, in a self-driving car, the regular process could be for navigating a route while the exception handlers could be for handling different types of obstacles. In short, by using its belief over a set of potential exceptions, the system alternates between regular decision making and exception handling.

Figure 5.2 illustrates how an exception recovery metareasoning system partitions its belief space over the set of potential exceptions into regions that are associated with distinct decision processes. Intuitively, each region of the belief space indicates

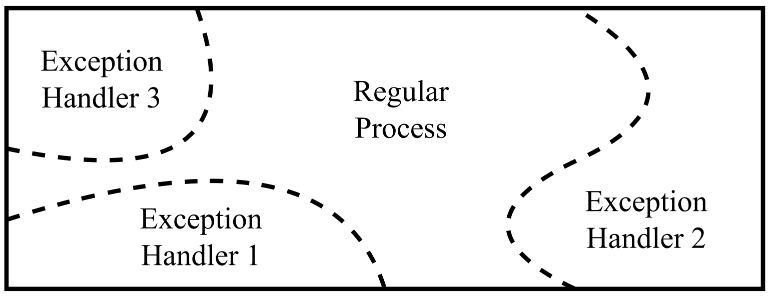


Figure 5.2: The space of beliefs of an exception recovery metareasoning system over the set of potential exceptions where each region is linked to a decision process.

whether or not the assumptions of the regular process have been violated. In general, the system executes whichever decision process is associated with the region that contains its current belief. In this diagram, the regular process corresponds to the largest region of the belief space. This region denotes normal operation because each belief indicates that the system has not encountered an exception. Thus, when its belief lies within this region, the system executes the regular process. Similarly, each exception handler correspond to a smaller region of the belief space. These regions mark exceptional operation since each belief suggests that the system has encountered an exception. Hence, when its belief lies within one of these regions, the system executes an exception handler instead. In short, the system simply executes whichever decision process is associated with the region containing its current belief.

The execution of an exception recovery metareasoning system can be viewed as a two-level hierarchy of decision processes: the high-level decision process is the exception recovery metareasoning system while the low-level decision process is the regular process or an exception handler. When the exception recovery metareasoning system executes the regular process or an exception handler, which can be viewed as a form of an option [141, 13], it transfers control to that decision process until a completion condition has been satisfied. The completion condition of the decision process depends on whether it is the regular process or an exception handler: the system executes the regular process *for a fixed duration* and an exception handler *until termination.* Once its completion condition has been met, the decision process generates an indicator that describes the status of its operation. Finally, after observing the indicator generated by the decision process, the exception recovery metareasoning system resumes execution once again. This repeats until the exception recovery metareasoning system has been terminated given the termination of the regular process.

Every decision process generates an *indicator* that describes its status after execution. An exception recovery metareasoning system uses each indicator to update its belief over potential exceptions. The information offered by the indicator, however, depends on the decision process. Because the objective of the regular process is to complete a specific task, it can generate a success or a failure signal (e.g., the route has been or cannot be completed) or a signal that suggests whether or not an exception has been encountered (e.g., an obstacle has been encountered). However, since the goal of an exception handler is to resolve a particular exception, it can generate a success signal (e.g., the obstacle is no longer blocking) or a signal that suggests different modes of failure (e.g., the obstacle is still blocking). Note that it is also possible for a decision process to generate other indicators, such as an abort signal if its execution is deemed no longer necessary.

An exception recovery metareasoning system always has a default exception handler, called the *human assistance exception handler*, that is assumed to handle any exception that is not yet linked to an exception handler. In particular, if there is no exception handler designed for a specific exception, it will execute the human assistance exception handler as a general form of exception handling. For example, if a self-driving car is blocked by an unrecognized obstacle, it will transfer control to the driver rather than use an obstacle handler. Thus, as new exception handlers are added to the system, its reliance on human assistance will diminish appropriately.

An exception recovery metareasoning system has standard attributes along with exceptions, decision processes, and indicators. That is, the system has *standard states*, *standard actions*, and *standard observations*. For instance along with its corresponding standard observations, an autonomous vehicle could have standard states for wait time and standard actions for waiting and edging in order to support information gathering during normal and exceptional operation. The system has a *standard transition function*, *standard reward function*, and *standard observation function* as well.

Since an exception recovery metareasoning system can naturally be represented as a belief space planning problem, we offer a formal description of an exception recovery metareasoning system by representing it as a POMDP below.

Definition 14. *An* ***exception recovery metareasoning system*** *can be described as a POMDP* $\langle E, P, I, \boldsymbol{S}, \boldsymbol{A}, \boldsymbol{T}, \boldsymbol{R}, \boldsymbol{\Omega}, \boldsymbol{O} \rangle$*, where*

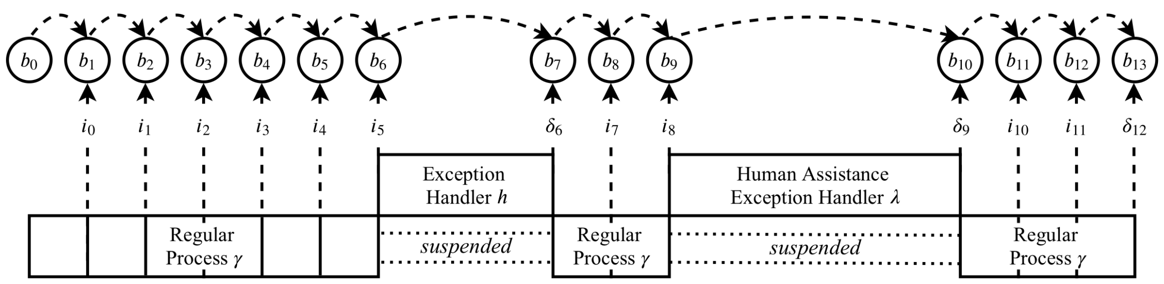
- E *is a set of exceptions that can be encountered (denoted as* e_i*),*
- P *is a set of decision processes that can be executed (denoted as* p_j*),*
- I *is a set of indicators that can be generated (denoted as* i_k*),*
- $\boldsymbol{S} = S \times E$ *is a set of factored states that is cross product of a set of standard states* S *and a set of exceptions* E*,*
- $\boldsymbol{A} = A \cup P$ *is a set of actions that is a union of a set of standard actions* A *and a set of decision processes* P*,*
- $\boldsymbol{T} : \boldsymbol{S} \times \boldsymbol{A} \times \boldsymbol{S} \rightarrow [0, 1]$ *is a transition function that is composed of a standard transition function* $T : S \times A \times S \rightarrow [0, 1]$*, a transition profile* $\tau_p : \boldsymbol{S} \rightarrow \triangle^{|S|}$*, and an exception profile* $\xi_p : \boldsymbol{S} \rightarrow \triangle^{|E|}$*,*
- $\boldsymbol{R} : \boldsymbol{S} \times \boldsymbol{A} \times \boldsymbol{S} \rightarrow \mathbb{R}$ *is a reward function that is composed of a standard reward function* $R : S \times A \times S \rightarrow [0, 1]$ *and a cost profile* $\zeta_p : \boldsymbol{S} \rightarrow \mathbb{R}$*,*
- $\boldsymbol{\Omega} = \Omega \cup I$ *is a set of observations that is a union of a set of standard observations* Ω *and a set of indicators* I*, and*

- $\boldsymbol{O} : \boldsymbol{S} \times \boldsymbol{A} \times \boldsymbol{\Omega} \rightarrow [0, 1]$ *is an observation function that is composed of a standard observation function* $O : S \times A \times \Omega \rightarrow [0, 1]$ *and an indicator profile* $\iota_p : \boldsymbol{S} \rightarrow \triangle^{|I|}$.

At a minimum, the first three attributes of an exception recovery metareasoning system contain several elements. The set of exceptions E requires normal operation η. The set of decision processes P requires the regular process γ and the human assistance exception handler λ. The set of indicators I requires a success signal σ and a failure signal ϕ. Note that the automated exception handler set, without the regular process or the human assistance exception handler, is denoted as H.

There are several principles that should be followed when building an exception handler. First, in order to cover as many exceptions as possible, an exception handler should be general rather than narrowly specialized. For instance, a self-driving car should have exception handlers for broad classes of obstacles that exhibit similar properties and behavior. Next, during the handling of an exception, an exception handler should meet the requirements of the regular process to prevent the exception handler from impacting the regular progress. Finally, by monitoring the conditions for which it has been activated, an exception handler should terminate itself if it determines that its execution is no longer necessary.

Following recent work on metareasoning for anytime algorithms [150, 149, 23], it is natural to view an exception recovery metareasoning system as a meta-level controller that *monitors* and *controls* the regular process at fixed intervals. In particular, the system *monitors* the regular process by maintaining a belief over whether or not the assumptions of normal operation have been violated and *controls* the regular process by executing it or suspending it to execute an exception handler. As a meta-level controller, the systems weighs the likelihood of normal and exceptional operation with the cost of executing the regular process or an exception handler.



b_0 b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8 b_9 b_{10} b_{11} b_{12} b_{13}
i_0 i_1 i_2 i_3 i_4 i_5 δ_6 i_7 i_8 δ_9 i_{10} i_{11} δ_{12}
Exception Handler h
Human Assistance Exception Handler λ
Regular Process γ
suspended
Regular Process γ
suspended
Regular Process γ

Figure 5.3: An exception recovery metareasoning system that interleaves the regular process with exception handlers based on its belief over possible exceptions.

Figure 5.3 offers an intuitive illustration of an exception recovery metareasoning system. Generally, in order to complete a particular task, the system runs different decision processes: it either executes the regular process for a fixed duration or an exception handler until termination. Once a decision process—that is, either the regular process or an exception handler—has satisfied its completion condition, it generates an indicator that can be used to update the belief of the system: the regular process emits an indicator after a fixed duration while an exception handler emits an indicator after termination. In this diagram, the system executes the regular process that emits indicator that suggest whether or not an exception has been encountered. However, in between several executions of the regular process, the system executes an exception handler that succeeds (emitting a success signal) and the human assistance exception handler that succeeds (emitting a success signal). During the execution of an exception handler, the regular process can be viewed as being *suspended* or *paused.* Once the regular process has reached a goal state or a dead end state, it terminates and emits a success or failure indicator that terminates the system.

5.2.2 Decision Process Profiles

Although a decision process can make decisions using a sophisticated decision-making model, an exception recovery metareasoning system does not rely on the internal mechanisms of a decision process. A decision process is instead summarized by a set of *profiles.* Intuitively, each profile forms as an abstraction over some feature of the internal mechanisms of the decision process within the system: that is to say, each profile describes a different dimension of the decision process within the context of the system. These profiles are used to compose the transition, reward, and observation functions in the definition of an exception recovery metareasoning system. We define each decision process profile below.

The first decision process profile indicates how a decision process transitions through the *standard state space* of the system. This can be expressed as a function that maps a *factored state* to a probability distribution over all *standard states.*

Definition 15. *A* ***transition profile****,* $\tau_p : \boldsymbol{S} \to \triangle^{|S|}$*, gives the probability of ending up in state* $s' \in S$ *after executing the decision process* $p \in P$ *in state* $\boldsymbol{s} \in \boldsymbol{S}$*.*

The second decision process profile captures how a decision process transitions through the *exception space* of the system. This can be viewed as a function that maps a *factored state* to a probability distribution over all *exceptions.*

Definition 16. *An* ***exception profile****,* $\xi_p : \boldsymbol{S} \to \triangle^{|E|}$*, gives the probability of ending up with exception* $e' \in E$ *after executing the decision process* $p \in P$ *in state* $\boldsymbol{s} \in \boldsymbol{S}$*.*

The third decision process profile encapsulates the *cost* of the system executing a decision process. This can be characterized as a function that maps a *factored state* to an *expected immediate cost* of a decision process as follows.

Definition 17. *A* ***cost profile****,* $\zeta_p : \boldsymbol{S} \to \mathbb{R}$*, gives the expected cost of executing the decision process* $p \in P$ *in state* $\boldsymbol{s} \in \boldsymbol{S}$*.*

The fourth decision process profile characterizes how a decision process emits an *indicator* to the system. This can be specified as a function that maps a *factored state* to a probability distribution over all *indicators* below.

Definition 18. *An* ***indicator profile****,* $\iota_p : \boldsymbol{S} \to \triangle^{|I|}$*, gives the probability of observing an indicator* $i \in I$ *after executing the decision process* $p \in P$ *and ending up in state* $\boldsymbol{s} \in \boldsymbol{S}$*.*

Finally, putting all of these profiles together, we present the complete description of a decision process as follows.

Definition 19. *A* ***decision process****, $p \in P$, can be described as a tuple of profiles $\langle \tau_p, \xi_p, \zeta_p, \iota_p \rangle$ that summarize its operation such that τ_p is the transition profile, ξ_p is the exception profile, ζ_p is the cost profile, and ι_p is the indicator profile.*

All decision processes use a policy to make decisions. A policy can be calculated in many ways. In most cases, it is possible to derive the policy from different decision-making models, such as an MDP, a POMDP, a decentralized POMDP, or an SSP. It is also possible for a domain expert to derive the policy by hand. Once the policy has been calculated, the transition, exception, and indicator profiles can be calculated either by hand or by sampling trajectories of when each decision process is executed by the exception recovery metareasoning system [167].

Figure 5.4 illustrates the transition of an exception recovery metareasoning system during the execution of a decision process. Intuitively, while its internal mechanisms may be sophisticated, a decision process is simply an action that is available to the system: when the system executes a decision process in a particular state, it transitions through states in its state space. In this illustration, the exception recovery metareasoning system executes a decision process p starting in state $\boldsymbol{s} = (s, e)$ and ending in state $\boldsymbol{s}' = (s', e')$. Once the system initiates the decision process in state $\boldsymbol{s} = (s, e)$, it transfers control to that decision process. The decision process then transitions through the states in its own state space by performing actions in its own action space starting from its start state (the *first doubled* node) and ending in its goal state (the *second doubled* node). After the decision process has been terminated, it transfers control back to the system in state $\boldsymbol{s}' = (s', e')$.

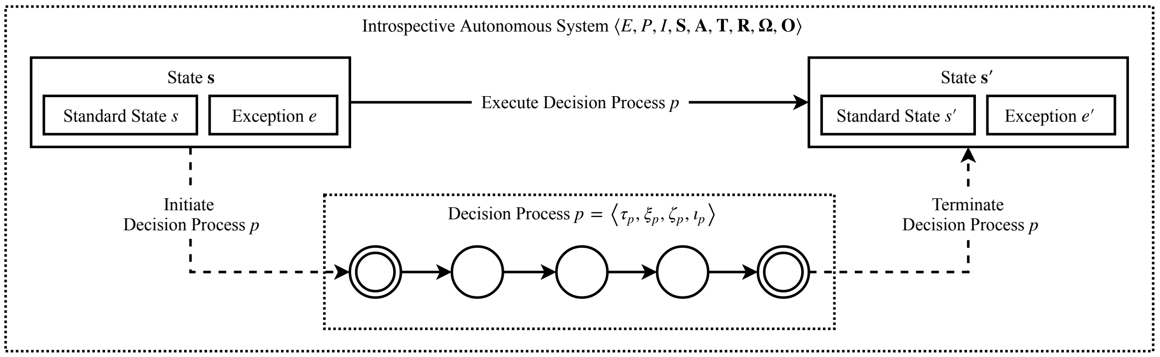

Introspective Autonomous System ⟨E, P, I, S, A, T, R, Ω, O⟩
State s
Standard State s
Exception e
Execute Decision Process p
State s′
Standard State s′
Exception e′
Initiate Decision Process p
Decision Process p = ⟨τp, ξp, ζp, ιp⟩
Terminate Decision Process p

Figure 5.4: The transition of an exception recovery metareasoning system in its state space during the execution of a decision process.

5.2.3 Dynamics

Now, by using the formal definition of a decision process, we can express the transition, reward, and observation functions of an exception recovery metareasoning system. For the transition function and the reward function, if the action is a decision process, the relevant profiles are used. Otherwise, the relevant standard function is used. Given a state $\boldsymbol{s} = (s, e) \in \boldsymbol{S}$, an action $\boldsymbol{a} \in \boldsymbol{A}$, and a successor state $\boldsymbol{s}' = (s', e') \in \boldsymbol{S}$, we describe the transition function and the reward function below.

$$\boldsymbol{T}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') = \begin{cases} \tau_{\boldsymbol{a}}(\boldsymbol{s}, s')\xi_{\boldsymbol{a}}(\boldsymbol{s}, e') & \textit{if } \boldsymbol{a} \in P \\ T(s, \boldsymbol{a}, s') & \textit{otherwise} \end{cases}$$

$$\boldsymbol{R}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') = \begin{cases} -\zeta_{\boldsymbol{a}}(\boldsymbol{s}) & \textit{if } \boldsymbol{a} \in P \\ R(s, \boldsymbol{a}, s') & \textit{otherwise} \end{cases}$$

For the observation function, if the action is a decision process and the observation is an indicator, the relevant profile is used. However, if the action is a standard action and the observation is a standard observation, the relevant standard function is used. Otherwise, the probability is nil. Given a successor state $\boldsymbol{s}' = (s', e') \in \boldsymbol{S}$, an action $\boldsymbol{a} \in \boldsymbol{A}$, and an observation $\boldsymbol{\omega} \in \boldsymbol{\Omega}$, we express the observation function as follows.

$$\boldsymbol{O}(\boldsymbol{s}', \boldsymbol{a}, \boldsymbol{\omega}) = \begin{cases} \iota_{\boldsymbol{a}}(\boldsymbol{s}', \boldsymbol{\omega}) & \textit{if } \boldsymbol{a} \in P \textit{ and } \boldsymbol{\omega} \in I \\ O(s', \boldsymbol{a}, \boldsymbol{\omega}) & \textit{if } \boldsymbol{a} \in A \textit{ and } \boldsymbol{\omega} \in \Omega \\ 0 & \textit{otherwise} \end{cases}$$

5.2.4 Robustness

An exception recovery metareasoning system enables the regular process to complete a task by handling exceptions that can be encountered during operation with a set of exception handlers. These exception handlers are critical to the effectiveness of the system. We therefore define the main properties of an exception handler below.

Definition 20. *An exception handler, $h \in H$, is* ***strong*** *if it is guaranteed to handle a specific exception $e \in E$ for all states $\boldsymbol{s} \in \boldsymbol{S}$.*

Definition 21. *An exception handler, $h \in H$, is* ***conditionally strong*** *if it is guaranteed to handle a specific exception $e \in E$ for some states $\boldsymbol{s} \in \boldsymbol{S}$.*

Definition 22. *An exception handler, $h \in H$, is* ***weak*** *if it is not strong or conditionally strong.*

Recall that the exception profile of a decision process describes its level of effectiveness in handling a particular exception. Intuitively, for a strong exception handler (Definition 20) and a conditionally strong exception handler (Definition 21), the exception profile of the decision process must indicate that a particular exception is handled with certainty for either all states or some states respectively. Otherwise, the exception handler is considered to be weak (Definition 22).

Finally, given all of these properties, we define the central property of an exception recovery metareasoning system in the following way.

Definition 23. *An exception recovery metareasoning system is* ***robust*** *if there exists a strong or conditionally strong exception handler, $h \in H$, that is guaranteed to handle any exception $e \in E$ that may arise in any state $\boldsymbol{s} \in \boldsymbol{S}$ during operation.*

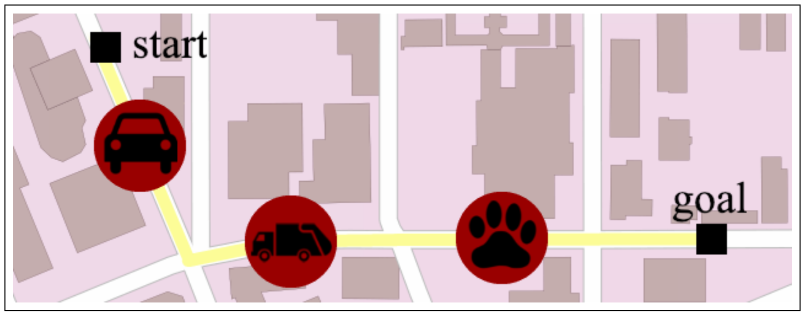


Figure 5.5: An example route with several obstacles.

5.3 Autonomous Driving Domain

In this section, we provide an application of exception recovery metareasoning systems to an autonomous driving domain. In this domain, an autonomous vehicle must drive along a route from a start location to a goal location. However, as the autonomous vehicle progresses along this route, it can encounter different types of obstacles of increasing volatility that must be resolved:

- a *static obstacle* that remains stopped permanently,
- a *dynamic obstacle* that stops and goes repeatedly, and
- an *erratic obstacle* that behaves unpredictably.

It is possible to extend the autonomous driving domain to include different obstacles, such as a pedestrian, a parked car, a garbage truck, a road block, a car that is parallel parking, a bicycle, a construction zone, or an obstructed traffic light [106, 107].

Figure 5.5 shows an example route that has a static obstacle (the *parked car* icon), a dynamic obstacle (the *garbage truck* icon), and an erratic obstacle (the *bear paw* icon) that must be passed by the autonomous vehicle. As discussed earlier, recall that an exception recovery metareasoning system requires a set of decision processes that includes the regular process and the set of exception handlers. We describe the regular process, the set of exception handlers, and the exception recovery metareasoning system of the autonomous driving domain below.

5.3.1 Navigation Problem

First, we consider the regular process of the system. The decision-making model of the regular process is a navigation problem where the autonomous vehicle must drive along a route from a start location to a goal location. In particular, for the regular process $\gamma \in P$, the **navigation problem** can be represented by the tuple $\langle S^\gamma, A^\gamma, T^\gamma, C^\gamma, s_0^\gamma, s_g^\gamma \rangle$, where

- S^γ is a set of states that represent intersections,
- A^γ is a set of actions that represent road segments,
- $T^\gamma : S^\gamma \times A^\gamma \times S^\gamma \rightarrow [0, 1]$ is a transition function that represents whether or not an intersection $s \in S^\gamma$ is connected to an intersection $s' \in S^\gamma$ by a road segment $a \in A^\gamma$,
- $C^\gamma : S^\gamma \times A^\gamma \times S^\gamma \rightarrow \mathbb{R}^+$ is a cost function that represents the length of a road segment $a \in A^\gamma$ that connects an intersection $s \in S^\gamma$ to an intersection $s' \in S^\gamma$,
- s_0^γ is a start intersection, and
- s_g^γ is a goal intersection.

The navigation problem assumes that all road segments do not have any obstacles at its level of abstraction. Given this limitation, when the autonomous vehicle is driving from one intersection to another intersection through some road segment, the navigation problem makes the assumption that the road segment will be traversed successfully. In the real world, however, there may be a number of obstacles that must be handled by the autonomous vehicle on any road segment.

5.3.2 Obstacle Handling Problem

Next, we consider each exception handler of the system. At minimum, the exception handlers include a human assistance obstacle handler λ that is guaranteed to resolve any obstacle with a high penalty. More importantly, the exception handlers also include an automated obstacle handler designed for each obstacle. Each decision-making model is based on a version of the obstacle handling problem. That is, for each automated obstacle handler, $h \in H$, the **obstacle handling problem** can be expressed as the tuple $\langle S^h, A^h, T^h, C^h, s_0^h, s_g^h \rangle$, where

- $S^h = S_p^h \times S_l^h \times S_r^h \times S_b^h$ is a set of factored states such that S_p^h describes the position of the autonomous vehicle (obstructed/passing/passing with caution/collision/unobstructed), S_l^h describes whether or not the left lane is available (open/closed), S_r^h describes whether or not the right lane is available (open/closed), and S_b^h describes whether or not the obstacle is blocking (blocking/not blocking),

- $A^h = \{\textsc{Stop}, \textsc{Edge}, \textsc{Go}, \textsc{Pass}, \textsc{PassWithCaution}\}$ is a set of actions that represents the maneuvers of the autonomous vehicle,

- $T^h : S^h \times A^h \times S^h \rightarrow [0, 1]$ is a transition function that multiplies the probabilities of a range of scenarios including the probability $\Pr(s_l' | s_l)$ that the availability of the left lane changes, the probability $\Pr(s_r' | s_r)$ that the availability of the right lane changes, and the probability $\Pr(s_b' | s_b)$ of whether or not the obstacle is blocking changes,

- $C^h : S^h \times A^h \times S^h \rightarrow \mathbb{R}^+$ is a cost function with unit cost for every state other than a goal state,

- s_0^h is a start state with an obstructed position, and

- s_g^h is a goal state with an unobstructed position.

Note that any state with a collision or infinite waiting is an absorbing dead end state with unit cost. This may occur in two situations: the $\textsc{Go}$ action is executed when the state factor s_b is blocking and the Pass or PassWithCaution action is executed when the state factor is closed s_l and the state factor s_r is closed.

All obstacles handlers are based on the obstacle handling problem. In particular, the transition function of the obstacle handling problem has been modified to follow the expected behavior of each type of obstacle that can be encountered by the autonomous vehicle. This involves adjusting the probability $\Pr(s_b' | s_b)$ of whether or not the obstacle is blocking changes. First, for the *static obstacle handler*, this probability will be low because a static obstacle has a low likelihood of changing its position. As a result, the policy of the static obstacle handler indicates to pass the obstacle immediately (the Pass action). Next, for the *dynamic obstacle handler*, this probability will be moderate since a dynamic obstacle has a medium likelihood of changing its position. Thus, the policy of the dynamic obstacle handler indicates to pass the obstacle cautiously (the PassWithCaution action). Finally, for the *erratic obstacle handler*, this probability will be high provided that an erratic obstacle handler has a high likelihood of changing its position. Hence, the policy of the erratic obstacle handler indicates to stop and wait for the obstacle to move (the Stop action).

5.3.3 Exception Recovery Metareasoning Vehicle

Given the regular process and each exception handler, we consider the exception recovery metareasoning system. The **exception recovery metareasoning vehicle**, v, can be described as an exception recovery metareasoning system represented as a tuple $\langle E^v, P^v, I^v, \boldsymbol{S}^v, \boldsymbol{A}^v, \boldsymbol{T}^v, \boldsymbol{R}^v, \boldsymbol{\Omega}^v, \boldsymbol{O}^v \rangle$, where

- $E^v = \{\eta, e_1, e_2, e_3\}$ is a set of exceptions such that η is no obstacle, and e_1, e_2, and e_3 is the presence of a static, dynamic, and erratic obstacle respectively,
- $P^v = \{\gamma, \lambda, h_1, h_2, h_3\}$ is a set of decision processes such that γ is the regular process, λ is the human assistance obstacle handler, and h_1, h_2, and h_3 is the static, dynamic, and erratic obstacle handler respectively,

- $I^v = \{\sigma, \phi, i_b, i_m\}$ is a set of indicators such that σ is the success signal, ϕ is the failure signal, and i_b and i_m are signals that indicate whether or not an obstacle is blocking and moving respectively,
- $\boldsymbol{S}^v = S^v \times E^v$ is a set of factored states: a standard state set S^v and the exception set E^v such that S^v is the wait time (none/short/medium/long),
- $\boldsymbol{A}^v = A^v \cup P^v$ is a set of actions: a standard action set $A^v = \{\text{Edge}, \text{Wait}\}$ and the decision process set P^v,
- $\boldsymbol{T}^v : \boldsymbol{S}^v \times \boldsymbol{A}^v \times \boldsymbol{S}^v \to [0, 1]$ is a transition function,
- $\boldsymbol{R}^v : \boldsymbol{S}^v \times \boldsymbol{A}^v \times \boldsymbol{S}^v \to \mathbb{R}$ is a reward function,
- $\boldsymbol{\Omega}^v = \Omega^v \cup I^v$ is a set of observations that is a union of a standard observation set Ω^v and the indicator set I^v, and
- $\boldsymbol{O}^v : \boldsymbol{S}^v \times \boldsymbol{A}^v \times \boldsymbol{\Omega}^v \to [0, 1]$ is an observation function.

Note that a monolithic POMDP that reasons about both the navigation problem and the obstacle handling problems together would rapidly become computationally intractable. Given the navigation problem with $|S^\gamma|$ states and n obstacle handling problems with $|S^h|$ states, such a POMDP requires $|S^\gamma| \cdot |S^h|^n$ states. For instance, if there are 30 intersections and 3 types of obstacles, there would be 1920000 states, which cannot be solved even with state-of-the-art POMDP solvers [110, 135].

5.3.4 Analysis

Our goal is to show that the exception recovery metareasoning vehicle can complete its route by handling all obstacles that can be detected and identified during navigation. This involves proving that all obstacle handlers are strong and the exception recovery metareasoning system is robust. First, for an obstacle handler to be strong, it must always handle a particular obstacle across all states of the system.

To do this, the obstacle handler problem must be guaranteed to reach the goal state, which indicates that the obstacle has been handled. We must therefore prove that the obstacle handling problem is an SSP, a class of decision-making models that ensures goal reachability [21, 82]. We show that every obstacle handler is strong below.

Proposition 1. *An obstacle handler, $h \in H^v \subset P^v$, is strong.*

Proof Sketch. At a high level, in order to show that an obstacle handler, $h \in H^v \subset P^v$, is strong, we prove that the obstacle handling problem is an SSP. The problem must satisfy two important conditions to be an SSP. First, there must exist a proper policy such that there is an action that can reach the goal state with unit probability for all states. Second, all improper policies must incur an infinite cost for all states from which it cannot reach the goal state with unit probability.

We begin with the first condition. There are two scenarios where the goal state cannot be reached due to a dead end state: either the Go action is selected when the state factor s_b is blocking or the Pass or PassWithCaution action is selected when the the state factors s_l and s_r are closed. A dead end state, however, can be avoided using the Stop action. Because the blocking probability $\Pr(s'_b|s_b)$ of the transition function is nonzero, the obstacle will eventually no longer be blocking and the goal state can be reached using the Go action. Thus, since there exists a proper policy, the first condition is met. We now prove the second condition. By definition, a dead end state is an absorbing state with nonzero cost. Hence, since all policies that can transition to a dead end state incur an infinite cost, the second condition is met.

Therefore, since the obstacle handling problem is an SSP because it satisfies both SSP conditions, it follows that the obstacle handler is strong. □

Finally, given that all obstacle handlers are strong, it is easy to show that the exception recovery metareasoning system is robust as follows.

Theorem 1. *An exception recovery metareasoning vehicle, v, is robust.*

Proof Sketch. To show that the exception recovery metareasoning vehicle, v, is robust, we prove there exists an obstacle handler that can always handle any obstacle. By Proposition 1, we know that all obstacle handlers are strong. Therefore, the exception recovery metareasoning vehicle is robust. □

5.4 Demonstration

In this section, we demonstrate that the exception recovery metareasoning vehicle is effective in simulation and on a fully operational prototype. In particular, we compare different versions of the exception recovery metareasoning vehicle to an autonomous vehicle that does not have exception recovery. Each exception recovery metareasoning vehicle can only execute a specific set of obstacle handlers. The set of obstacle handlers, $H^{v_i} \subset P^{v_i}$, that are available to each exception recovery metareasoning vehicle v_i is listed below:

- $H^{v_1} = \{\lambda\}$,
- $H^{v_2} = \{\lambda, h_1\}$,
- $H^{v_3} = \{\lambda, h_1, h_2\}$, and
- $H^{v_4} = \{\lambda, h_1, h_2, h_3\}$

The autonomous vehicle that does not have exception recovery cannot use any obstacle handlers. We refer to this autonomous vehicle as the regular autonomous vehicle.

In simulation, each experiment represents an instance of the navigation problem with different obstacle handling problems: the exception recovery metareasoning vehicle has to complete a route with static, dynamic, and erratic obstacles from a start location to a goal location. To do this, we run an *exception recovery metareasoning vehicle process.* This process uses a belief to interleave decision process during operation. When the current belief suggests that normal operation, the *navigation process*

Obstacle Handlers	Incidents	Autonomy (%)	Transfers	Time (s)
None	12	—	—	—
λ	0	51.4	12	750.2
λ, h_1	0	60.3	9	700.0
λ, h_1, h_2	0	72.0	6	649.8
λ, h_1, h_2, h_3	0	84.3	3	599.5

Table 5.1: The performance of all autonomous vehicles on exception recovery.

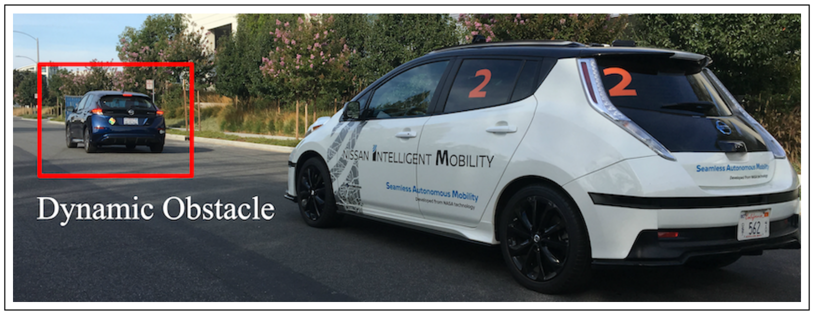


Figure 5.6: A fully operational exception recovery metareasoning vehicle prototype.

is executed. However, when the current belief suggests exceptional operation, an *obstacle handling process* is executed. The exception recovery metareasoning vehicle process finally terminates when the navigation process has been terminated.

All autonomous vehicles traverse a route with 3 instances of each type of obstacle that can be resolved by a particular obstacle handler and 3 instances of an unrecognized obstacle that can only be resolved by the human assistance obstacle handler for a total of 12 obstacles. Other routes can be constructed using the observation that the expected time required to handle each type of obstacle remains consistent: handling a static obstacle, a dynamic obstacle, an erratic obstacle, and an unrecognized obstacle requires 12.3, 15.1, 12.5, and 14.7 seconds respectively. Due to implementation constraints, transferring control to and from the driver safely requires roughly 8 seconds following recent work on the transfer of control problem [167, 2, 170].

Table 5.1 shows the performance of the regular autonomous vehicle and each version of the exception recovery metareasoning vehicle. First, the *Obstacle Handlers* column lists the obstacle handlers available to the vehicle. Second, the *Incidents* column includes the number of exceptions that prevent the vehicle from completing its route due to an exception that leads to a collision or infinite waiting. Third, the *Autonomy* column shows the percentage of time that the vehicle is driven autonomously. Fourth, the *Transfers* column includes the number of activations of the human assistance exception handler by the vehicle. Fifth, the *Time* column presents the time needed for the vehicle to complete its route in seconds. Note that the dashes denote that the vehicle cannot complete its route for the regular autonomous vehicle.

On a fully operational autonomous vehicle prototype, we demonstrate that the exception recovery metareasoning vehicle is effective on a route in the real world. The route included a static obstacle, a dynamic obstacle, and an erratic obstacle. The static obstacle was a parked vehicle, the dynamic obstacle was a slow-moving vehicle, and the erratic obstacle was an unpredictable pedestrian. The vehicle completed its route by resolving all obstacles. Figure 5.6 shows the fully operational autonomous vehicle prototype passing a dynamic obstacle during the demonstration.

5.5 Discussion

All experiments highlight the effectiveness of the exception recovery metareasoning vehicle in the real world. In general, as each obstacle handler is added to the exception recovery metareasoning vehicle in Table 5.1, the scope of autonomy increases while the duration of the route decreases without any potential incidents. Initially, when the vehicle cannot execute any obstacle handlers, it does not complete its route due to 12 potential incidents. Once the human assistance obstacle handler is added to the vehicle, the vehicle completes its route without any potential incidents since a human operator can handle all obstacles that can be encountered during navigation. However, because the vehicle cannot handle any obstacle without a human operator, it exhibits a low level of autonomy (51.4%) and route time (750.2 s), which includes 12 transfers. More importantly, as each obstacle handler is added to the vehicle, its level of autonomy increases without any potential incidents. Finally, when the vehicle can execute all obstacle handlers, its exhibits a high level of autonomy (84.3%) and route time (599.5 s), which only includes 3 transfers. The remaining unknown obstacles cannot be handled by any obstacle handler: they must be handled by the human assistance obstacle handler instead. We emphasize that the duration of the route decreases because the overhead of transferring control to a human operator decreases as each obstacle handler is added to the system. In short, the exception recovery metareasoning vehicle becomes progressively more independent while decreasing the duration of its route without impacting the safety and reliability of operation.

Exception recovery metareasoning systems offer a number of advantages over traditional autonomous systems. First, because the system reasons over a space of beliefs, it can represent the uncertainty over whether or not the assumptions of the regular process have been violated. Next, since the system has a belief over the set of potential exceptions, it can perform actions that gather more information about whether or not an exception has been encountered. Moreover, without increasing the complexity of the regular process, each exception handler can exploit additional information that is not available to the regular process that may be necessary to handling a particular exception. Finally, the system offers a modular framework that can easily be extended with additional exception handlers without affecting the effectiveness of the regular process. Exception recovery metareasoning systems are therefore a natural approach to recovering from exceptions during operation.

5.6 Summary

This chapter introduces a metareasoning approach to exception recovery. An exception recovery metareasoning system interleaves a regular process with a set of exception handlers to detect, identify, and handle exceptions by using belief space planning. By reasoning over the assumptions of normal operation, our approach interleaves the regular process with different exception handlers to identify, detect, and handle exception in a scalable way. In our experiments, we show that an application of an exception recovery metareasoning system to autonomous driving is effective in simulation and on a fully operational prototype.

CHAPTER 6
METAREASONING FOR SAFETY

6.1 Introduction

Going beyond our work in the previous chapter, we propose a metareasoning approach that enables autonomous systems to maintain and restore safety during operation. While planning and robotics experts carefully design, build, and test the models used by autonomous systems for high-level decision making, it is often infeasible for these models to ensure safety across every scenario within the domain of operation [148]. This is due to the inherent challenge of specifying comprehensive decision-making models that results from the complexity of the state space or action space, a lack of information about the environment, or a misunderstanding of the limitations of the autonomous system [17]. For example, a courier robot could use a decision-making model with features for safely interacting with different types of doors but not for navigating a crosswalk, which increases the risk of endangering people, damaging property, or breaking the courier robot when navigating a crosswalk [18]. Therefore, as autonomous systems grow in independence and complexity [7], it is critical to give them the ability to maintain and restore safety during operation.

A naive approach to giving an autonomous system the ability to maintain and restore safety would be to use a comprehensive decision-making model with every feature needed to cover every scenario within the domain of operation. Such a model, however, would suffer from two main drawbacks in real world environments [148]. First, the model would simply be infeasible to design due to the intractability of complex environments. Second, even if it were feasible to design, the model would likely be infeasible to solve with exact or even approximate methods due to the urgency of real-time environments. Hence, in order to avoid the infeasibility of a monolithic model, this chapter offers a scalable framework for safe decision making in autonomous systems that decouples the system into a primary process with features that are necessary to achieving its main goal and secondary processes each with features that are necessary to responding to a particular hazard.

There are several areas that work toward safety in autonomous systems that have seen recent attention [7]. First, methods avoid *negative side effects* that cause a system to interfere with its environment (e.g., by adding an extra term to its objective function [124, 125] or modifying its decision-making model based on human feedback [174, 16]). Next, methods mitigate *reward hacking* that cause a system to game its reward function (e.g., by applying ethical constraints to its behavior [12, 130, 79, 144, 143, 145, 104] or treating its reward function as an observation of its true objective function [56, 55, 88]). Finally, methods handle *distributional change* that cause a system to perform poorly in a new environment that differs from its original environment (e.g., by detecting anomalies using Monte Carlo methods based on particle filters [38, 158, 98] or multiple model estimation based on neural networks [119, 49]). However, while these areas are critical to safety in autonomous systems, this chapter focuses on tweaking the operation of an autonomous system for safe decision making.

In particular, we propose a disciplined, decision-theoretic metareasoning approach to safe decision making in autonomous systems [142]. A *safety metareasoning system* executes in parallel a *task process* that completes a specified task and *safety processes* that each address a specified safety concern with a *conflict resolver* for arbitration. Like a standard autonomous system, the task process completes a specified task by performing an action in its current state following its policy. However, at fixed intervals as the task process performs each action, there are two extra operations that

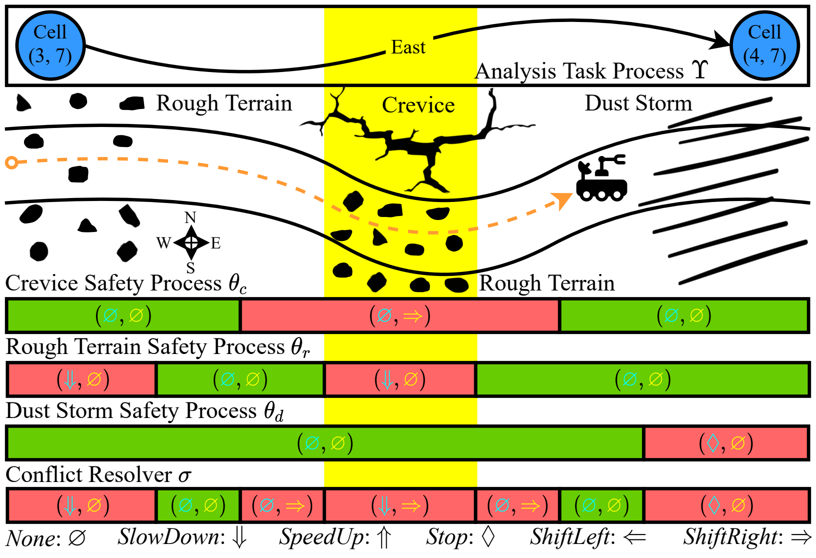


Figure 6.1: An illustration of a safety metareasoning system.

are not considered by a standard autonomous system. First, the safety processes each address a specified safety concern by recommending a rating over a set of parameters in its current state that can adjust the action being performed by the task process. Second, the conflict resolver for arbitration selects the optimal parameter that will adjust the action being performed by the task process given the ratings over the set of parameters recommended by the safety processes. Our experiments highlight that our approach optimizes the *severity* of safety concerns (the danger of particular hazards) and the *interference* to the task (the overhead of safety on the main goal).

Consider the planetary rover exploration domain that is illustrated in Figure 6.1. In this domain, a planetary rover executes a task process Υ that analyzes different points of interest within a region of a planet and safety processes θ_c, θ_d, and θ_r that address crevices, dust storms, and rough terrain with a conflict resolver σ for arbitration. Consider the highlighted time slice that shows the planetary rover completing the analysis task while addressing crevices, dust storms, and rough terrain. Intuitively, (1) the task process performs the East action starting in the cell $(3, 7)$ and ending in the cell $(4, 7)$, (2) the safety processes θ_c, θ_d, and θ_r recommend the parameters $(\varnothing, \Rightarrow)$, $(\varnothing, \varnothing)$, and $(\Downarrow, \varnothing)$, that can adjust the wheel rotation rate and the steering of the East action being performed by the task process Υ, and (3) the conflict resolver σ selects the optimal parameter $(\Downarrow, \Rightarrow)$ that adjusts the East action being performed by the task process Υ given the parameters $(\varnothing, \Rightarrow)$, $(\varnothing, \varnothing)$, and $(\Downarrow, \varnothing)$ recommended by the safety processes θ_c, θ_d, and θ_r. It is important to note that this example refers to a *parameter* for each safety process instead of a *rating over a set of parameters* in the interest of illustrating our approach.

6.1.1 Contributions

In this chapter, we make the following contributions: (1) a formal definition of a safety metareasoning system and its key attributes, (2) a recommendation algorithm for a safety process, (3) an arbitration algorithm for a conflict resolver, (4) an application of a safety metareasoning system to a planetary rover exploration domain, and (5) a demonstration that our approach is effective in simulation.

6.2 Safety

We begin by proposing the metareasoning framework, called a *safety metareasoning system*, that enables an autonomous system to maintain and restore safety. A safety metareasoning system executes in parallel a *task process* that completes a specified task and *safety processes* that each address a specified safety concern with a *conflict resolver* for arbitration. We describe each attribute of our approach below.

6.2.1 Completing Tasks

The task process completes a specified task by performing an action in its current state following its policy. The representation of the task process must reflect the properties of the task. In this chapter, the task process is represented by an MDP, a decision process for tasks with full observability, because it is a standard model used throughout planning and robotics [140]. However, it is possible to use different classes of decision processes for tasks with partial observability [76] or start and goal states [82]. We define the task process more formally below.

Definition 24. *The **task process**, which is represented by an MDP $\Upsilon = \langle S, A, T, R \rangle$, performs an action $a = \pi(s) \in A$ in a state $s \in S$ following a policy π in order to complete a specified task.*

Example. *To complete the analysis task, the planetary rover in the highlighted area of Figure 6.1 must execute the task process Υ that performs the* East *action starting in the cell $(3, 7)$ and ending in the cell $(4, 7)$.*

6.2.2 Addressing Safety Concerns

A safety process addresses a specified safety concern by recommending a rating over a set of parameters in its current state that can adjust the action being performed by the task process. The representation of a safety process is a variant of an MDP with several attributes: a set of states that describe the safety concern, a set of parameters that can adjust the action being performed by the task process, a transition function that reflects the dynamics of the world, a severity function that reflects the danger of particular hazards, and an interference function that reflects the overhead of safety on the main goal. We define a safety process more formally below.

Definition 25. *A **safety process**, which is represented by a variant of an MDP $\theta = \langle \bar{S}, \bar{P}, \bar{T}, \phi, \psi \rangle \in \Theta$, recommends a rating $\rho^\theta_{\bar{s}}$ over a set of parameters $\bar{P}$ in a state $\bar{s} \in \bar{S}$ that can adjust the action $a \in A$ being performed by the task process Υ in order to address a specified safety concern.*

- *$\bar{S}$ is a set of states that describe the safety concern.*
- *$\bar{P} = \bar{P}_1 \times \bar{P}_2 \times \cdots \times \bar{P}_N$ is a set of parameters such that each parameter factor $\bar{P}_i$ adjusts the action $a \in A$ being performed by the task process Υ with a $\varnothing \in \bar{P}_i$ symbol that indicates no adjustment.*

- $\bar{T} : \bar{S} \times \bar{P} \times \bar{S} \rightarrow [0, 1]$ *is a transition function that represents the probability of reaching a state* $\bar{s}' \in \bar{S}$ *after using a parameter* $\bar{p} \in \bar{P}$ *in a state* $\bar{s} \in \bar{S}$.

- $\phi : \bar{S} \rightarrow \{1, 2, \ldots, L\}$ *is a severity function that represents the severity of the safety concern in a state* $\bar{s} \in \bar{S}$ *such that* 1 *is the lowest severity level and* L *is the highest severity level where a severity level* $1 \leq \ell \leq L$ *is strictly safer than a severity level* $1 \leq \ell + 1 \leq L$.

- $\psi : \bar{P} \rightarrow \mathbb{R}^+$ *is an interference function that represents the interference of a parameter* $\bar{p} \in \bar{P}$ *on the action* $a \in A$ *being performed by the task process* Υ.

Example. *To address crevices, dust storms, and rough terrain, the planetary rover in the highlighted area of Figure 6.1 must execute the safety processes* θ_c, θ_d, *and* θ_r *that recommend the parameters* $(\varnothing, \Rightarrow)$, $(\varnothing, \varnothing)$, *and* $(\Downarrow, \varnothing)$ *that can adjust the wheel rotation rate and steering of the* EAST *action being performed by the task process* Υ.

It is critical that each safety process recommends *a rating over a set of parameters* instead of only *a parameter*. Intuitively, this enables the conflict resolver to select a parameter that addresses the safety concern of each safety process simultaneously. Accordingly, for a given state, this rating contains $|L| + 1$ values for each of the $|\bar{P}|$ parameters. For each severity level, it includes the *expected discounted cumulative frequency of a severity level* that is incurred by the safety process when using a specific parameter in a given state. It also includes the *expected discounted cumulative interference* that is incurred by the safety process when using a specific parameter in a given state. These quantities allow a safety metareasoning system to not only minimize each severity level but also minimize interference in the objective function described later in the chapter. We define a rating with these values below.

Definition 26. *A* ***rating****,* $\rho_{\bar{s}}^{\theta}$, *over a set of parameters* $\bar{P}$ *in a state* $\bar{s} \in \bar{S}$ *recommended by a safety process* $\theta \in \Theta$ *is expressed as a* $|\bar{P}| \times (|L| + 1)$ *matrix:*

$$\rho^\theta_{\bar{s}} = \begin{bmatrix} \Phi^\theta_{\bar{s},\bar{p}_1}[1] & \Phi^\theta_{\bar{s},\bar{p}_1}[2] & \dots & \Phi^\theta_{\bar{s},\bar{p}_1}[L] & \Psi^\theta_{\bar{s},\bar{p}_1} \\ \Phi^\theta_{\bar{s},\bar{p}_2}[1] & \Phi^\theta_{\bar{s},\bar{p}_2}[2] & \dots & \Phi^\theta_{\bar{s},\bar{p}_2}[L] & \Psi^\theta_{\bar{s},\bar{p}_2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \Phi^\theta_{\bar{s},\bar{p}_N}[1] & \Phi^\theta_{\bar{s},\bar{p}_N}[2] & \dots & \Phi^\theta_{\bar{s},\bar{p}_N}[L] & \Psi^\theta_{\bar{s},\bar{p}_N} \end{bmatrix}.$$

When a safety process $\theta \in \Theta$ uses a parameter $\bar{p} \in \bar{P}$ in a state $\bar{s} \in \bar{S}$, the ***expected discounted frequency of a severity level*** $1 \leq \ell \leq L$ *incurred is the following:*

$$\Phi^\theta_{\bar{s},\bar{p}}[\ell] = [\phi(\bar{s}) = \ell] + \gamma \sum_{\bar{s}' \in \bar{S}} \bar{T}(\bar{s}, \bar{p}, \bar{s}') \min_{\bar{p}' \in \bar{P}} \Phi^\theta_{\bar{s}',\bar{p}'}[\ell].$$

When a safety process $\theta \in \Theta$ uses a parameter $\bar{p} \in \bar{P}$ in a state $\bar{s} \in \bar{S}$, the ***expected discounted cumulative interference*** *incurred is the following:*

$$\Psi^\theta_{\bar{s},\bar{p}} = \psi(\bar{p}) + \gamma \sum_{\bar{s}' \in \bar{S}} \bar{T}(\bar{s}, \bar{p}, \bar{s}') \min_{\bar{p}' \in \bar{P}} \Psi^\theta_{\bar{s}',\bar{p}'}.$$

Note that the operator $[\cdot]$ denotes Iverson bracket notation.

6.2.3 Resolving Conflicts

The conflict resolver for arbitration selects the optimal parameter that will adjust the action being performed by the task process given the ratings over the set of parameters recommended by the safety processes. Intuitively, if no safety process or only one safety process encounters its safety concern, there is no need for arbitration. However, if multiple safety processes encounter their safety concerns, the conflict resolver arbitrates by selecting the parameter that optimally addresses each safety concern. The representation of the conflict resolver is a function that maps the ratings over the set of parameters recommended by the safety processes to a parameter. We define the conflict resolver more formally below.

Definition 27. *The **conflict resolver**, $\sigma : \rho^{\theta_1}_{\bar{s}^1} \times \rho^{\theta_2}_{\bar{s}^2} \times \cdots \times \rho^{\theta_n}_{\bar{s}^n} \rightarrow \bar{P}$, selects the optimal parameter $\bar{p} \in \bar{P}$ that adjusts the action $a \in A$ being performed by the task process Υ given the ratings $\rho^{\theta_i}_{\bar{s}^i}$ over the set of parameters $\bar{P}$ recommended by the safety processes $\theta_i \in \Theta$ for arbitration.*

Example. *For arbitration, the planetary rover in the highlighted area of Figure 6.1 must use the conflict resolver σ that selects the optimal parameter $(\Downarrow, \Rightarrow)$ that adjusts the* East *action being performed by the task process Υ given the parameters $(\varnothing, \Rightarrow)$, $(\varnothing, \varnothing)$, $(\Downarrow, \varnothing)$ recommended by the safety processes θ_c, θ_d, and θ_r.*

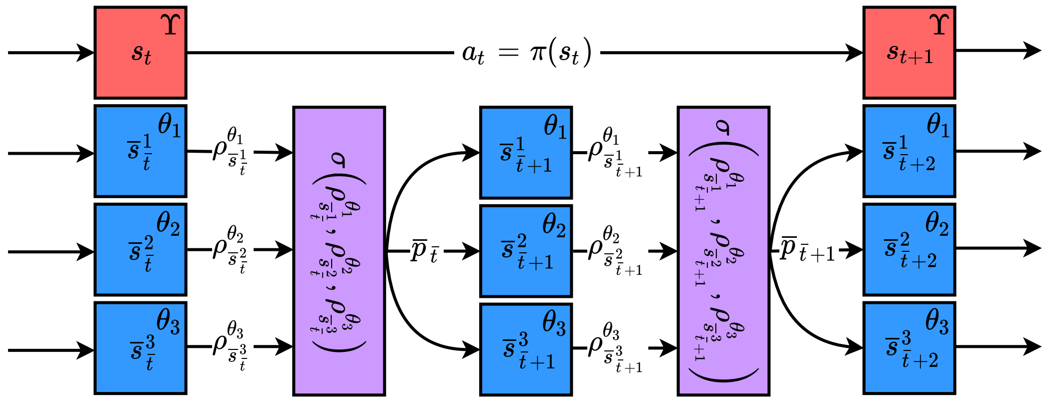
The optimal parameter selected by the conflict resolver satisfies a lexicographic objective function. This lexicographic objective function allows a safety metareasoning system to—in sequence—minimize each severity level and the interference of the safety processes. First, in the order of decreasing severity level, this parameter must *minimize* the *maximum* expected discounted frequency of each severity level incurred across all safety processes (minimize the maximum anticipated danger of particular hazards). Second, this parameter must *minimize* the *maximum* expected discounted cumulative interference incurred across all safety processes (minimize the maximum anticipated overhead of safety on the main goal). Formally, given each rating $\rho^{\theta_i}_{\bar{s}^i}$ over the set of parameters $\bar{P}$ recommended by each safety process $\theta_i \in \Theta$ in its state $\bar{s}^i \in \bar{S}^i$, we define this function below.

$$\min_{\bar{p} \in \bar{P}} \Big[\max_{\theta_i \in \Theta} \big[\Phi^{\theta_i}_{\bar{s}^i, \bar{p}}[L] \succ \Phi^{\theta_i}_{\bar{s}^i, \bar{p}}[L-1] \succ \cdots \succ \Phi^{\theta_i}_{\bar{s}^i, \bar{p}}[1] \succ \Psi^{\theta_i}_{\bar{s}^i, \bar{p}} \big] \Big]$$

Note that the lexicographic preference operator $\succ$ denotes that the left term is always optimized prior to the right term without any slack.

6.2.4 Safety Metareasoning Systems

Putting the analysis process, the safety processes, and the conflict resolver together, we now provide a description of a safety metareasoning system below.



Υ
s_t
$a_t = \pi(s_t)$
Υ
s_{t+1}
$\bar{s}^1_{\bar{t}}$ θ_1
$\bar{s}^2_{\bar{t}}$ θ_2
$\bar{s}^3_{\bar{t}}$ θ_3
$\rho^{\theta_1}_{\bar{s}^1_{\bar{t}}}$
$\rho^{\theta_2}_{\bar{s}^2_{\bar{t}}}$
$\rho^{\theta_3}_{\bar{s}^3_{\bar{t}}}$
$\sigma\left(\rho^{\theta_1}_{\bar{s}^1_{\bar{t}}}, \rho^{\theta_2}_{\bar{s}^2_{\bar{t}}}, \rho^{\theta_3}_{\bar{s}^3_{\bar{t}}}\right)$
$\bar{p}_{\bar{t}}$
$\bar{s}^1_{\bar{t}+1}$ θ_1
$\bar{s}^2_{\bar{t}+1}$ θ_2
$\bar{s}^3_{\bar{t}+1}$ θ_3
$\rho^{\theta_1}_{\bar{s}^1_{\bar{t}+1}}$
$\rho^{\theta_2}_{\bar{s}^2_{\bar{t}+1}}$
$\rho^{\theta_3}_{\bar{s}^3_{\bar{t}+1}}$
$\sigma\left(\rho^{\theta_1}_{\bar{s}^1_{\bar{t}+1}}, \rho^{\theta_2}_{\bar{s}^2_{\bar{t}+1}}, \rho^{\theta_3}_{\bar{s}^3_{\bar{t}+1}}\right)$
$\bar{p}_{\bar{t}+1}$
$\bar{s}^1_{\bar{t}+2}$ θ_1
$\bar{s}^2_{\bar{t}+2}$ θ_2
$\bar{s}^3_{\bar{t}+2}$ θ_3

Figure 6.2: A safety metareasoning system that has the task process (*red*), a set of safety processes (*blue*), and the conflict resolver (*purple*).

Definition 28. *A **safety metareasoning system**, $\langle \Upsilon, \Theta, \sigma \rangle$, runs in parallel a task process Υ that completes a specified task and safety processes Θ that each address a specified safety concern with a conflict resolver σ for arbitration.*

Figure 6.2 summarizes a safety metareasoning system. There is a task transition for the task process Υ from the state $s_t \in S$ at time step $t \in H$ to the successor state $s_{t+1} \in S$ at time step $(t + 1) \in H$ given the action $a_t = \pi(s_t) \in A$. During this task transition, there are many safety transitions for each safety process $\theta_i \in \Theta$ from the state $\bar{s}^i_{\bar{t}} \in \bar{S}^i$ at time step $\bar{t} \in \bar{H}$ to the successor state $\bar{s}^i_{\bar{t}'} \in \bar{S}^i$ at time step $\bar{t}' \in \bar{H}$. In each safety transition, each safety process $\theta_i \in \Theta$ recommends a rating $\rho^{\theta_i}_{\bar{s}^i_{\bar{t}}}$ over the set of parameter $\bar{P}$ to the conflict resolver σ. The conflict resolver σ then selects the optimal parameter $\bar{p}_{\bar{t}} \in \bar{P}$ that satisfies the lexicographic objective function. Once the optimal parameter $\bar{p}_{\bar{t}} \in \bar{P}$ is selected by the conflict resolver σ, the action $a_t = \pi(s_t) \in A$ of the task process Υ is adjusted in a way that reflects that optimal parameter. Notice that the task process Υ operates on course-grained time steps $t \in H$ while each safety process $\theta_i \in \Theta$ operates on fine-grained time steps $\bar{t} \in \bar{H}$ as the actions performed by the task process can continually be adjusted by the parameters recommended by the safety processes.

The actions of the task process and the parameters of the safety processes are tightly integrated. In particular, a safety metareasoning system must send an action and a parameter to a motion planner that computes motor commands that reflect performing the action subject to the constraints imposed by the parameter. Suppose that a planetary rover performs the NORTH action with the $(\Downarrow, \Leftarrow)$ parameter for slowing down and shifting left. Here, the planetary rover must send the NORTH action and the $(\Downarrow, \Leftarrow)$ parameter to the motion planner that must compute motor commands that move the planetary rover north subject to the constraints of slowing down and swerving left. Formally, an action $a \in A$ of a task process can be viewed as a parameterized action $a[\bar{p}] \in A$ given a parameter $\bar{p} \in \bar{P}$ of the safety processes.

In the following sections, we describe two main algorithms required by a safety metareasoning system. First, the *recommendation algorithm* generates a matrix that is used to construct the rating for each state of a safety process. Second, the *arbitration algorithm* implements the conflict resolver that selects the optimal parameter across all safety processes. We now describe both algorithms in detail below.

6.2.4.1 Recommendation Algorithm

The *recommendation algorithm* in Algorithm 4 generates a matrix that is used to construct the rating for each state of a safety process (between the *blue* and *purple* objects in Figure 6.2). In particular, given a safety process, the recommendation algorithm generates multiple values that are discussed in the prior section for every state-parameter pair of the safety process: the *expected discounted frequency of each severity level* incurred and the *expected discounted cumulative interference* incurred when using a parameter in a state for each state and parameter of that safety process. The basis for computing these values involves—for each severity level in the order of decreasing severity level followed by the interference—performing $|L| + 1$ executions

Algorithm 4: The recommendation algorithm for a safety process.

```
Input: The safety process $\theta = \langle \bar{S}, \bar{P}, \bar{T}, \phi, \psi \rangle$
Output: The matrix $\rho^\theta$ that is used to construct the rating $\rho^\theta_{\bar{s}}$ for each state
        $\bar{s} \in \bar{S}$ of the safety process $\theta$
1  for $\ell \rightarrow L, L-1, \ldots, 1$ do
2      $\Phi^\theta[\ell] \leftarrow 0_{\bar{S} \times \bar{P}}$
3  $\Psi^\theta \leftarrow 0_{\bar{S} \times \bar{P}}$
4  $\Lambda \leftarrow \emptyset$
5  for $\ell \rightarrow L, L-1, \ldots, 1$ do
6      $\kappa(\bar{s}) \coloneqq [\phi(\bar{s}) = \ell]$
7      $\Phi^\theta[\ell] \leftarrow$ ModifiedValueIteration$(\theta, \kappa, \Lambda)$
8      for $\bar{s}$ in $\bar{S}$ do
9          $\alpha \leftarrow \min_{\bar{p} \in \bar{P}} \Phi^\theta_{\bar{s},\bar{p}}[\ell]$
10         for $\bar{p}$ in $\bar{P}$ do
11             if $\Phi^\theta_{\bar{s},\bar{p}}[\ell] > \alpha$ then
12                 $\Lambda \leftarrow \Lambda \cup (\bar{s}, \bar{p})$
13 $\kappa(\bar{p}) \coloneqq \psi(\bar{p})$
14 $\Psi^\theta \leftarrow$ ModifiedValueIteration$(\theta, \kappa, \Lambda)$
15 return $\rho^\theta = \left[\Phi^\theta[1], \Phi^\theta[2], \ldots, \Phi^\theta[L], \Psi^\theta\right]$
```

of modified value iteration that operates on a space of states and parameters with a cost function in place of a space of states and actions with a reward function.

At a high level, the recommendation algorithm executes modified value iteration for each severity level in the order of decreasing severity level followed by the interference to satisfy the lexicographic objective function. Generally, as we will describe later this section, this algorithm performs two steps. First, it executes modified value iteration for each severity level in the order of decreasing severity level. Second, it executes modified value iteration solely for the interference. Across every execution of modified value iteration, the algorithm keeps a set of state-parameter pairs that violate the lexicographic objective function, which grows with each execution of modified value iteration. This set of violating state-parameter pairs is thereby used in each execution of modified value iteration to ignore—for each state—any parameter worse than the optimal parameter in the prior executions of modified value iteration. The final output of the recommendation algorithm is a matrix that is used to construct the rating for each state of the safety process. Note that this algorithm is performed *offline* for each safety process before the operation of the safety metareasoning system.

Algorithm 4 describes the recommendation algorithm. Initially, for each severity level and the interference, an $|\bar{S}| \times |\bar{P}|$ matrix is initialized (Lines 1-3). The $|\bar{S}| \times |\bar{P}|$ matrix for each severity level (Lines 5-7) and the interference (Lines 13-14) is then populated with its corresponding expected discounted values using *modified* value iteration that minimizes over *states* and *parameters* given a *cost* function instead of maximizing over *states* and *actions* given a *reward* function. Observe that the cost function $\kappa(\bar{s})$ is used to compute the expected discounted frequency of each severity level (Line 6) while the cost function $\kappa(\bar{p})$ is used to compute the expected discounted cumulative interference (Line 13). Finally, the $|\bar{S}| \times |\bar{P}|$ matrix for each severity level and the interference is stacked into an $|\bar{S}| \times |\bar{P}| \times (L+1)$ matrix (Line 15) that is used to construct the rating for each state of a safety process.

Most importantly, in order to satisfy the lexicographic objective function, a set of violating state-parameter pairs is initialized (Line 4). For each severity level, a state-parameter pair is added to the set of violating state-parameter pairs *if* that parameter in that state is worse than the optimal parameter in that state (Lines 8-12). The set of violating state-parameter pairs enables modified value iteration to forbid every state-parameter pair that did not satisfy the lexicographic objective function from the previous executions of modified value iteration (Lines 7 and 14).

We provide a sketch that demonstrates the correctness and the worst-case time complexity of the recommendation algorithm below.

Proposition 2 (Correctness). *Algorithm 4 generates a matrix ρ^θ of the expected discounted frequency $\Phi^\theta_{\bar{s},\bar{p}}[\ell]$ of each severity level $1 \leq \ell \leq L$ and the expected discounted* *cumulative interference $\Psi^{\theta}_{\bar{s},\bar{p}}$ for each state $\bar{s} \in \bar{S}$ and parameter $\bar{p} \in \bar{P}$ of a safety process $\theta \in \Theta$ that satisfies the lexicographic objective function.*

Proof Sketch. Observe that there is an execution of a form of value iteration for each severity level and the interference in the order of the lexicographic objective function. It is well known that standard value iteration without any set of violating state-parameter pairs is guaranteed to compute the corresponding expected discounted values for each severity level and the interference but may not satisfy the lexicographic objective function. However, by forbidding the set of violating state-parameter pairs, modified value iteration satisfies the lexicographic objective function. Thus, Algorithm 4 is correct. □

Proposition 3 (Time Complexity)**.** *Algorithm 4 exhibits a worst-case time complexity of $O((L+1)|\bar{S}|^2|\bar{P}|)$.*

Proof Sketch. There are $L + 1$ executions of value iteration that each have a time complexity of $O(|\bar{S}|^2|\bar{P}|)$ for a total time complexity of $O((L+1)|\bar{S}|^2|\bar{P}|)$. □

6.2.4.2 Arbitration Algorithm

The *arbitration algorithm* in Algorithm 5 implements the conflict resolver that selects the parameter that optimally addresses the safety concern of each safety process (between the *purple* and *blue* objects in Figure 6.2). At a high level, this algorithm prunes a set of potentially optimal parameters for each severity level in the order of decreasing severity level followed by the interference to optimize the lexicographic objective function. That is, this algorithm initializes a set of potentially optimal parameters that can be recommended by each safety process. Given the ordering of the lexicographic objective function, the algorithm prunes the set of potentially optimal parameters in two steps. First, the algorithm prunes the set of potentially optimal parameters in the order of decreasing severity level based on the expected discounted frequency of each severity level incurred using a given parameter in a specific state.

Algorithm 5: The arbitration algorithm for a conflict resolver.

```
Input: The ratings $\rho^{\theta_i}_{\bar{s}^i}$ in the current state $\bar{s}^i \in \bar{S}^i$ of the safety processes
       $\theta_i \in \Theta$
Output: A random optimal parameter $\bar{p} \in \bar{P}$
1 $\bar{P}^* \leftarrow \bar{P}$
2 for $\nu \to \Phi[L], \Phi[L-1], \dots, \Phi[1], \Psi$ do
3     $\alpha \leftarrow \min_{\bar{p} \in \bar{P}} \left[\max_{\theta_i \in \Theta} \nu^{\theta_i}_{\bar{s}^i, \bar{p}}\right]$
4     for $\bar{p}$ in $\bar{P}^*$ do
5         $\beta \leftarrow \max_{\theta_i \in \Theta} \nu^{\theta_i}_{\bar{s}^i, \bar{p}}$
6         if $\beta > \alpha$ then
7             $\bar{P}^* \leftarrow \bar{P}^* \setminus \{\bar{p}\}$
8 return RANDOM($\bar{P}^*$)
```

Second, the algorithm prunes the set of potentially optimal parameters based on the expected discounted cumulative interference incurred by the safety process using a given parameter in a specific state. The result is a random optimal parameter that optimizes the lexicographic objective function. Note that this algorithm is performed *online* during the operation of the safety metareasoning system.

Algorithm 5 describes the arbitration algorithm. Initially, a set of potentially optimal parameters is initialized (Line 1). Each severity level in the order of decreasing severity level followed by the interference is then processed (Line 2). To optimize the lexicographic objective function, the set of potentially optimal parameters is then pruned (Line 3-7). This involves computing the value of the parameter that minimizes the maximum respective expected discounted value over all safety processes (Line 3). With the value of this parameter, each parameter that has a maximum respective expected discounted value greater than that value is pruned (Line 4-7). Finally, a random optimal parameter that optimally addresses each safety process is selected (Line 8). This algorithm is performed *online* during the operation of the system.

We provide a sketch that demonstrates the correctness and the worst-case time complexity of the arbitration algorithm below.

Proposition 4 (Correctness)**.** *Algorithm 5 selects a random optimal parameter* $\bar{p} \in \bar{P}$ *that optimizes the lexicographic objective function given the ratings* $\rho^{\theta_i}_{\bar{s}^i}$ *in the current state* $\bar{s}^i \in \bar{S}^i$ *of the safety processes* $\theta_i \in \Theta$*.*

Proof Sketch. In the order of the lexicographic objective function, any parameter with a maximum expected discounted frequency greater than the optimal parameter for each severity level is pruned and any parameter with a maximum discounted cumulative interference greater than the optimal parameter for the interference is pruned. Since this optimizes the lexicographic objective function, it follows that any remaining parameter is optimal. Hence, Algorithm 5 is correct. □

Proposition 5 (Time Complexity)**.** *Algorithm 5 exhibits a worst-case time complexity of* $O((L+1)|\bar{P}||\Theta|)$*.*

Proof Sketch. There are L severity level pruning steps that each have a time complexity of $O(|\bar{P}||\Theta|)$ and an interference pruning step that has a time complexity of $O(|\bar{P}||\Theta|)$ for a total time complexity of $O((L+1)|\bar{P}||\Theta|)$. □

6.3 Planetary Rover Exploration Domain

We turn to an application of safety metareasoning systems to a planetary rover exploration domain [44] that forms the basis of the experiments used to evaluate our approach in the next section. Such a domain is relevant because a planetary rover should address its safety concerns while completing its task without relying on any human assistance from mission control on Earth that could take hours or even days. This would not only slow down every mission but also make some missions infeasible.

In this domain, a planetary rover must analyze different points of interest within a region of a planet while addressing crevices, dust storms, and rough terrain. The planetary rover has a battery, a rock analyzer, a soil analyzer, and an objective report for the analysis statuses of all points of interests. Moreover, the planetary rover is within a region of the planet that is composed as a grid where each cell experiences a type of weather. In each cell, the planetary rover can move north, east, south, or west and can also reboot its analyzers, charge its battery, analyze its current cell, and transmit its data back to mission control on Earth to complete its mission. We now describe the analysis task of the planetary rover in detail below.

The planetary rover has 4 internal components: a battery of a battery level $b \in B = \{0, 1, \ldots, M\}$ where 0 is a discharged battery and M is a charged battery, a rock analyzer of a health status $h_1 \in H_1 = \{\textsc{Nominal}, \textsc{Error}\}$, a soil analyzer of a health status $h_2 \in H_2 = \{\textsc{Nominal}, \textsc{Error}\}$, and an objective report $o \in O = \{\textsc{True}, \textsc{False}\}^I$ with an analysis status True or False for all points of interest I.

The planetary rover is within a region of a planet as an m by n grid where each cell is at a horizontal location $x \in X = \{1, 2, \ldots, n\}$ and a vertical location $y \in Y = \{1, 2, \ldots, m\}$ with weather of a type $w \in W = \{\textsc{Light}, \textsc{Dark}\}$.

The planetary rover can perform 4 movement actions in each cell (x, y): it can move *north* to a cell $(x, y + 1)$, *east* to a cell $(x + 1, y)$, *south* to a cell $(x, y - 1)$, or *west* to a cell $(x - 1, y)$ as long as the new horizontal position is between 1 and n and the new vertical position is between 1 and m.

The planetary rover can perform 4 stationary actions in each cell (x, y): it can *reboot* its analyzers to set the health statuses of the rock analyzer h_1 and the soil analyzer h_2 to Nominal, *charge* its battery to the battery level $b' = (b + 2)$ if the cell (x, y) has weather of a type $w = \textsc{Light}$, *analyze* the cell (x, y) if the health statuses of the rock analyzer h_1 and the soil analyzer h_2 are set to Nominal, and *transmit* its data to complete the mission if the objective report is $o = \{\textsc{True}\}^I$ with an analysis status True for all points of interest I.

Importantly, to incorporate battery management, all actions discharge the battery to a battery level $b' = (b - 1)$ and requires the battery to be at a battery level $b > 0$.

6.3.1 Task Process

We consider the task process Υ designed to complete the analysis task of the planetary rover that is represented by an MDP $\Upsilon = \langle S, A, T, R \rangle$. The set of states $S = X \times Y \times B \times H_1 \times H_2 \times O$ crosses a set of horizontal positions X, a set of vertical positions Y, a set of battery levels B, a set of rock analyzer health statuses H_1, a set of soil analyzer health statuses H_2, and a set of objective reports O. The set of actions $A = \{\uparrow, \rightarrow, \downarrow, \leftarrow, \ominus, \oplus, \odot, \oslash\}$ contains the *north* action $\uparrow$, the *east* action $\rightarrow$, the *south* action $\downarrow$, the *west* action $\leftarrow$, the *reboot* action $\ominus$, the *charge* action $\oplus$, the *analyze* action $\odot$, and the *transmit* action $\oslash$. The transition function T and the reward function R are designed for the analysis task of the task process Υ.

6.3.2 Safety Processes

We consider each safety process, $\theta = \langle \bar{S}, \bar{P}, \bar{T}, \phi, \psi \rangle \in \Theta$, designed to address a safety concern of the planetary rover. Intuitively, each safety process has information about its safety concern and can adjust the action performed by the task process by changing its wheel rotation rate (i.e., *speed*) and its steering (i.e., *direction*).

Formally, each safety process $\theta \in \Theta$ has a different set of states $\bar{S}_\theta$ that describe the safety concern but uses the same set of parameters $\bar{P} = \bar{P}_1 \times \bar{P}_2$ with parameter factors $\bar{P}_1$ and $\bar{P}_2$: the wheel rotation rate parameter factor $\bar{P}_1 = \{\varnothing, \Downarrow, \Uparrow, \Diamond\}$ such that the value $\Downarrow$ decreases the wheel rotation rate, the value $\Uparrow$ increases the wheel rotation rate, and the value $\Diamond$ stops the wheel rotation rate and the steering parameter factor $\bar{P}_2 = \{\varnothing, \Leftarrow, \Rightarrow\}$ such that the value $\Leftarrow$ shifts the planetary rover to the left and the value $\Rightarrow$ shifts the planetary rover to the right (with the $\varnothing$ symbol that indicates no adjustment in both parameter factors). The transition function $\bar{T}_\theta$, the severity function ϕ_θ, and the interference function ψ_θ are designed to address the specific safety concern of each safety process $\theta \in \Theta$. We describe the crevice safety process, the dust storm safety process, and the rough terrain safety process below.

6.3.2.1 Crevices

The crevice safety process, $\theta_c = \langle \bar{S}_c, \bar{P}, \bar{T}_c, \phi_c, \psi_c \rangle$, monitors for crevices to prevent the planetary rover from inhibiting the movement of its wheels. The set of states $\bar{S}_c = F_c^1 \times F_c^2 \times F_c^3 \times F_c^4$ crosses the horizontal rover position relative to the crevice $F_c^1 = \{\textsc{None}, \textsc{Approaching}, \textsc{At}\}$, the vertical rover position relative to the crevice $F_c^2 = \{\textsc{None}, \textsc{Left}, \textsc{Center}, \textsc{Right}\}$, the rover speed $F_c^3 = \{\textsc{None}, \textsc{Low}, \textsc{Normal}, \textsc{High}\}$, and the rover offset relative to its normal path $F_c^4 = \{\textsc{Left}, \textsc{Center}, \textsc{Right}\}$. The transition function $\bar{T}_c$ reflects the dynamics between a state $\bar{s} \in \bar{S}_c$, a parameter $\bar{p} \in \bar{P}_c$, and a successor state $\bar{s}' \in \bar{S}_c$, the severity function ϕ_c indicates the severity of a crevice in a state $\bar{s} \in \bar{S}_c$, and the interference function ψ_c represents the interference of a parameter $\bar{p} \in \bar{P}_c$ on an action $a \in A$ performed by the task process. These three attributes are designed to enable the crevice safety process to avoid navigating into crevices that inhibit the movement of the wheels of the planetary rover.

6.3.2.2 Dust Storms

The dust storm safety process, $\theta_d = \langle \bar{S}_d, \bar{P}, \bar{T}_d, \phi_d, \psi_d \rangle$, monitors for dust storms to prevent the planetary rover from damaging its sensitive sensors. The set of states $\bar{S}_d = F_d^1 \times F_d^2$ crosses the dust storm density $F_d^1 = \{1, 2, \ldots, J\}$ with a limit J and the rover mode $F_d^2 = \{\textsc{IsAwake}, \textsc{IsSleeping}\}$. The transition function $\bar{T}_d$ reflects the dynamics between a state $\bar{s} \in \bar{S}_d$, a parameter $\bar{p} \in \bar{P}_d$, and a successor state $\bar{s}' \in \bar{S}_d$, the severity function ϕ_d indicates the severity of the dust storm in a state $\bar{s} \in \bar{S}_d$, and the interference function ψ_d represents the interference of a parameter $\bar{p} \in \bar{P}_d$ on an action $a \in A$ performed by the task process. These three attributes are designed to enable the dust storm safety process to avoid damaging the sensitive sensors of the planetary rover.

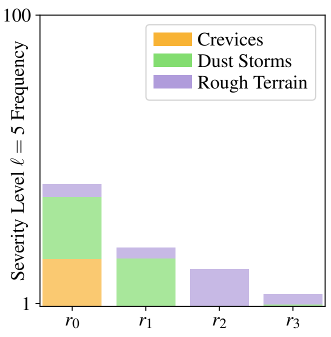
6.3.2.3 Rough Terrain

The rough terrain safety process, $\theta_r = \langle \bar{S}_r, \bar{P}, \bar{T}_r, \phi_r, \psi_r \rangle$, monitors for rough terrain to prevent the planetary rover from damaging its wheels. The set of states $\bar{S}_r = F_r^1 \times F_r^2 \times F_r^3$ crosses the horizontal rover position relative to the crevice $F_r^1 = \{\textsc{None}, \textsc{Approaching}, \textsc{At}\}$, the rover speed $F_r^2 = \{\textsc{None}, \textsc{Low}, \textsc{Normal}, \textsc{High}\}$, and the terrain roughness $F_r^3 = \{1, 2, \ldots, K\}$ with a limit K. The transition function $\bar{T}_r$ reflects the dynamics between a state $\bar{s} \in \bar{S}_r$, a parameter $\bar{p} \in \bar{P}_r$, and a successor state $\bar{s}' \in \bar{S}_r$, the severity function ϕ_r indicates the severity of the rough terrain in a state $\bar{s} \in \bar{S}_r$, and the interference function ψ_r represents the interference of a parameter $\bar{p} \in \bar{P}_r$ on an action $a \in A$ performed by the task process. These three attributes are designed to enable the rough terrain safety process to avoid damaging the wheels of the planetary rover.

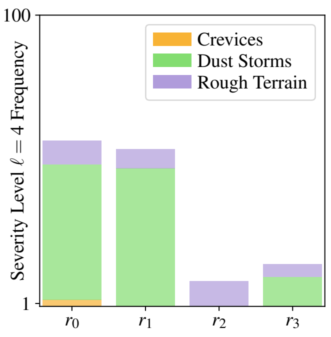
6.4 Demonstration

In this section, we demonstrate that the application of safety metareasoning systems to the planetary rover exploration domain is effective in simulation. In particular, we compare a standard planetary rover to different safety metareasoning planetary rovers. The standard planetary rover r_0 does not have any safety metareasoning while each safety metareasoning planetary rover $r_{i>0}$ has a growing set of safety processes: $\Theta^{r_0} = \{\}$, $\Theta^{r_1} = \{\theta_c\}$, $\Theta^{r_2} = \{\theta_c, \theta_d\}$, and $\Theta^{r_3} = \{\theta_c, \theta_d, \theta_r\}$.

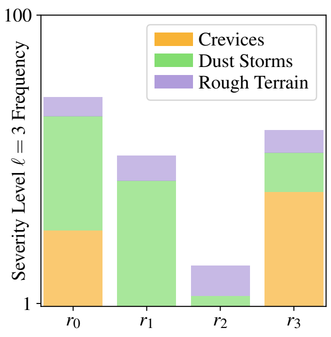
Each planetary rover must complete the analysis task while addressing crevices, dust storms, and rough terrain that occur stochastically either in isolation or simultaneously during 50 simulations. For the analysis task, the internal components of the planetary rover begin with a battery level $b = M = 10$, health statuses $h_1 = h_2 = \textsc{Nominal}$, and an objective report $o = (\textsc{False}, \textsc{False})$ while the region of the planet has $|I| = 2$ points of interest in an $n = 10$ by $m = 10$ grid such that each cell has weather of a type $w = \textsc{Light}$ with 0.8 probability or $w = \textsc{Dark}$



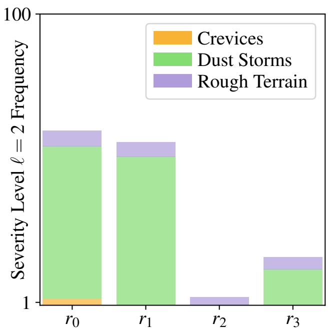
(a) Severity Level $\ell = 5$



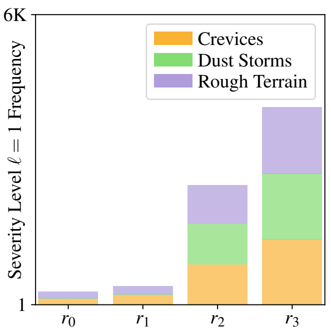
(b) Severity Level $\ell = 4$



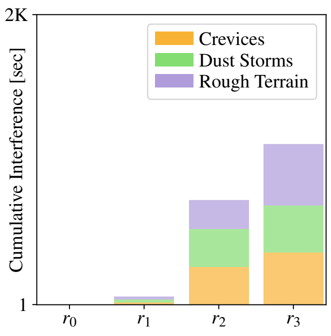
(c) Severity Level $\ell = 3$



(d) Severity Level $\ell = 2$



(e) Severity Level $\ell = 1$



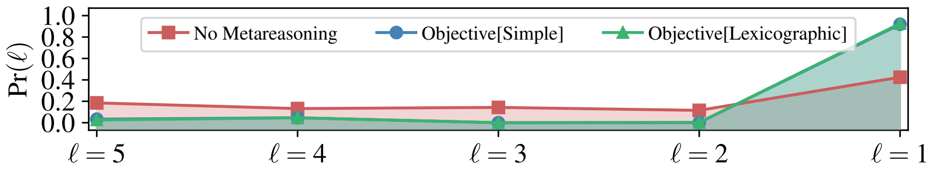
(f) Interference

Figure 6.3: The performance of each planetary rover for the severity levels and the interference starting with no safety processes and ending with all safety processes.

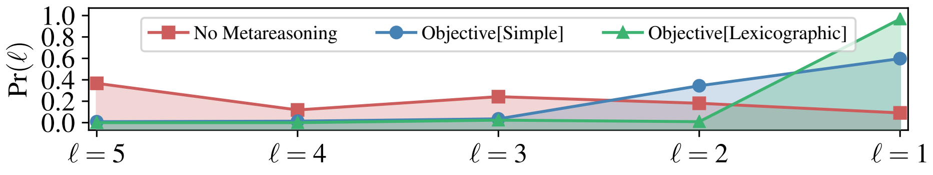
with 0.2 probability. For dust storms, the dust storm density limit is $J = 10$. For rough terrain, the terrain roughness limit is $K = 10$.

6.5 Discussion

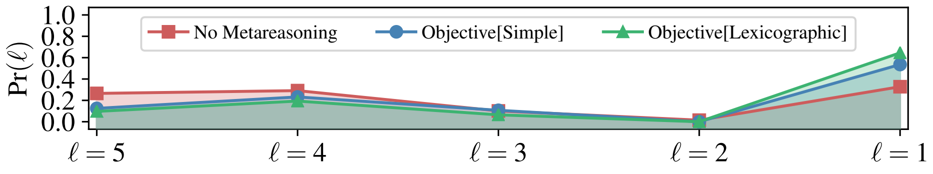
Figure 6.3 shows that our approach is effective in simulation. Note that Figures 6.3a to 6.3d have a limit of 100 as unsafe operation is rare, Figure 6.3e has a limit of 6000 as safe operation is common, and Figure 6.3f has a limit of 2000. In Figure 6.3a and 6.3e, at the highest and lowest severity levels, the severity level 5 frequency decreases while the severity level 1 frequency increases from r_0 to r_3 as expected. In Figures 6.3b, 6.3c, and 6.3d, at the middle severity levels, the severity level 4, 3, and 2 frequencies remain roughly equal or decrease from r_0 to r_2 but then increase at r_3. This is because the severity level 4, 3, and 2 frequencies for crevices and



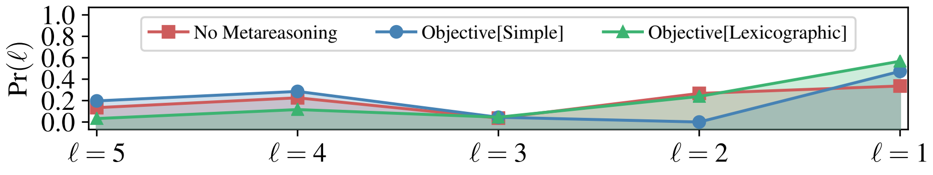
(a) No safety concern or isolated crevices, dust storms, and rough terrain.



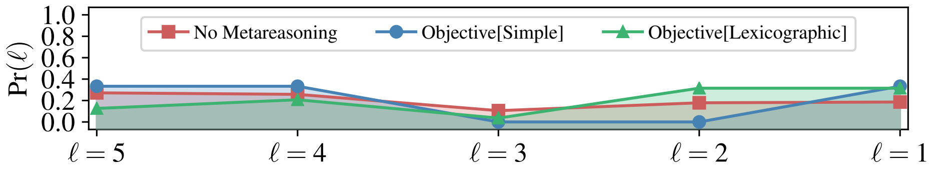
(b) Occurrences of simultaneous crevices and dust storms.



(c) Occurrences of simultaneous crevices and rough terrain.



(d) Occurrences of simultaneous dust storms and rough terrain.



(e) Occurrences of simultaneous crevices, dust storms, and rough terrain.

Figure 6.4: The severity level probability distributions for different combinations of safety concerns across every simulation.

dust storms must increase to decrease the severity level 5 frequency for rough terrain because a lower severity level is strictly preferred to a higher severity level due to the lexicographic objective function. In Figure 6.3f, the cumulative interference increases from r_0 to r_3 as expected. This is because the interference must increase to shift the severity level frequencies from the severity level 5 to the severity level 1 but only as

Capabilities	Size[Naive]	Size[Proposed]	Overhead[Proposed] (s)
Analysis Task	16000	16000	$4.23 \times 10^{-7} \pm 1.75 \times 10^{-8}$
+ Crevices	+ 2288000	+ 144	$6.63 \times 10^{-5} \pm 2.53 \times 10^{-7}$
+ Dust Storms	+ 43776000	+ 20	$7.97 \times 10^{-5} \pm 1.16 \times 10^{-7}$
+ Rough Terrain	+ 5483520000	+ 120	$1.08 \times 10^{-4} \pm 2.87 \times 10^{-7}$

Table 6.1: A comparison of a naive approach and our approach to safety.

much as necessary due to the lexicographic objective function. Overall, the system optimizes the severity of its safety concerns and the interference to its task.

Figure 6.4 compares our approach with the *lexicographic* objective function to a *simple* objective function for arbitration. The simple objective function always addresses safety concerns both sequentially and independently: that is, it first addresses a crevice (if any), then a dust storm (if any), and finally rough terrain (if any) *without* reasoning about how addressing one safety concern could impact other safety concerns or how addressing multiple safety concerns could be performed simultaneously. Typically, for each figure, the lexicographic objective function (Objective[Lexicographic]) exhibits a severity level probability distribution that encourages low severity levels but discourages high severity levels compared to the simple objective function (Objective[Simple]). Note that we use a baseline approach that has no metareasoning to evaluate the lexicographic objective function and the simple objective function.

Table 6.1 compares our *proposed* approach to a *naive* approach that would use a monolithic MDP with every feature of the analysis task and each safety process. However, note that the naive approach is intractable to design and solve given the complexity of its state space and action space. Generally, as the agent becomes capable of addressing each safety concern by including the set of states for each safety process, the naive approach grows multiplicatively (Size[Naive]) while our approach grows additively (Size[Proposed]) with negligible overhead for arbitration (Overhead[Proposed]) with the set of states for each safety process (Capabilities).

6.6 Summary

This chapter introduces a metareasoning approach to safety. A safety metareasoning system executes in parallel a task process that completes a specified task and a set of safety processes that each address a specified safety concern with a conflict resolver for arbitration by using probabilistic planning. By decoupling an autonomous system into a task process and a set of safety processes, our approach provides a framework for autonomous systems to complete a task while addressing safety concerns in a way that avoids a monolithic decision-making model that is often not only intractable but also infeasible to build correctly. In our experiments, we show that an application of a safety metareasoning system to planetary rover exploration is effective in simulation.

CHAPTER 7
CONCLUSION

7.1 Summary of Contributions

The primary objective of this thesis has been to introduce principled metareasoning for monitoring and controlling the planning processes and the execution processes of an autonomous system in order for it to operate more effectively in the real world. To do this, we proposed four different forms of metareasoning in autonomous systems that led to not only more effective meta-level control for planning but also expanded the scope of meta-level control to execution. This has resulted in four metareasoning approaches that enable an autonomous system to (1) determine when to interrupt an anytime algorithm and act on the current solution, (2) adjust the hyperparameters of an adjustable algorithm at runtime, (3) detect, identify, and recover from exceptions during operation, and (4) maintain and restore safety during operation. For each approach, we demonstrated that it outperformed standard techniques on a number of common benchmark domains and applications, including autonomous driving and planetary rover exploration. Overall, this thesis highlights that metareasoning is a robust framework for boosting the efficiency and reliability of the decision making of autonomous systems in a way that goes beyond prevailing work. We summarize the contributions of this thesis below.

7.1.1 Metareasoning for Stopping

Chapter 3 introduced two metareasoning approaches to *stopping* of anytime algorithms. For the *model-based* approach, we first developed an online performance prediction framework that can be used by a meta-level control technique to predict the performance of an anytime algorithm. We then built a model-based meta-level control technique that determines the optimal stopping point by using the online performance prediction framework. For the *model-free* approach, we first developed a formal MDP representation of the meta-level control problem for anytime algorithms that can be used to learn the optimal stopping point of an anytime algorithm with reinforcement learning. We then built a model-free meta-level control technique that learns the optimal stopping point by using the formal MDP representation of the meta-level control problem for anytime algorithms. Finally, we showed that both approaches outperform existing meta-level control techniques that require substantial offline work on several common benchmark domains and a mobile robot domain.

7.1.2 Metareasoning for Hyperparameter Tuning

Chapter 4 introduced a metareasoning approach to *hyperparameter tuning* of adjustable algorithms. First, we developed a formal generalization of an anytime algorithm called an adjustable algorithm that can be interrupted at any time for its current solution with hyperparameters that can be tuned at runtime. Next, we built a meta-level control technique that monitors and controls an adjustable algorithm by using deep reinforcement learning to learn optimal stopping and optimal hyperparameter tuning. Finally, we showed that our approach boosts overall performance on a common benchmark domain that uses anytime weighted A* to solve a range of heuristic search problems and a mobile robot application that uses RRT* to solve motion planning problems.

7.1.3 Metareasoning for Exception Recovery

Chapter 5 introduced a metareasoning approach to *exception recovery*: an exception recovery metareasoning system interleaves a regular process with a set of exception handlers to detect, identify, and handle exceptions by using belief space planning. First, we developed a formal definition of an exception recovery metareasoning system and its key properties. Next, we offered a framework for profiling decision processes for the main decision process and each exception handler. Finally, we showed that an application of an exception recovery metareasoning system to autonomous driving is effective in simulation and on a fully operational prototype.

7.1.4 Metareasoning for Safety

Chapter 6 introduced a metareasoning approach to *safety*: a safety metareasoning system executes in parallel a task process that completes a specified task and a set of safety processes that each address a specified safety concern with a conflict resolver for arbitration by using probabilistic planning. First, we developed a formal definition of a safety metareasoning system and its key attributes. Next, we offered a formal definition of a safety metareasoning system as well as a recommendation algorithm for a safety process and an arbitration algorithm for a conflict resolver with a theoretical analysis of the correctness and worst-case time-complexity for each algorithm. Finally, we showed that an application of a safety metareasoning system to planetary rover exploration is effective in simulation.

7.2 Future Work

There are many promising areas of future work. We discuss several areas of future work in metareasoning for planning and metareasoning for execution below.

7.2.1 Metareasoning for Planning

Our work on metareasoning for planning has focused on monitoring and controlling a *single* algorithm. It is possible, however, to use metareasoning for monitoring and controlling *multiple* algorithms at once. Existing work has generally focused on *algorithm selection* methods that chooses the best algorithm to solve a specific instance of a problem among a set of algorithms [85] and *algorithm portfolio* methods that execute a set of algorithms in parallel by assigning different allocations of computation time to each algorithm [69, 109]. However, while these methods have been effective for general algorithms, they have not typically taken advantage of the main property of anytime algorithms. An interesting line of work could therefore focus on developing methods that solve a specific instance of a problem by executing multiple anytime algorithms in sequence that each share an intermediate solution representation. Intuitively, once the current anytime algorithm reaches a point of diminishing returns along its performance curve, it would pass its current intermediate solution to another anytime algorithm within the portfolio of anytime algorithms, which would perhaps overcome the diminishing returns of the current anytime algorithm. Overall, the goal of this line of work would be to compute solutions of better quality in less computation time by executing multiple anytime algorithms in sequence.

Our work on metareasoning for hyperparameter tuning can also be extended in several ways. First, while we offered a preliminary analysis of the meta-level control problem for monitoring and controlling anytime weighted A* and RRT*, there is further optimization and exploration that could be performed to find the best meta-level control problem for each algorithm. For example, the states of computation could include more informative features while the actions of computation could include more useful hyperparameters in order to produce more effective meta-level control. Next, although we conducted an initial exploration of how our approach chooses to adjust the hyperparameters of anytime weighted A* and RRT* that intuitively explains the boost in their overall performance, there is further experimentation that could be performed to better understand its choices. Finally, since our experiments employ the standard implementations of anytime weighted A* and RRT*, it would be beneficial to examine more sophisticated versions of each algorithm while also applying our approach to different classes of algorithms that are popular throughout planning and robotics. Needless to say, there are many extensions of our work on metareasoning for hyperparameter tuning that have not been investigated yet.

7.2.2 Metareasoning for Execution

Our work on metareasoning for both exception recovery and safety can be expanded in several ways. First, our approaches could be applied to a variety of domains, such as package delivery, physical therapy, and household assistance. This would demonstrate the generality of each approach and the challenges of applying them to different classes of domains. Next, our approaches could employ more sophisticated general-purpose exception handlers and safety processes that could be used in a range of different tasks that are common in planning and robotics. This would highlight that each approach can go beyond the preliminary exception handlers and safety processes considered in this thesis that have been simplified in the interest of clarity. Finally, for both approaches, it is possible to perform a deeper theoretical analysis of their assumptions and their guarantees for exception recovery or safety in different classes of domains. Overall, there is a seemingly endless amount of work that could be done to further develop metareasoning for exception recovery and safety.

7.2.3 Integrating Planning and Execution

This thesis proposes a two-pronged metareasoning framework with a planning module and an execution module. However, the communication between each module poses many interesting research questions surrounding how the modules could inform each other of their progress and problems. On the one hand, if the planning module computes a plan of low solution quality despite determining when to interrupt and how to tune the hyperparameters of its planning processes, this could indicate to the execution module that its execution processes may encounter more exceptions and safety concerns than usual. On the other hand, if the execution module observes that its execution processes continue to encounter more exceptions and safety concerns than usual, this could indicate to the planning module that it may need to monitor and control its planning processes in a different way. Developing a formal way of integrating the planning module and the execution module is an important area of research that we have not given attention to in this thesis.

7.3 Final Thoughts

Throughout this thesis, we have shown that metareasoning can be a powerful approach to building autonomous systems that are required to operate in noisy, stochastic, unstructured domains for long periods of time. In particular, we have shown that metareasoning can enable autonomous systems to optimize their own planning processes and their own execution processes for more efficient and reliable decision making. To this end, we have applied metareasoning to an array of algorithms and applications, including heuristic search, motion planning, genetic algorithms, simulating annealing, path planning, autonomous driving, and planetary rover exploration. Most importantly, the main objective of this thesis has been to demonstrate to researchers and practitioners in planning and robotics that many challenging research questions that arise during the design, development, and deployment of autonomous systems in the real world can be answered using metareasoning. Finally, while this thesis represents only a small contribution to building general-purpose autonomous systems with efficient and reliable decision making, we hope that researchers and practitioners find value in the main ideas that have been presented here.

BIBLIOGRAPHY

[1] Adenso-Diaz, Belarmino, and Laguna, Manuel. Fine-tuning of algorithms using fractional experimental designs and local search. *Operations Research 54*, 1 (2006), 99–114.

[2] Agrawal, Ravi, Wright, Timothy J., Samuel, Siby, Zilberstein, Shlomo, and Fisher, Donald L. Effects of a change in environment on the minimum time to situation awareness in transfer of control scenarios. *Transportation Research Record: Journal of the Transportation Research Board 2663*, 16 (2017), 126–133.

[3] Aine, Sandip, Chakrabarti, PP, and Kumar, Rajeev. AWA$*$: A window constrained anytime heuristic search algorithm. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence* (2007), pp. 2250–2255.

[4] Akgun, Baris, and Stilman, Mike. Sampling heuristics for optimal motion planning in high dimensions. In *Proceedings of the International Conference on Intelligent Robots and Systems* (2011), IEEE.

[5] Alexander, George, Raja, Anita, and Musliner, David J. Controlling deliberation in coordinators. In *Metareasoning: Thinking about Thinking*, M. Cox and A. Raja, Eds. MIT Press, Cambridge, MA, USA, 2011.

[6] Alterovitz, Ron, Koenig, Sven, and Likhachev, Maxim. Robot planning in the real world. *AI Magazine* (2016).

[7] Amodei, Dario, Olah, Chris, Steinhardt, Jacob, Christiano, Paul, Schulman, John, and Mané, Dan. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565* (2016).

[8] Anderson, Michael L, Oates, Tim, Chong, Waiyian, and Perlis, Don. The metacognitive loop: Enhancing reinforcement learning with metacognitive monitoring and control for improved perturbation tolerance. *Journal of Experimental and Theoretical Artificial Intelligence 18*, 3 (2006), 387–411.

[9] Anderson, Michael L, and Perlis, Donald R. Logic, self-awareness and self-improvement: The metacognitive loop and the problem of brittleness. *Journal of Logic and Computation 15*, 1 (2005), 21–40.

[10] Ansótegui, Carlos, Sellmann, Meinolf, and Tierney, Kevin. A gender-based genetic algorithm for the automatic configuration of algorithms. In *Proceedings of the Fifteenth International Conference on Principles and Practice of Constraint Programming* (2009), Springer, pp. 142–157.

[11] Arcos, Josep LLuís, Mülâyim, Oguz, and Leake, David B. Using introspective reasoning to improve CBR system performance. In *Metareasoning: Thinking about Thinking*, M. Cox and A. Raja, Eds. MIT Press, Cambridge, MA, USA, 2011.

[12] Arkin, Ronald C. Governing lethal behavior: Embedding ethics in a hybrid deliberative/reactive robot architecture. In *Proceedings of the Third ACM/IEEE International Conference on Human-Robot Interaction* (2008).

[13] Barto, Andrew G., and Mahadevan, Sridhar. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems* (2003).

[14] Bartz-Beielstein, Thomas, Lasarczyk, Christian WG, and Preuss, Mike. Sequential parameter optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation* (2005), vol. 1, IEEE, pp. 773–780.

[15] Bartz-Beielstein, Thomas, and Markon, Sandor. Tuning search algorithms for real-world applications: A regression tree based approach. In *Proceedings of the IEEE Congress on Evolutionary Computation* (2004), pp. 1111–1118.

[16] Basich, Connor, Svegliato, Justin, Beach, Allyson, Zilberstein, Shlomo, Wray, Kyle Hollins, and Witwicki, Stefan J. Improving competence via iterative state space refinement. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems* (2021).

[17] Basich, Connor, Svegliato, Justin, Wray, Kyle Hollins, Witwicki, Stefan, Biswas, Joydeep, and Zilberstein, Shlomo. Learning to optimize autonomy in competence-aware systems. In *Proceedings of the Nineteenth International Conference on Autonomous Agents and Multiagent Systems* (2020).

[18] Basich, Connor, Svegliato, Justin, Zilberstein, Shlomo, Wray, Kyle Hollins, and Witwicki, Stefan J. Improving competence for reliable autonomy. In *Proceedings of the ECAI Workshop on Agents and Robots for Reliable Engineered Autonomy* (2020).

[19] Belghith, Khaled, Kabanza, Froduald, Hartman, Leo, and Nkambou, Roger. Anytime dynamic path-planning with flexible probabilistic roadmaps. In *Proceedings of the IEEE International Conference on Robotics and Automation* (2006), IEEE, pp. 2372–2377.

[20] Bellman, Richard. Dynamic programming. *Science* (1966).

[21] Bertsekas, Dimitri P, and Tsitsiklis, John N. An analysis of stochastic shortest path problems. *Mathematics of Operations Research 16*, 3 (1991).

[22] Bhatia, Abhinav, Svegliato, Justin, and Zilberstein, Shlomo. On the benefits of randomly adjusting anytime weighted A*. In *Proceedings of the Fourteenth Symposium on Combinatorial Search* (2021).

[23] Bhatia, Abhinav, Svegliato, Justin, and Zilberstein, Shlomo. Tuning the hyperparameters of anytime planning: A deep reinforcement learning approach. In *Proceedings of the ICAPS Workshop on Heuristics and Search for Domain-Independent Planning* (2021).

[24] Bierwirth, Christian. A generalized permutation approach to job shop scheduling with genetic algorithms. *Operations Research Spectrum 17*, 2-3 (1995), 87–92.

[25] Birattari, Mauro, Stützle, Thomas, Paquete, Luis, and Varrentrapp, Klaus. A racing algorithm for configuring metaheuristics. In *Proceedings of the Fourth Genetic and Evolutionary Computation Conference* (2002), vol. 2.

[26] Birattari, Mauro, Yuan, Zhi, Balaprakash, Prasanna, and Stützle, Thomas. F-race and iterated F-race: An overview. *Experimental Methods for the Analysis of Optimization Algorithms* (2010), 311–336.

[27] Boddy, Mark, and Dean, Thomas L. Deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence 67*, 2 (1994), 245–285.

[28] Breese, John S., and Horvitz, Eric J. Ideal reformulation of belief networks. In *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence* (1991), pp. 129–143.

[29] Burkard, Rainer E., Karisch, Stefan E., and Rendl, Franz. QAPLIB–A quadratic assignment problem library. *Journal of Global Optimization 10*, 4 (1997), 391–403.

[30] Burns, Ethan, Ruml, Wheeler, and Do, Minh Binh. Heuristic search when time matters. *Journal of Artificial Intelligence Research 47* (2013), 697–740.

[31] Casper, Jennifer, and Murphy, Robin R. Human-robot interactions during the robot-assisted urban search and rescue response at the World Trade Center. *IEEE Transactions on Systems, Man, and Cybernetics 33*, 3 (2003).

[32] Cassandra, Anthony, Littman, Michael L., and Zhang, Nevin L. Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence* (1997), Morgan Kaufmann Publishers Inc.

[33] Chen, Sheng-Lei, and Wei, Yan-Mei. Least-squares SARSA(Lambda) algorithms for reinforcement learning. In *Proceedings of the Fourth International Conference on Natural Computation* (2008), pp. 632–636.

[34] Cox, Michael, Alavi, Zohreh, Dannenhauer, Dustin, Eyorokon, Vahid, Munoz-Avila, Hector, and Perlis, Don. MIDCA: A metacognitive, integrated dual-cycle architecture for self-regulated autonomy. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence* (2016), vol. 30.

[35] Cserna, Bence, Ruml, Wheeler, and Frank, Jeremy. Planning time to think: Metareasoning for on-line planning with durative actions. In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling* (2017).

[36] Davis, Lawrence. Job shop scheduling with genetic algorithms. In *Proceedings of the First International Conference on Genetic Algorithms* (1985), pp. 136–140.

[37] Dean, Thomas L., and Boddy, Mark S. An analysis of time-dependent planning. In *Proceedings of the Seventh AAAI National Conference on Artificial Intelligence* (1988), pp. 49–54.

[38] Dearden, Richard, Willeke, Thomas, Simmons, Reid, Verma, Vandi, Hutter, Frank, and Thrun, Sebastian. Real-time fault detection and situational awareness for rovers: Report on the mars technology program task. In *Proceedings of the IEEE Aerospace Conference* (2004), vol. 2, IEEE, pp. 826–840.

[39] Della Croce, Federico, Tadei, Roberto, and Volta, Giuseppe. A genetic algorithm for the job shop problem. *Computers & Operations Research 22*, 1 (1995), 15–24.

[40] Dietterich, Thomas G. Ensemble methods in machine learning. In *Proceedings of the First International Workshop on Multiple Classifier Systems* (2000), pp. 1–15.

[41] Epstein, Susan L, and Petrovic, Smiljana. Learning expertise with bounded rationality and self-awareness. In *Metareasoning: Thinking about Thinking*, M. Cox and A. Raja, Eds. MIT Press, Cambridge, MA, USA, 2011.

[42] Fern, Alan, and Givan, Robert. Online ensemble learning: An empirical study. *Machine Learning 53*, 1-2 (2003), 71–109.

[43] Ferrando, Angelo, Dennis, Louise A., Ancona, Davide, Fisher, Michael, and Mascardi, Viviana. Recognising assumption violations in autonomous systems verification. In *Proceedings of the Seventeenth International Conference on Autonomous Agents and Multiagent Systems* (Richland, SC, 2018), International Foundation for Autonomous Agents and Multiagent Systems, pp. 1933–1935.

[44] Geißer, Florian, Speck, David, and Keller, Thomas. An analysis of the probabilistic track of the ipc 2018. In *Proceedings of the ICAPS 2019 Workshop on the International Planning Competition* (2019), pp. 27–35.

[45] Gigerenzer, Gerd. *Adaptive Thinking: Rationality in the Real World.* Oxford University Press, Oxford, UK, 2000.

[46] Gigerenzer, Gerd, and Todd, Peter M. *Simple Heuristics That Make Us Smart.* Oxford University Press, Oxford, UK, 1999.

[47] Gilmore, Paul C. Optimal and suboptimal algorithms for the quadratic assignment problem. *Journal of the Society for Industrial and Applied Mathematics 10*, 2 (1962), 305–313.

[48] Goel, Ashok K, and Jones, Joshua. Metareasoning for self-adaptation in intelligent agents. In *Metareasoning: Thinking about Thinking*, M. Cox and A. Raja, Eds. MIT Press, Cambridge, MA, USA, 2011.

[49] Goel, Puneet, Dedeoglu, Göksel, Roumeliotis, Stergios I, and Sukhatme, Gaurav S. Fault detection and identification in a mobile robot using multiple model estimation and neural network. In *Proceedings of the IEEE International Conference on Robotics and Automation* (2000), vol. 3, IEEE, pp. 2302–2309.

[50] Gomes, Carla P., and Selman, Bart. Algorithm portfolios. *Artificial Intelligence 126*, 1-2 (2001), 43–62.

[51] Gomoluch, Pawel, Alrajeh, Dalal, Russo, Alessandra, and Bucchiarone, Antonio. Learning neural search policies for classical planning. In *Proceedings of the International Conference on Automated Planning and Scheduling* (2020), vol. 30, pp. 522–530.

[52] Good, Irving J. Twenty-seven principles of rationality. In *Foundations of Statistical Inference*, P. Godambe and D. A. Sprott, Eds. Holt, Rinehart, Winston, Toronto, Canada, 1971, pp. 108–141.

[53] Goodrich, Michael A., Morse, Bryan S., Gerhardt, Damon, Cooper, Joseph L., Quigley, Morgan, Adams, Julie A., and Humphrey, Curtis. Supporting wilderness search and rescue using a camera-equipped mini UAV. *Journal of Field Robotics* (2008).

[54] Gu, Rong, Marinescu, Raluca, Seceleanu, Cristina, and Lundqvist, Kristina. Formal verification of an autonomous wheel loader by model checking. In *Proceedings of the Sixth Conference on Formal Methods in Software Engineering* (New York, NY, USA, 2018), FormaliSE '18, ACM, pp. 74–83.

[55] Hadfield-Menell, Dylan, Milli, Smitha, Abbeel, Pieter, Russell, Stuart J, and Dragan, Anca. Inverse reward design. *Advances in Neural Information Processing Systems 30* (2017), 6765–6774.

[56] Hadfield-Menell, Dylan, Russell, Stuart J, Abbeel, Pieter, and Dragan, Anca. Cooperative inverse reinforcement learning. *Proceedings of the Thirtieth Conference on Neural Information Processing Systems 29* (2016).

[57] Hansen, Eric A. Indefinite-horizon POMDPs with action-based termination. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence* (2007), pp. 1237–1242.

[58] Hansen, Eric A., and Zhou, Rong. Anytime heuristic search. *Journal of Artificial Intelligence Research 28* (2007), 267–297.

[59] Hansen, Eric A., and Zilberstein, Shlomo. LAO$*$: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence 129*, 1-2 (2001), 35–62.

[60] Hansen, Eric A., and Zilberstein, Shlomo. Monitoring and control of anytime algorithms: A dynamic programming approach. *Artificial Intelligence 126*, 1-2 (2001), 139–157.

[61] Hansen, Eric A., Zilberstein, Shlomo, and Danilchenko, Victor A. Anytime heuristic search: First results. Tech. Rep. 97-50, Computer Science Department, University of Massachussetts Amherst, 1997.

[62] Hart, Peter E, Nilsson, Nils J, and Raphael, Bertram. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics 4*, 2 (1968), 100–107.

[63] Hay, Nicholas, Russell, Stuart, Tolpin, David, and Shimony, Solomon Eyal. Selecting computations: Theory and applications. *arXiv preprint arXiv:1408.2048* (2014).

[64] Horvitz, Eric. Reasoning under varying and uncertain resource constraints. In *Proceedings of the Seventh AAAI National Conference on Artificial Intelligence* (1988), pp. 111–116.

[65] Horvitz, Eric, and Rutledge, Geoffrey. Time-dependent utility and action under uncertainty. In *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence* (1991), pp. 151–158.

[66] Horvitz, Eric J. Reasoning about beliefs and actions under computational resource constraints. In *Proceedings of the Third Workshop on Uncertainty in Artificial Intelligence* (1987).

[67] Horvitz, Eric J. *Computation and action under bounded resources.* PhD thesis, Stanford University, CA, 1990.

[68] Huang, Ling, Jia, Jinzhu, Yu, Bin, Chun, Byung-Gon, Maniatis, Petros, and Naik, Mayur. Predicting execution time of computer programs using sparse polynomial regression. In *Proceedings of the Twenty-Fourth Conference on Neural Information Processing Systems* (2010), pp. 883–891.

[69] Huberman, Bernardo A., Lukose, Rajan M., and Hogg, Tad. An economics approach to hard computational problems. *Science 275*, 5296 (1997), 51–54.

[70] Hutter, Frank, Hoos, Holger H, and Leyton-Brown, Kevin. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the Fifth International Conference on Learning and Intelligent Optimization* (2011), Springer, pp. 507–523.

[71] Hutter, Frank, Hoos, Holger H, Leyton-Brown, Kevin, and Murphy, Kevin. Time-bounded sequential parameter optimization. In *Proceedings of the Fourth International Conference on Learning and Intelligent Optimization* (2010), Springer, pp. 281–298.

[72] Hutter, Frank, Hoos, Holger H, Leyton-Brown, Kevin, and Murphy, Kevin P. An experimental investigation of model-based parameter optimisation: Spo and beyond. In *Proceedings of the Eleventh Annual Conference on Genetic and Evolutionary Computation* (2009), pp. 271–278.

[73] Hutter, Frank, Hoos, Holger H, Leyton-Brown, Kevin, and Stützle, Thomas. Paramils: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research 36* (2009), 267–306.

[74] Hutter, Frank, Hoos, Holger H, and Stützle, Thomas. Automatic algorithm configuration based on local search. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence* (2007), vol. 7, pp. 1152–1157.

[75] Hutter, Frank, Xu, Lin, Hoos, Holger H., and Leyton-Brown, Kevin. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence 206* (2014), 79–111.

[76] Kaelbling, Leslie Pack, Littman, Michael L., and Cassandra, Anthony R. Planning and acting in partially observable stochastic domains. *Journal of Artificial Intelligence Research* (1998).

[77] Karaman, Sertac, Walter, Matthew R., Perez, Alejandro, Frazzoli, Emilio, and Teller, Seth. Anytime motion planning using the RRT∗. In *Proceedings of the IEEE International Conference on Robotics and Automation* (2011), IEEE, pp. 1478–1483.

[78] Karayev, Sergey, Fritz, Mario, and Darrell, Trevor. Anytime recognition of objects and scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2014), pp. 572–579.

[79] Kasenberg, Daniel, and Scheutz, Matthias. Norm conflict resolution in stochastic domains. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence* (2018), vol. 32.

[80] Kiesel, Scott, Burns, Ethan, and Ruml, Wheeler. Abstraction-guided sampling for motion planning. In *Proceedings of the Fifth Annual Symposium on Combinatorial Search* (2012).

[81] Kim, H.J., Jordan, Michael I., Sastry, Shankar, and Ng, Andrew Y. Autonomous helicopter flight via reinforcement learning. In *Proceedings of the Conference on Neural Information Processing Systems* (2004), pp. 799–806.

[82] Kolobov, Andrey, Mausam, and Weld, Daniel S. A theory of goal-oriented mdps with dead ends. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence* (2012).

[83] Konidaris, George, Osentoski, Sarah, and Thomas, Philip S. Value function approximation in reinforcement learning using the Fourier basis. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence* (2011), vol. 6, p. 7.

[84] Kotseruba, Iuliia, and Tsotsos, John K. 40 years of cognitive architectures: Core cognitive abilities and practical applications. *Artificial Intelligence Review 53*, 1 (2020), 17–94.

[85] Kotthoff, Lars. Algorithm selection for combinatorial search problems: A survey. In *Data Mining and Constraint Programming*. Springer, 2016, pp. 149–190.

[86] Kumar, Akshat, and Zilberstein, Shlomo. Anytime planning for decentralized POMDPs using expectation maximization. In *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence* (2010), pp. 294–301.

[87] Kwak, Jun-young, Varakantham, Pradeep, Maheswaran, Rajiv, Tambe, Milind, Jazizadeh, Farrokh, Kavulya, Geoffrey, Klein, Laura, Becerik-Gerber, Burcin, Hayes, Timothy, and Wood, Wendy. SAVES: A sustainable multiagent application to conserve building energy considering occupants. In *Proceedings of the Eleventh International Conference on Autonomous Agents and Multiagent Systems* (2012), pp. 21–28.

[88] Leike, Jan, Krueger, David, Everitt, Tom, Martic, Miljan, Maini, Vishal, and Legg, Shane. Scalable agent alignment via reward modeling: A research direction. *arXiv preprint arXiv:1811.07871* (2018).

[89] Leyton-Brown, Kevin, Nudelman, Eugene, Andrew, Galen, McFadden, Jim, and Shoham, Yoav. Boosting as a metaphor for algorithm design. In *Proceedings of the Ninth International Conference on Principles and Practice of Constraint Programming* (2003), pp. 899–903.

[90] Lieder, Falk, Plunkett, Dillon, Hamrick, Jessica B., Russell, Stuart J., Hay, Nicholas J., and Griffiths, Thomas L. Algorithm selection by rational metareasoning as a model of human strategy. In *Proceedings of the Conference on Neural Information Processing Systems* (2014).

[91] Likhachev, Maxim, Gordon, Geoff, and Thrun, Sebastian. ARA*: Anytime A* with provable bounds on sub-optimality. In *Proceedings of the Conference on Neural Information Processing Systems* (2004), pp. 767–774.

[92] Lin, Christopher H, Kolobov, Andrey, Kamar, Ece, and Horvitz, Eric. Metareasoning for planning under uncertainty. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence* (2015).

[93] Lin, Shen, and Kernighan, Brian W. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research 21*, 2 (1973), 498–516.

[94] Littman, Michael L. The Witness algorithm: Solving partially observable Markov decision processes. Tech. rep., Brown University, 1994.

[95] Luna, R., Şucan, I. A., Moll, M., and Kavraki, L. E. Anytime solution optimization for sampling-based motion planning. In *Proceedings of the IEEE International Conference on Robotics and Automation* (2013), pp. 5068–5074.

[96] Maei, Hamid Reza, Szepesvári, Csaba, Bhatnagar, Shalabh, and Sutton, Richard S. Toward off-policy learning control with function approximation. In *Proceedings of the Twenty-Seventh International Conference on Machine Learning* (2010).

[97] Manne, Alan S. Linear programming and sequential decisions. *Management Science* (1960).

[98] Mendoza, Juan Pablo, Veloso, Manuela, and Simmons, Reid. Mobile robot fault detection based on redundant information statistics. In *Proceedings of the IROS Workshop on Safety in Human-Robot Coexistence and Interaction* (2012), vol. 945, Citeseer, pp. 7–11.

[99] Misevičius, Alfonsas. A modified simulated annealing algorithm for the quadratic assignment problem. *Informatica 14*, 4 (2003), 497–514.

[100] Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, et al. Human-level control through deep reinforcement learning. *Nature 518*, 7540 (2015), 529.

[101] Murdock, J William, and Goel, Ashok K. Meta-case-based reasoning: Self-improvement through self-understanding. *Journal of Experimental & Theoretical Artificial Intelligence 20*, 1 (2008), 1–36.

[102] Nakano, Ryohei, and Yamada, Takeshi. Conventional genetic algorithm for job shop problems. In *Proceedings of the Fourth International Conference on Genetic Algorithms* (1991), pp. 474–479.

[103] Nashed, Samer B, Svegliato, Justin, Brucato, Matteo, Basich, Connor, Grupen, Roderic A, and Zilberstein, Shlomo. Solving Markov decision processes with partial state abstractions. In *Proceedings of the IEEE International Conference on Robotics and Automation* (2021).

[104] Nashed, Samer B, Svegliato, Justin, and Zilberstein, Shlomo. Ethically compliant planning within moral communities. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society* (2021).

[105] Otten, Lars, and Dechter, Rina. Anytime and/or depth-first search for combinatorial optimization. *AI Communications 25*, 3 (2012), 211–227.

[106] Parr, Shane, Khatri, Ishan, Svegliato, Justin, and Zilberstein, Shlomo. Agent-aware state estimation: Effective traffic light classification for autonomous vehicles. In *Proceedings of the ICRA Workshop on Sensing, Estimating and Understanding the Dynamic World* (2020).

[107] Parr, Shane, Khatri, Ishan, Svegliato, Justin, and Zilberstein, Shlomo. Agent-aware state estimation in autonomous vehicles. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems* (2021).

[108] Paul, C. J., Acharya, Anurag, Black, Bryan, and Strosnider, Jay K. Reducing problem-solving variance to improve predictability. *Communications of the ACM 34*, 8 (1991), 80–93.

[109] Petrik, Marek, and Zilberstein, Shlomo. Learning parallel portfolios of algorithms. *Annals of Mathematics and Artificial Intelligence (AMAI) 48*, 1-2 (2006), 85–106.

[110] Pineau, Joelle, Gordon, Geoff, and Thrun, Sebastian. Point-based Value Iteration: An anytime algorithm for POMDPs. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence* (2003), pp. 1025–1032.

[111] Pineda, Luis, Takahashi, Takeshi, Jung, Hee-Tae, Zilberstein, Shlomo, and Grupen, Roderic. Continual planning for search and rescue robots. In *Proceedings of the IEEE-RAS Fifteenth International Conference on Humanoid Robots* (Seoul, Korea, 2015).

[112] Pineda, Luis, and Zilberstein, Shlomo. Planning under uncertainty using reduced models: Revisiting determinization. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling* (Portsmouth, New Hampshire, 2014), pp. 217–225.

[113] Pineda, Luis Enrique, and Zilberstein, Shlomo. Probabilistic planning with reduced models. *Journal of Artificial Intelligence Research 65* (2019), 271–306.

[114] Raja, Anita, Alexander, George, Lesser, Victor R, and Krainin, Michael. Coordinating agents' metalevel control. In *Metareasoning: Thinking about Thinking*, M. Cox and A. Raja, Eds. MIT Press, Cambridge, MA, USA, 2011.

[115] Ramos, Fabio Tozeto, and Cozman, Fabio Gagliardi. Anytime anyspace probabilistic inference. *International Journal of Approximate Reasoning 38*, 1 (2005), 53–80.

[116] Richter, Silvia, Thayer, Jordan Tyler, and Ruml, Wheeler. The joy of forgetting: Faster anytime search via restarting. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling* (2010), pp. 137–144.

[117] Richtsfeld, Andreas, Zillich, Michael, and Vincze, Markus. Anytime perceptual grouping of 2D features into 3D basic shapes. In *Proceedings of the Ninth International Conference on Computer Vision Systems* (2013), pp. 73–82.

[118] Robertson, Paul, and Laddaga, Robert. Metareasoning-based self-adaptive tracking. In *Proceedings of the Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshop* (2010), IEEE, pp. 275–281.

[119] Roumeliotis, Stergios I., Sukhatme, Gaurav S., and Bekey, George A. Fault detection and identification in a mobile robot using multiple-model estimation. In *Proceedings of the IEEE International Conference on Automation and Robotics* (1998), vol. 3, IEEE, pp. 2223–2228.

[120] Rubinstein, Zachary B, Smith, Stephen F, and Zimmerman, Terry L. The role of metareasoning in achieving effective multiagent coordination. In *Metareasoning: Thinking about Thinking*, M. Cox and A. Raja, Eds. MIT Press, Cambridge, MA, USA, 2011.

[121] Russell, Stuart, and Wefald, Eric. Principles of metareasoning. *Artificial Intelligence 49* (1991), 361–395.

[122] Russell, Stuart J., Subramanian, Devika, and Parr, Ron. Provably bounded optimal agents. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence* (1993), pp. 338–344.

[123] Russell, Stuart J., and Wefald, Eric H. *Do the Right thing: Studies in Limited Rationality.* MIT Press, Cambridge, MA, 1991.

[124] Saisubramanian, Sandhya, Kamar, Ece, and Zilberstein, Shlomo. A multi-objective approach to mitigate negative side effects. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence* (2020).

[125] Saisubramanian, Sandhya, Roberts, Shannon C, and Zilberstein, Shlomo. Understanding user attitudes towards negative side effects of AI systems. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (2021).

[126] Saisubramanian, Sandhya, and Zilberstein, Shlomo. Safe reduced models for probabilistic planning. In *Proceedings of the ICML/IJCAI/AAMAS Workshop on Planning and Learning* (Stockholm, Sweden, 2018).

[127] Saisubramanian, Sandhya, and Zilberstein, Shlomo. Adaptive outcome selection for planning with reduced models. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems* (Macau, China, 2019).

[128] Saisubramanian, Sandhya, and Zilberstein, Shlomo. Minimizing the negative side effects of planning with reduced models. In *Proceedings of the AAAI Workshop on Artificial Intelligence Safety* (Honolulu, Hawaii, 2019).

[129] Saisubramanian, Sandhya, Zilberstein, Shlomo, and Shenoy, Prashant J. Planning using a portfolio of reduced models. In *Proceedings of the Seventeenth International Conference on Autonomous Agents and Multiagent Systems* (Stockholm, Sweden, 2018), pp. 2057–2059.

[130] Shim, Jaeeun, Arkin, Ronald, and Pettinatti, Michael. An intervening ethical governor for a robot mediator in patient-caregiver relationship. In *IEEE International Conference on Robotics and Automation* (2017).

[131] Silver, David, Hubert, Thomas, Schrittwieser, Julian, Antonoglou, Ioannis, Lai, Matthew, Guez, Arthur, Lanctot, Marc, Sifre, Laurent, Kumaran, Dharshan, Graepel, Thore, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science 362*, 6419 (2018), 1140–1144.

[132] Simon, Herbert A. *Administrative Behavior.* Macmillan, New York, NY, USA, 1947.

[133] Simon, Herbert A. *Models of Bounded Rationality.* MIT Press, Cambridge, MA, 1982.

[134] Sondik, Edward Jay. The optimal control of partially observable Markov processes. Tech. rep., Stanford University, 1971.

[135] Spaan, Matthijs TJ, and Vlassis, Nikos. Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research 24* (2005), 195–220.

[136] Speck, David, Biedenkapp, André, Hutter, Frank, Mattmüller, Robert, and Lindauer, Marius. Learning heuristic selection with dynamic algorithm configuration. In *Proceedings of the International Conference on Automated Planning and Scheduling* (2021), vol. 31, pp. 597–605.

[137] Srivihok, Anongnart, and Sukonmanee, Pisit. E-commerce intelligent agent: Personalization travel support agent using Q-learning. In *Proceedings of the Seventh International Conference on Electronic Commerce* (2005), pp. 287–292.

[138] Sun, Xiaoxun, Druzdzel, Marek J., and Yuan, Changhe. Dynamic weighting A* search-based MAP algorithm for Bayesian networks. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence* (2007), pp. 2385–2390.

[139] Sutton, Richard S. Learning to predict by the methods of temporal differences. *Machine Learning 3*, 1 (1995), 9–44.

[140] Sutton, Richard S, and Barto, Andrew G. *Reinforcement learning: An introduction.* MIT press, 2018.

[141] Sutton, Richard S., Precup, Doina, and Singh, Satinder. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence 112*, 1-2 (1999), 181–211.

[142] Svegliato, Justin, Basich, Connor, Saisubramanian, Sandhya, and Zilberstein, Shlomo. Using metareasoning to maintain and restore safety for reliably autonomy. In *Proceedings of the IJCAI Workshop on Robust and Reliable Autonomy in the Wild* (2021).

[143] Svegliato, Justin, Nashed, Samer, and Zilberstein, Shlomo. Ethically compliant planning in moral autonomous systems. In *Proceedings of the IJCAI Workshop on AI Safety* (2020).

[144] Svegliato, Justin, Nashed, Samer, and Zilberstein, Shlomo. An integrated approach to moral autonomous systems. In *Proceedings of the Twenty-Forth European Conference on Artificial Intelligence* (2020).

[145] Svegliato, Justin, Nashed, Samer B, and Zilberstein, Shlomo. Ethically compliant sequential decision making. In *Proceedings of the Thirty-Fifth International Conference on Artificial Intelligence* (2021), AAAI.

[146] Svegliato, Justin, Sharma, Prakhar, and Zilberstein, Shlomo. A model-free approach to meta-level control of anytime algorithms. In *Proceedings of the IEEE International Conference on Robotics and Automation* (Paris, France, 2020).

[147] Svegliato, Justin, Witty, Sam, Houmansadr, Amir, and Zilberstein, Shlomo. Belief space planning for automated malware defense. In *Proceedings of the IJCAI/ECAI Workshop on AI for Internet of Things* (2018).

[148] Svegliato, Justin, Wray, Kyle Hollins, Witwicki, Stefan J, Biswas, Joydeep, and Zilberstein, Shlomo. Belief space metareasoning for exception recovery. In *Proceedings of the 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2019), IEEE, pp. 1224–1229.

[149] Svegliato, Justin, Wray, Kyle Hollins, and Zilberstein, Shlomo. Meta-level control of anytime algorithms with online performance prediction. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence* (2018).

[150] Svegliato, Justin, and Zilberstein, Shlomo. Adaptive metareasoning for bounded rational agents. In *Proceedings of the IJCAI Workshop on Architectures and Evaluation for Generality, Autonomy and Progress in AI* (2018).

[151] Tan, Ying, Liu, Wei, and Qiu, Qinru. Adaptive power management using RL. In *Proceedings of the International Conference on Computer-Aided Design* (2009), pp. 461–467.

[152] Tesauro, Gerald. Temporal difference learning and TD-gammon. *Communications of the ACM 38*, 3 (1995), 58–68.

[153] Thayer, Jordan, and Ruml, Wheeler. Using distance estimates in heuristic search. In *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling* (2009), pp. 382–385.

[154] Thayer, Jordan, and Ruml, Wheeler. Anytime heuristic search: Frameworks and algorithms. In *Proceedings of the Third Annual Symposium on Combinatorial Search* (2010), pp. 121–128.

[155] Thrun, Sebastian, Hahnel, Dirk, Ferguson, David, Montemerlo, Michael, Triebel, Rudolph, Burgard, Wolfram, Baker, Christopher, Omohundro, Zachary, Thayer, Scott, and Whittaker, William. A system for volumetric robotic mapping of abandoned mines. In *Proceedings of the IEEE International Conference on Robotics and Automation* (2003), pp. 4270–4275.

[156] Urmson, Chris, and Simmons, Reid. Approaches for heuristically biasing RRT growth. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems* (2003), vol. 2, IEEE, pp. 1178–1183.

[157] Van Den Berg, Jur, Ferguson, Dave, and Kuffner, James. Anytime path planning and replanning in dynamic environments. In *Proceedings of the IEEE International Conference on Robotics and Automation* (2006), Ieee, pp. 2366–2371.

[158] Verma, Vandi, Gordon, Geoff, Simmons, Reid, and Thrun, Sebastian. Particle filters for fault diagnosis. *IEEE Robotics and Automation Magazine 11* (2004), 56–64.

[159] Vinyals, Oriol, Babuschkin, Igor, Czarnecki, Wojciech M, Mathieu, Michaël, Dudzik, Andrew, Chung, Junyoung, Choi, David H, Powell, Richard, Ewalds, Timo, Georgiev, Petko, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature 575*, 7782 (2019), 350–354.

[160] Wagner, Thomas, Garvey, Alan, and Lesser, Victor R. Criteria-directed task scheduling. *International Journal of Approximate Reasoning 19*, 1-2 (1998), 91–118.

[161] Wallace, Richard J., and Freuder, Eugene C. Anytime algorithms for constraint satisfaction and SAT problems. *ACM SIGART Bulletin 7*, 2 (1996), 7–10.

[162] Watkins, Christopher J. C. H., and Dayan, Peter. Q-learning. *Machine Learning 8*, 3 (1992), 279–292.

[163] Wellman, Michael P. *Formulation of Tradeoffs in Planning under Uncertainty.* Pitman, London, UK, 1990.

[164] Wilhelm, Mickey R., and Ward, Thomas L. Solving quadratic assignment problems by “simulated annealing”. *IIE Transactions 19*, 1 (1987), 107–119.

[165] Williams, Brian C., Ingham, Michel D., Chung, Seung H., and Elliott, Paul H. Model-based programming of intelligent embedded systems and robotic space explorers. *Proceedings of the IEEE* (2003).

[166] Wilt, Christopher, and Ruml, Wheeler. When does weighted A* fail? In *Proceedings of the Fifth Annual Symposium on Combinatorial Search* (2012), pp. 137–144.

[167] Wray, Kyle Hollins, Pineda, Luis, and Zilberstein, Shlomo. Hierarchical approach to transfer of control in semi-autonomous systems. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence* (2016).

[168] Wray, Kyle Hollins, Ruiken, Dirk, Grupen, Roderic A., and Zilberstein, Shlomo. Log-space harmonic function path planning. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems* (2016), pp. 1511–1516.

[169] Wray, Kyle Hollins, Witwicki, Stefan J., and Zilberstein, Shlomo. Online decision-making for scalable autonomous systems. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence* (2017).

[170] Wright, Timothy J., Agrawal, Ravi, Samuel, Siby, Wang, Yuhua, Zilberstein, Shlomo, and Fisher, Donald L. Effects of alert cue specificity on situation awareness in transfer of control in level 3 automation. *Transportation Research Record: Journal of the Transportation Research Board 2663*, 16 (2017), 27–33.

[171] Xiao, Xuesu, Liu, Bo, Warnell, Garrett, Fink, Jonathan, and Stone, Peter. APPLD: Adaptive planner parameter learning from demonstration. *IEEE Robotics and Automation Letters 5*, 3 (2020), 4541–4547.

[172] Xu, Lin, Hutter, Frank, Hoos, Holger H., and Leyton-Brown, Kevin. Satzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research 32* (2008), 565–606.

[173] Zhang, Junzhe, and Bareinboim, Elias. Characterizing the limits of autonomous systems. In *Proceedings of the Seventeenth International Conference on Autonomous Agents and Multiagent Systems* (2018).

[174] Zhang, Shun, Durfee, Edmund H, and Singh, Satinder P. Minimax-regret querying on side effects for safe optimality in factored Markov decision processes. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence* (2018), pp. 4867–4873.

[175] Zilberstein, Shlomo. *Operational Rationality through Compilation of Anytime Algorithms*. PhD thesis, Computer Science Division, University of California Berkeley, 1993.

[176] Zilberstein, Shlomo. Resource-bounded sensing and planning in autonomous systems. *Autonomous Robots 3*, 1 (1996), 31–48.

[177] Zilberstein, Shlomo. Using anytime algorithms in intelligent systems. *AI Magazine 17*, 3 (1996), 73.

[178] Zilberstein, Shlomo. Metareasoning and bounded rationality. In *Metareasoning: Thinking about Thinking*, M. Cox and A. Raja, Eds. MIT Press, Cambridge, MA, USA, 2011.

[179] Zilberstein, Shlomo, Charpillet, Francois, and Chassaing, Philippe. Optimal sequencing of contract algorithms. *Annals of Mathematics and Artificial Intelligence 39*, 1-2 (2003), 1–18.

[180] Zilberstein, Shlomo, and Mouaddib, Abdel-Illah. Optimal scheduling of progressive processing tasks. *International Journal of Approximate Reasoning 25*, 3 (2000), 169–186.

[181] Zilberstein, Shlomo, and Russell, Stuart J. Anytime sensing, planning and action: A practical model for robot control. In *Proceedings of the International Joint Conference on Artificial Intelligence* (1993), vol. 93, pp. 1402–1407.

[182] Zilberstein, Shlomo, and Russell, Stuart J. Approximate reasoning using anytime algorithms. In *Imprecise and Approximate Computation*, S. Natarajan, Ed. Springer, 1995, pp. 43–62.

[183] Zilberstein, Shlomo, Washington, Richard, Bernstein, Daniel S., and Mouaddib, Abdel-Illah. Decision-theoretic control of planetary rovers. In *Revised Papers from the International Seminar on Advances in Plan-Based Control of Robotic Agents* (London, UK, 2002), Springer-Verlag.