July 2019

Abstractions in Reasoning for Long-Term Autonomy

Kyle Hollins Wray
University of Massachusetts Amherst

ABSTRACTS IN REASONING FOR LONG-TERM AUTONOMY

A Dissertation Presented

by

KYLE HOLLINS WRAY

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

DOCTOR OF PHILOSOPHY

May 2019

College of Information and Computer Sciences

ABSTRACTIONS IN REASONING FOR LONG-TERM AUTONOMY

A Dissertation Presented

by

KYLE HOLLINS WRAY

Approved as to style and content by:

Shlomo Zilberstein, Chair

Joydeep Biswas, Member

Roderic A. Grupen, Member

Benjamin M. Marlin, Member

Weibo Gong, Member

Stefan J. Witwicki, Member

James Allan, Chair of the Faculty
College of Information and Computer Sciences

ACKNOWLEDGMENTS

It is difficult for me to convey my gratitude to everyone that has contributed to my experience here in Amherst. I am grateful for the amazing experience it has been. I will always cherish the fond memories shared among friends, colleagues, and mentors. With much pleasure, I take this brief opportunity to thank many of the people who have helped shape my journey.

Shlomo Zilberstein, my advisor, has been a constant source of knowledge and inspiration. He has always encouraged me to pursue my own passions in research and has given me the freedom to do so, providing direction when it was needed. Through his guidance, conducting research has become an endlessly enjoyable and productive experience. I could not have asked for a better mentor. Thank you, Shlomo, for these great years.

I would also like to thank my committee members, who have all been very helpful. Joydeep Biswas provided insightful ideas on many occasions. He encouraged a high standard of work, which I always appreciate and respect. Rod Grupen has continually been a supportive teacher and collaborator who welcomed me into his lab and helped introduce me to robotics. Ben Marlin has been a fantastic teacher and mentor. I appreciated his thoughts on the graphical model presented in this dissertation. Weibo Gong provided great feedback throughout each stage of the dissertation process.

The team at the Nissan Research Center in Silicon Valley has been among the most supportive and fun groups I have had the privilege to work with. Stefan Witwicki, who also served on my committee, has been an amazing collaborator. I am fortunate and grateful to have worked with such a supportive and intelligent researcher. Thank you to everyone at Nissan, including director Maarten Sierhuis as well as Atsu Kobashi, Liam Pedersen, Kathy Sun, and Kelsie Zhao, who have contributed to actualizing my research on an autonomous vehicle.

Some of the most enjoyable moments during my time here have been with my fellow labmates in the Resource-Bounded Reasoning Lab. Luis Pineda and Sandhya Saisubramanian have been both productive collaborators and close friends. They have most certainly made me a better researcher and person. I will greatly miss our discussions. I am also thankful for the other fantastic members of RBR, including: Connor Basich, Rick Freedman, Shuwa Miura, John Peterson, Justin Svegliato, and Xiaojian Wu, all of whom were great colleagues and friends.

I would like to thank Michele Roberts for her support and professionalism in grant-related matters, and Leeanne Leclerc for her exceptional work and advice over the years, without which I would not have navigated the complex graduate processes, nor would I have met many of my close friends.

During my time here, I have met many wonderful friends and housemates. Dirk Ruiken, in particular, has been an exceptional friend, colleague, and housemate. It was only through his guidance that I was able to earnestly pursue robotics. My current housemates Matteo Brucato, Samer Nashed, and Justin Svegliato are also great colleagues and close friends. With these four housemates, I have shared countless hours of meals, games, road trips, laughs, and discussions. Thank all of you for the wonderful memories and heartfelt camaraderie at Gray Street.

Other wonderful friends with whom I have shared many great memories are: Garrett Bernstein, Su Lin Blodgett, Marc Cartright, Ilene Cartright, Orion Cartright, Sebastian Cartright, Kaleigh Clary, Joe Collard, Sally Craig, Hanifa Darwish, Francisco Garcia, Brendan Gavin, Tom Goldstein, Emily Harrington, Mitch Hebert, Tom Hooper, Sam Huston, Myungha Jang, Niri Karina, Yoonheui Kim, Li Yang Ku, Tina Kuksin, Mike Lanighan, Tiffany Liu, Katerina Marazopoulou, Jess McIver, Tamara Mercanti, Annamalai Natarajan, Pinar Ozisik, Sadegh Rabiee, Shiraj Sen, Takeshi Takahashi, Phil Thomas, John Vilk, Sofya Vorotnikova, and Kevin Winner. Thank all of you for making the past years memorable.

This dissertation would not have been possible without the support of my family who have always encouraged me to reach for my dreams. My parents, Robin Wray and Peter Wray, are among the most wisest, most loving, and talented people that I know. Thank you both so much for the values you have instilled in me and for your continued support throughout my life. My wife, Sajia Darwish, has brought a light to my life that I never knew possible. This dissertation was written with her by my side. Thank you Sajia *janem* for your eternal love, care, and support. I look forward to our lifelong journey together.

ABSTRACT

ABSTRACTIONS IN REASONING FOR LONG-TERM AUTONOMY

MAY 2019

KYLE HOLLINS WRAY

B.S., THE PENNSYLVANIA STATE UNIVERSITY

B.S., THE PENNSYLVANIA STATE UNIVERSITY

M.S., THE PENNSYLVANIA STATE UNIVERSITY

M.A., THE PENNSYLVANIA STATE UNIVERSITY

M.S., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Shlomo Zilberstein

The path to build adaptive, robust, intelligent agents has led researchers to develop a suite of powerful models and algorithms for agents with a single objective. However, in recent years, attempts to use this monolithic approach for an ever-expanding set of real-world problems have illuminated its inability to scale, resulting in a fragmented collection of hierarchical and multi-objective models. This trend continues into the algorithms as well, as each approximates an optimal solution in a different manner for scalability. These models and algorithms represent an attempt to solve pieces of an overarching problem: how can an agent explicitly model and integrate the necessary aspects of reasoning required for long-term autonomy?

This thesis presents a general hierarchical and multi-objective model called a *policy network* that unifies prior fragmented solutions into a single graphical decision-making structure. Additionally, the underlying notion of a policy, which is used by all algorithms to describe internal agent state and action, is generalized and unified in this thesis under *controller family policies*.

Policy networks are broadly useful to solve numerous real-world problems including: (1) an autonomous vehicle's (AV) intersection decision-making, semi-autonomous route planning with provable notations of safety, and capability to balance multiple objectives; and (2) a home healthcare robot's reasoning about multiple delivery and monitoring tasks. This thesis presents solutions to both of these distinct problems, with policy networks serving as their shared framework, each acting as part of the overall solution for rich, real-world, scalable decision-making in long-term autonomy.

Reasoning about multiple objectives is common in many real-world domains due to necessary trade-offs among objectives such as solution cost, quality, and time. This thesis considers a general model of *topologically ordered objectives in decision-making* with a preference ordering over objectives induced by a directed acyclic graph and a slack term—allowable deviation from optimal—for each objective. It is applied to AV route planning, which minimizes travel time with slack to improve time autonomous. Topologically ordered objectives in decision-making serve as a demonstration of policy constraints—multiple models, with identical state-action spaces, in a graph of constraints—within a policy network.

To achieve long-term autonomy deployed in the real world, periods of semi-autonomy will be necessary for safety and to expand the agent's capabilities beyond what it can do autonomously. Thus, control will need to be safely transferred to a human for approval or even direct control. This thesis considers a formal framework for *semi-autonomous systems* that explicitly models varying levels of human or agent assistance and safely transfers control among them. It is applied to semi-autonomous vehicles for street-level route planning with proactive transfer of control, given only some of the roads are capable of autonomy, with provable notions of safety. Semi-autonomous systems serve as a demonstration of policy abstractions—multiple models, with differing state-action spaces, that transfer control among themselves—within a policy network.

Finally, a central challenge in long-term autonomy is the ability to manage multiple decision-making problems, perhaps simultaneously, that are unspecified a priori, each with differing state spaces and objectives. This thesis considers a solution for solving *multiple online decision-components with interacting actions (MODIA)* with integrated learning to customize each decision-making scenario over time for ever-improving performance. It is applied to an AV interacting with other vehicles and pedestrians at intersections and demonstrated on a fully operational AV prototype acting on real public roads. MODIA serves as a demonstration of an indefinite number of models—multiple models, with differing state spaces but identical action spaces, the quantity of which is finite but unknown a priori—within a policy network.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

Autonomous systems have been successfully deployed in a wide variety of applications ranging from space exploration [147], water reservoir control [23], smart wheelchairs [116], energy conservation [75], home healthcare robots [139], and autonomous vehicles [134]. *Long-term autonomy* (LTA) has arguably always been the goal of many of these autonomous robotic agents. Generally, LTA refers to an agent that is able to be "deployed for extended periods in real-world environments" [71] and "adapt to changes in the environment in order to remain autonomous" [11]. LTA solutions must provide a "tight integration of a number of autonomous components, including a symbiotic human-robot relationship" [11], such that "integrating state-of-the-art artificial intelligence and robotics research" will "increase their robustness" [54]. Consequently, "LTA systems inherently present an integration challenge, particularly when different AI abilities need to work together" [71]. This includes reasoning about "localisation and navigation; object and/or person perception; plus task planning and/or scheduling" [71]. Due to the sheer complexity of such systems, it is only recently becoming a reality to deploy these kinds of agents at a large scale so as to directly interact with the day-to-day lives of individuals.

One of the most prominent recent applications of long-term autonomy is *autonomous vehicles (AVs).* They serve as the motivating domain for much of the work contained in this thesis. AVs must be able to drive autonomously the majority of the time in both highways and urban environments. They must reliably perform the complex merges, lane changes, yields to oncoming traffic, cautious edges forward for visibility, and negotiations with vehicles, bikes, and pedestrians at a diverse array of intersections. In this thesis, we call these kinds of tasks mid-level decision-making to differentiate it from so called high-level route planning (e.g., GPS navigation) and low-level path planning (e.g., millimeter precision control of the wheels). In other words, mid-level reasoning considers the "task planning and/or scheduling" and incorporates partial observability models for "object and/or person perception." The low- and high-level reasoning considers the "localisation and navigation" of the steering wheel and for the overall route, respectively. Reasoning at these levels should be tightly integrated in order to increase the AV's robustness. All three levels arguably can also entail more than one objective, as AV passenger safety, external vehicle and pedestrian safety, time to reach a destination, and even social acceptability, must all be prioritized and reasoned about for each decision made. Moreover, a human that can monitor the AV is typically required by law, namely the driver, such that a “symbiotic human-robot relationship” must be part of the integrated solution, with a safe transfer of control between the two. AV research overall has advanced rapidly since the DARPA Grand Challenge [127], which acted as a catalyst for subsequent work on low-level sensing [120, 111] and control [74, 35], as well as our recent work on mid-level decision-making [134] and high-level semi-autonomous route planning [132], both discussed within the thesis. These growing lines of research have highlighted the inability for a monolithic model—that is, a massive model with one large state-action space and one objective—to alone tractably capture all necessary aspects of reasoning.

1.1 Models of Reasoning

This thesis focuses specifically on the principled formulation and integration of these reasoning aspects—for example, the low-, mid-, and high-level AV reasoning—towards the goal of long-term autonomy, using the *Markov decision process (MDP)* [8, 55] as a foundation. An MDP consists of four components to describe the necessary pieces to make decisions sequentially over time. First, *states* describe what information is important, such as locations of the AV and other vehicles. Second, *actions* describe what the agent can do, such as stop or go forward. Third, *transitions* describe how the world’s state changes after an action is performed, such as how going forward advances the AV’s location. Fourth, *rewards* describe which states and actions are good or bad for the agent, such as how it is good to arrive at the AV’s destination. At each state, the agent chooses an action to perform. We call this decision-making assignment of state to action a *policy*. These components, or equivalent representations, form the mathematical foundation on top of which the vast majority of planning and learning techniques are built within artificial intelligence research.

As discussed above, for real-world applications such as AVs, a large monolithic MDP simply cannot tractably capture the diverse array of tasks that a long-term autonomous agent inevitably encounters in the environment. Fortunately, many techniques that extend the MDP can address some of the key challenges found in designing the reasoning components for long-term autonomous agents. Hierarchical methods such as the options framework [122], hierarchical abstract machines [87], or macro-actions [53] allow for different compositions of hierarchical policies the agent follows. While this enables tractable solutions to be implemented, each approach has benefits and drawbacks without a unified view about how these hierarchical solutions operate. Multi-objective methods allow for a single task to consider many objectives simultaneously (e.g., quality, cost, and time), and includes scalarization approaches [103], constrained MDPs [2], and lexicographic MDPs [135]. While this enables more than one objective to be considered, each approach again has benefits and drawbacks, without a unified view regarding their relation. Moreover, these multi-objective methods only allow for reasoning about the *same* state and action spaces—the scope of the problem space is identical, with only the description of objective differing. Applications such as AVs require reasoning *simultaneously* about multiple *distinct* tasks or problems, perhaps with different concepts of state and action, such as when there are any number of vehicle and pedestrian interactions present [134]. Lastly, such applications of long-term autonomy can greatly benefit from human feedback and collaboration to further empower the agent to complete its objectives [132]. With all of these pieces available to solve the reasoning problems in an agent with long-term autonomy, no technique exists to easily integrate them. The need for a unified model to simultaneously handle all of these necessary aspects of reasoning for long-term autonomy motivates the approaches contained in this thesis.

1.2 Models of Reasoning in Long-Term Autonomy

Inspired by the need for mathematically principled solutions to scalable reasoning models in long-term autonomy, and building off of the established hierarchical and multi-objective work, this thesis presents a *policy network*. The overall idea is simple and intuitive: (1) break the big problem into small pieces, and (2) represent how the small pieces relate to one another.

For example, consider driving between home and work repeatedly everyday. Each day is different and each individual trip can take a long time even by itself. It is impossible to reason *at the same time* about all interactions with all vehicles we might encounter at all traffic light, stop sign, and yield intersections; every millimeter of control for the wheels at all places; all pedestrians that may cross our path; and all possible roads we could traverse. Instead, we reason and learn about solutions to each small piece separately, with a holistic understanding how each piece relates to others. The combination of these pieces enables us to solve the larger problem and drive between home and work repeatedly everyday.

In a policy network, we define each of these small pieces with its by its own policy space for a reward, describing concerns only related to that task. We then relate these small pieces to one another. For example, one task can constrain what another can do by limiting its policy space, or one task can leverage another as a subtask by temporarily transferring control to it.

Each chapter presents a focused exploration of a novel aspect of a policy network. Moreover, they describe how it can be used as one of these pieces of the holistic solution to AVs. Chapter 4 explores a generalized notion of policy within a policy network. Chapter 5 explores a graph-based preference structure among different objectives in a policy network. Chapter 6 explores the transfer of control among different models in a policy network. Chapter 7 explores the simultaneous interaction of multiple different models in a policy network. These aspects, when combined, aim to provide a holistic representation of many key reasoning aspects towards the goal of long-term autonomy, specifically focused on the motivating autonomous vehicle domain.

1.3 Contributions

This thesis unifies the numerous models and their policy representations for constructing intelligent agents with long-term autonomy. Additionally, it presents solutions to three critical components within such agents: incorporating multiple objectives, leveraging human assistance for safety, and fusing multiple models online for scalable long-term autonomy. We present solutions to autonomous vehicle reasoning and demonstrate success on a true autonomous vehicle prototype.

Policy Networks This thesis presents a single formal unified model that encapsulates hierarchical and multi-objective decision-making: policy networks. We provide complete novel formal proofs for various prior models (e.g., options and constrained MDPs) as well as the three new models presented within this thesis. A novel general algorithm to solve policy networks is described and analyzed. Numerous examples and illustrations are provided to properly illuminate the nuances of policy networks as a new solution to long-term autonomous systems. A home healthcare robot domain is fully explained, implemented, and evaluated on a real robot to demonstrate the effectiveness of policy networks to solve real-world problems.

Controller Family Policies In addition to the unified view of models, this thesis unifies the policy and value formulations used by state-of-the-art algorithms: controller family policies. We provide complete novel formal proofs that the POMDP policies used by these algorithms are actually instances of the controller family representation. We present a novel controller family policy for POMDPs as well: belief-infused finite state controllers. This approach is demonstrated and analyzed in both simulation on standard benchmark domains and on a real robot acting in the world.

Multi-Objective Decision-Making A novel form of multi-objective model called a topological MDP (TMDP) is described, as well as algorithms that scalably solve TMDPs. A theoretical analysis of its properties is provided, with proofs showing that it generalizes lexicographic and constrained MDPs. These models are analyzed on a semi-autonomous vehicle route planning domain using real-world road data.

Semi-Autonomous Systems The novel formal definition of a semi-autonomous system (SAS), including important properties such as strong and transfer of control, is provided. It describes a form of provably safe shared control of a robot between any number of agents and humans. A novel semi-autonomous vehicle domain is presented that implements the SAS model and its transfer of control process. A theoretical analysis shows that it preserves this form of provably safe shared control. Experiments are provided, which use real public road data, illustrating the benefits of the SAS formulation.

Scalable Online Decision-Making A novel formulation called MODIA is presented that allows for multiple decision-making components to simultaneously be active and control the system together. This enables scalability and the ability to handle an unknown a priori number of simultaneous problems encountered. Formal definitions, theoretical analysis, and integration of learning are described in detail. This novel theoretical formulation is analyzed within the context of intersection, pedestrian, lane change, merge, and pass obstacle scenarios for long-term deployments of autonomous vehicles in urban and highway settings. This is implemented on a fully operational autonomous vehicle acting on real public roads.

Autonomous Vehicle Decision-Making Throughout this thesis, we use autonomous vehicles for our motivation and experimentation, as it is the first major example of long-term autonomy within society. As a result, the combination of the solutions presented in each chapter offers a strong overall solution for autonomous vehicle decision-making. The actual design and full implementation on autonomous vehicles clearly demonstrates that our approaches are successful at solving the important decision-making problems for one of the first long-term autonomous systems acting in society.

1.4 Outline

Chapter 2 introduces the core concepts used throughout the thesis. Specifically, it introduces the notion of an MDP and a POMDP, as well as numerous variants. The various forms of policy—that is, how the agent plans to behave in its environment—are covered, in addition to the core algorithms used for each model and policy form. It concludes with a discussion of the relevant hierarchical and multi-objective models as well as other solutions to address scalability in autonomy.

Chapter 3 formally defines policy networks with detailed explanations and formal proofs that prior related work are instances of policy networks. It also presents an algorithm and analyzes its characteristics. It is evaluated on a home healthcare robot domain and demonstrated on a real robot. The following three subsequent chapters include a formal mapping of the chapter's model to a policy network.

Chapter 4 describes controller family policies as a unifying framework for the policy and value representations used by algorithms to solve POMDPs and its variants. Formal proofs that prior policy forms, such as belief point-based, finite state controller (FSC), and compression methods, are all members of the controller family. A new policy form is presented for POMDPs, with experiments that show it can improve the performance of a POMDP solver and an evaluation on a real robot.

Chapter 5 describes the process handling multiple objectives as a directed acyclic graph with slack constraints including a detailed theoretical analysis and approximate algorithms. Detailed solutions which use them are presented for autonomous vehicles.

Chapter 6 formalizes the notion of semi-autonomous systems (SAS), including the definition of a strong SAS—a notion of safety—as well as a formal description of learning to improve autonomy over time. Detailed solutions which use them are presented for semi-autonomous vehicles.

Chapter 7 presents a formal model that allows multiple decision-making models to be simultaneously active, each essentially recommending an action with an executor model making the final decision. This enables a single agent to handle an unknown a priori number of simultaneously encountered scenarios and/or entities in the world. We define properties of this model and a special class of solution that works well in practice. We demonstrate successful implementation in simulation and on Nissan's fully operational AV prototype acting in real intersections on public roads.

Chapter 8 takes a step back and examines all of the techniques presented in the thesis: policy networks, controller family policies, TMDPs, SAS, and MODIA. Conclusions are drawn about what has been accomplished, how this fits into the broader research community, and where this line of work is heading. It concludes with final thoughts on the wide array of novel solutions presented within the thesis toward the goal of agents with long-term autonomy.

CHAPTER 2
BACKGROUND

In this chapter, we introduce the foundational definitions of sequential decision-making models and the core algorithms that solve them in practice. We begin with the Markov decision process (MDP) and discuss related fully observable models. Next, we cover the partially observable MDP (POMDP) and the numerous algorithms used to solve them approximately. The MDP and POMDP serve as the primary models used throughout the thesis. Lastly, we cover relevant hierarchical and multi-objective techniques as well as other general approaches that address scalability in decision-making.

2.1 Markov Decision Process (MDP)

The MDP was created in the 1950's with its notions of optimality given by Bellman [8] and additional refinements by Howard [55]. The model defines a set of relevant states and actions to a decision-maker. At each time step, an action is performed causing the state to update to a successor state following a stochastic process. This stochastic process is said to have the *Markov property*—the next state only depends on the current state and action performed, not any other state visited in the past. Immediate rewards are given to this decision-maker at each time step. While a few notions of objective exist, they each seek to maximize a notion of expected reward experienced over time.

This thesis primarily focuses on a class of MDP: *discrete time, finite state, and finite action over an infinite horizon with discounting starting from an initial state.* This is a modern artificial intelligence perspective on MDPs. Thus, the notation and definitions for the MDP model is first presented with this in mind to unify the overall discussion throughout the thesis. The nuances of the other MDP definitions are discussed once this modern artificial intelligence definition is presented.

2.1.1 Formal Definition

A **Markov decision process (MDP)** [8] is a sequential decision-making model defined by the tuple $\langle S, A, T, R \rangle$:

- 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 state transition function such that $T(s,a,s') = Pr(s'|s,a)$ is the probability of successor state s' given action a was performed in state s, and
- $R : S \times A \to \mathbb{R}$ is a reward function such that $R(s,a)$ is immediate reward for performing action a in state s.

An MDP operates as a stochastic control process over discrete time steps up to a **horizon** $h \in \mathbb{N} \cup \{\infty\}$. At each time step, the system is in a state s and an action a is performed. Immediately, a reward signal is generated following $R(s,a)$. Then, a successor state s' is stochastically generated following $T(s,a,s')$. This process continues until horizon h is reached. An MDP policy describes which actions are performed in which states over time. This policy is then used in an objective function to describe the notion of expected reward. We consider two **objective functions**: expected reward over finite and infinite horizon.

An **infinite horizon** MDP has a horizon $h = \infty$ and a **discount factor** $\gamma \in [0,1)$. The **policy** $\pi : S \to A$ maps each state to an action. Let Π denote the set of all policies. The objective is to find a policy π that maximizes the expected reward over all states:

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

with s^t denoting a random variable for the state at time t generated following T. Therefore, for a policy π, the **value** $V^\pi : S \to \mathbb{R}$ is the expected reward at state s following the *Bellman equation*:

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

It is also convenient to define the Q-value function $Q^\pi : S \times A \to \mathbb{R}$ for state s and action a:

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

The **optimal** policy $\pi^* \in \Pi$ is the policy that obtains the maximal value V^*. The optimal value can be computed by the *Bellman optimality equation*:

$$V^*(s) = \max_{a \in A} \Big(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^*(s')\Big), \tag{2.4}$$

or equivalently denoted $V^*(s)=\max_a Q^*(s,a)$. The optimal policy can be extracted by $\pi^*(s)=\operatorname{argmax}_a Q^*(s,a)$. Equation 2.4 is a contraction operator on the Banach space of value functions with the max norm metric and Lipschitz constant γ. It computes the unique fixed point following Banach's fixed point theorem. This is an important property with infinite horizon objectives, as there is exactly one unique fixed point in value. Additionally, to obtain these optimal values over time, a policy need not also be dependent on the time step. For a given objective, if its policies have this property then they are called *stationary*. There exist optimal stationary policies for infinite horizon MDPs [12]. If this were not the case, then it might require infinite memory to store the policy over time.

A **finite horizon** MDP has horizon $h \in \mathbb{N}$ and time steps denoted $\mathcal{T}=\{1,\ldots,h\}$. In the finite horizon case, optimal policies are often *non-stationary*. Thus, the policy $\pi : S \times \mathcal{T} \to A$ maps states and the current time to an action. Similarly, the objective is to find a policy π that maximizes the expected reward over all states and time:

$$\mathbb{E}\Big[\sum_{t=0}^{h} \gamma^t R(s^t, \pi(s^t,t)) \Big| \pi\Big]. \tag{2.5}$$

The value function $V^\pi : S \times \mathcal{T} \to \mathbb{R}$ at time t for state s^t is:

$$V^\pi(s^t,t) = R(s^t,\pi(s^t,t)) + \sum_{s^{t-1}\in S} T(s^t,\pi(s^t,t),s^{t-1}) V^\pi(s^{t-1},t-1) \tag{2.6}$$

with $V^\pi(s,0)=0$. The Bellman optimality equation to compute V^* follows in the natural way:

$$V^*(s^t,t) = \max_{a\in A}\Big(R(s^t,a) + \sum_{s^{t-1}\in S} T(s^t,a,s^{t-1}) V^*(s^{t-1},t-1)\Big). \tag{2.7}$$

2.1.1.1 Important Related Terminology

The objective function in Equation 2.1 requires that all states maximize expected utility over time given a single policy. In many problems, we know the starting state of the system and do not need to plan for all possible states. This can be exploited to great effect in terms of performance and memory usage. It is also necessary for tractability in continuous or otherwise infinite state spaces. We call this an **initial state** $s^0 \in S$. Hereafter, we will assume initial state is provided.

Also, there is the important notion of an **absorbing state**. A state s is called absorbing if $T(s,a,s)=1$ for all actions a. This is also called a *terminal state* in some cases. These will be particularly useful in the next few sections.

For any initial state s^0, we can define a **history** $\bar{h} = \langle s^0, a^0, s^1, a^1, \ldots, a^{h-1}, s^h \rangle$ over some horizon h as the sequence of states encountered and actions performed over time. Let $\bar{H}$ denote the set of all histories, and $\bar{H}^h$ denote the set of all histories of horizons 0, 1, 2, ..., $h-1$, and h. This is also called a *trial* in some planning contexts or an *episode* in some reinforcement learning contexts, though there are subtle differences. If the absorbing state is known, trials often terminate once one has been reached.

Most problem domains describe the state space S in terms of **state factors** $S_1, \ldots, S_k$ such that $S = S_1 \times \cdots \times S_k$. This is opposed to a so-called *flat* state space representation. The action space A can also be factored in terms of **action factors** $A_1, \ldots, A_k$ such that $A = A_1 \times \cdots \times A_k$. Factored representations provide no additional benefits aside from convenience. Instead, additional assumptions are required, such as independence assumptions in state transitions or reward, to allow this explicit structure to be exploited.

Finally, the application of the Bellman optimality equation, also called an *update equation*, allows us to compute a **residual** error. Given V and a V' resulting after an update, the residual for a state s is $|V(s) - V'(s)|$. The residual over all states is given by a *max norm* $\|V - V'\|_\infty = \max_s |V(s) - V'(s)|$. This may be used to check convergence in algorithms.

2.1.1.2 Additional Refinements

The modern use of MDPs within the artificial intelligence community has refined the definitions surrounding MDPs. For example, it is unfortunately common practice to abuse notion and put the discount factor γ in the model definition. This confuses the three distinct components: model, policy, and objective. Additionally, the distinction of finite versus infinite state or action spaces is not always mentioned, as finite is essentially assumed by default. Similarly, it is assumed discrete time by default as well. There are, however, some common refinements that arise in practice.

First, the reward function can simply depend on the state $R' : S \to \mathbb{R}$ or depend on the state, action, and successor $R'' : S \times A \times S \to \mathbb{R}$. The former is a weaker definition of reward due to the max inside the Bellman optimality equation. That is to say, given any R', we can write the equation with $R : S \times A \to \mathbb{R}$ instead using $R(s,a) = R'(s)$; however, it is not always possible for R' to be defined for an arbitrary R, without adding extra states, etc. Interestingly, R'' and R are equivalent representations due to the summation inside the Bellman optimality equation. Namely, we can rewrite one as the other following:

$$R(s,a) = \sum_{s' \in S} T(s,a,s') R''(s,a,s').$$

We choose the cleaner reward R throughout this thesis, though it might affect approximate algorithms that manipulate the Bellman optimality equation.

A different form of policies also exists that admits *stochastic* actions with $\pi : S \times A \rightarrow [0,1]$ such that $\pi(s,a) = Pr(a|s)$. Importantly, it is proven that for all MDP models we consider, there exists a deterministic policy $\pi : S \rightarrow A$ that obtains optimal values with $V^* = V^\pi$. Stochastic policies are *not required* in general. Stochastic policies are primarily useful for approximation and online learning algorithms. However, one notable exception are *constrained MDPs*, as detailed in Section 2.4.2.2, which require stochastic policies to obtain their maximal possible V^*.

Briefly, there are two other adjustments used in practice. First, reward can be described in terms of *cost*. In many cases, this refers to the negated reward. Second, without loss of generality, *distinct action sets* can be defined for each state s by overloading notation with $A(s) \subseteq A$.

This thesis focuses on infinite horizon MDPs since the finite horizon case can, in essence, be captured by the infinite horizon case. Concretely, let $\gamma = 1$, a state factor $\mathcal{T}$ for time that decrements each step, and an absorbing state when the state factor for $\mathcal{T}$ is 0 with a reward of 0. Technically, classical definitions of finite horizon MDPs allowed for T and R to depend on the time remaining as well, which is handled by this mapping. Note that this form still has the Markov property, as the state itself only depends on the previous state, even if time is a factor.

Finally, a third less commonly used, though equally valid, objective function exists as well: the average reward MDP. We leave this and other rare objectives to future analysis outside this thesis.

2.1.2 Stochastic Shortest Path (SSP) Problems

While MDPs are quite expressive in their own right, there is a more general formulation that is widely used. Consider an MDP in which there is no discounting and the horizon is finite but unknown a priori. We call this an *indefinite horizon*. Given an initial and goal state, this describes a stochastic shortest path problem.

Formally, a **stochastic shortest path (SSP) problem** [10] is a sequential decision-making model defined by the tuple $\langle S, A, T, C, s^0, s^g \rangle$:

- S is a finite set of states,
- A is a finite set of actions,
- $T : S \times A \times S \rightarrow [0,1]$ is a state transition function such that $T(s,a,s') = Pr(s'|s,a)$ is the probability of successor state s' given action a was performed in state s,

- $C : S \times A \rightarrow \mathbb{R}^+$ is a non-negative cost function such that $C(s,a)$ is immediate cost for performing action a in state s,
- $s^0 \in S$ is an initial state, and
- $s^g \in S$ is a goal state with $T(s^g, \cdot, s^g) = 1$ and $C(s^g, \cdot) = 0$ (i.e., absorbing zero-cost).

Like MDPs, an SSP policy is defined by $\pi : S \rightarrow A$, provided it is stationary. Moreover, a stationary policy is said to be **proper** if the probability of reaching the goal s^g is 1 as $t \rightarrow \infty$. Any state in which this probability is less than 1 is called a **dead end** [68]. Importantly, SSP problems assume: (1) there exists a proper stationary policy, and (2) all improper stationary policies have infinite cost.

Similar to an MDP, the objective is to find the policy π that *minimizes* the expected cost to reach the goal starting from the initial state:

$$\mathbb{E}\Big[\sum_{t=0}^{\infty} C(s^t, \pi(s^t)) \Big| \pi, s^0\Big] \tag{2.8}$$

with s^t denoting a random variable for the state at time t generated following T. For a policy π, the value $V^\pi : S \rightarrow \mathbb{R}$ is the expected cost at state s following:

$$V^\pi(s) = C(s, \pi(s)) + \sum_{s' \in S} T(s, \pi(s), s') V^\pi(s'). \tag{2.9}$$

Similarly, the Bellman optimality equation is given by:

$$V^*(s) = \min_{a \in A} \Big(C(s,a) + \sum_{s' \in S} T(s,a,s') V^*(s') \Big). \tag{2.10}$$

The optimal policy is likewise extracted by $\pi^*(s) = \operatorname{argmin}_a Q^*(s,a)$. Lastly, it is an operator similar to MDPs, however, it has a weighted max norm metric.

Technically, the original SSP definition allows for negative costs and a non-finite set of actions. However, the proper policy assumption must be modified, and a second assumption is needed with respect to the compactness of actions and continuity with respect to V. Also, a single initial and goal state in the SSP model automatically generalize to multiple initial and goal states. This mapping uses a zero-cost transition without discounting to trivially model any number of initial or goal states using just one of each.

In any case, this form of SSP is more general than the MDP models and objectives previously defined—that is, any finite or infinite horizon MDP with discrete time, finite state, and finite action can be mapped to an SSP. For any MDP, simply add an initial and goal state. Assign zero cost uniform transition over all MDP states from the initial, multiply the MDP state transitions by γ for remaining in the MDP's original state space, and assign a probability of $(1-\gamma)$ for transitioning from these states to the goal. The converse mapping does not always exist, as indefinite horizon without discounting cannot be properly mapped to the aforementioned formulation of MDPs.

2.1.3 Path Planning with Harmonic Functions

Most robots have a more focused planning problem: quickly and smoothly navigate from one location to another while avoiding obstacle collisions. This domain of problems and solutions are called *path and motion planning.* For path planning used within this thesis, we consider a special kind of *potential field method* that employs a solution to Laplace's equation called *harmonic functions* [29, 133], fully formalized for robotics by Connolly and Grupen [30]. Among the many benefits of harmonic function solutions is their unique relation to SSPs. Due to this important relationship, we will use the SSP notation for overall consistency.

Intuitively, harmonic functions *flow* from a source to a sink following physics equations that can model basic fluid dynamics. These paths or *streamlines* are smooth natural paths that optimally minimize the probability of hitting an obstacle while travelling to the goal. Formally, Laplace's equation is a form of Poisson's equation with an equality of zero. We have twice continuously differentiable function $\phi : X \rightarrow \mathbb{R}$ defined on n-dimensional region $X \subset \mathbb{R}^n$ with boundary ∂X:

$$\bigtriangledown^2 \phi = \sum_{i=1}^{n} \frac{\partial^2 \phi}{\partial x_i^2} = 0.$$

The solutions to Laplace's equation are called **harmonic functions**. This is solved using a bounded discrete regular sampled grid (of *states*) on X denoted $S = S_1 \times \cdots \times S_n$ with each $S_i = \{1, \ldots, m_i\}$ for some $m_i \in \mathbb{N}$. On each grid state, we apply a Taylor series approximation, use Dirichlet boundary conditions with 1 for obstacles and 0 for goals, and apply finite (central) difference. Let $G \subset S$ denote the set of absorbing goal states, and $O \subset S$ denote the set of absorbing obstacle states. It is assumed all boundary states are either goals or obstacles. Lastly, let $N(s) = \{s' \in S | s'_i = s_i \pm 1 \forall i\}$ be the set of all $2n$ neighboring grid states of $s \notin G \cup O$.

As in SSPs, there is a resulting system of equations acting as a value update equation. Let $V : S \rightarrow \mathbb{R}$ be the **value** of each grid state $s \notin G \cup O$ following:

$$V(s) = \sum_{s' \in N(s)} \frac{1}{|N(s)|} V(s'). \tag{2.11}$$

For a goal $s \in G$, $V(s) = 0$, and for an obstacle $s \in O$, $V(s) = 1$. As written, this is a special case of successive over relaxation (SOR) called **Gauss-Seidel**. Interestingly, the optimal values correspond directly to the probability of hitting an obstacle while moving to a goal state [28].

These values determine the path the robot will take called **streamlines**. Streamlines are essentially the **policy** of the harmonic function with respect to the original state space X. It follows gradient descent by interpolating over the grid values and terminates early once it reaches a goal or obstacle. Formally, given an initial x^0, a streamline of length τ is defined recursively by the locations $\langle x^0, x^1, \ldots, x^\tau \rangle$ induced by:

$$x^{t+1} = x^t - h \bigtriangledown \hat{\phi}(x^t) \tag{2.12}$$

with step size h and $\hat{\phi} : X \to \mathbb{R}$ denoting the interpolated approximate of ϕ using V^*, assuming it is normalized to a unit vector.

Other techniques exist for robotic path and motion planning. *Probabilistic roadmaps (PRMs)* [64] begin by randomly selecting points to create a collision-free graph. Then, it enters a query phase that connects the start and goal locations. Eventually it finds the fastest route within the final graph. *Rapidly-exploring random trees (RRT)* [69] randomly construct trees following any general state transition, including any non-holonomic constraints. While fast, this random exploration does not always produce the desired smooth optimal trajectories, and does not explicitly incorporate measures of obstacle avoidance. We leave these other forms of path planning to analysis in future work outside this thesis.

2.1.4 Algorithms

Solving MDPs in general is P-complete in the size of the problem in states and actions [86]. However, it is common practice to have an exponential state space defined by all permutations of state factors [77]. In any case, there are two very important optimal algorithms for MDPs that informed the decades of subsequent algorithms which followed.

Value iteration (VI) [8] applies Equation 2.4 to all states until a convergence criterion is met. A very important corollary is that these updates can be applied to any state in any order, provided no state is starved of updates (i.e., all states are sampled infinitely often as $t \to \infty$). This is called asynchronous VI, and it forms the foundation of the majority of approximate algorithms due to its ease to prove convergence in the limit. Policy iteration (PI) [55] instead iterates over two steps: policy evaluation and policy improvement. Policy evaluation consists of solving the system of linear equations formed by a fixed policy in Equation 2.2. Policy improvement chooses the best action for each state given the policy evaluation. This process terminates once the policy remains unchanged.

SSP algorithms are also applicable to MDPs but can benefit from a heuristic to guide the search process. Two equally important optimal algorithms exist. Labeled real-time dynamic programming (LRTDP) [14] generalizes RTDP which samples trials (i.e., histories) and applies the Bellman update equation in post order traversal along the trial. LRTDP includes a labelling procedure that intelligently skips unnecessary exploration and application of the update equation on collections of states with low residual error. LAO* generalizes AO* to handle the loops found in SSP problems [52]. It similarly explores reachable states following a policy and essentially checks if all states within this best partial solution graph are solved with low residual. Both LRTDP and LAO* converge to the optimal policy in SSPs and MDPs, often much faster than value or policy iteration.

2.2 Partially Observable MDP (POMDP)

After MDPs were developed by Bellman, it took a few decades of attempts to generalize the MDP to capture partial observability. A form of partial observability in Markov processes was developed by Drake [36] in his 1962 thesis. Three years later, Åström [101] defined an early version of the modern POMDP through the idea of controlling a partially observable Markov process, leveraging what we might now call a belief MDP. The modern formulation of POMDPs was solidified in the 1970s by Sondik and Smallwood [112] for a finite horizon. Years later, a tractable solution to the infinite horizon objective was developed by Smallwood [117]. The model itself is an MDP in which the state is not necessarily observable. Instead, the agent obtains hints at the true state through noisy observations. This state uncertainty is represented in the agent as maintained *belief* over the true state. Its action decisions are made according to this belief.

2.2.1 Formal Definition

A **partially observable Markov decision process (POMDP)** [112] is a sequential decision-making model defined by the tuple $\langle S, A, \Omega, T, O, R \rangle$:

- S is a finite set of states,
- A is a finite set of actions,
- Ω is a finite set of observations,
- $T : S \times A \times S \to [0,1]$ is a state transition function such that $T(s,a,s') = Pr(s'|s,a)$ is the probability of successor state s' given action a was performed in state s,

- $O : A \times S \times \Omega \rightarrow [0,1]$ is an observation function such that $O(a,s',\omega) = Pr(\omega|a,s')$ is the probability of observing ω given action a was performed resulting in successor s', and
- $R : S \times A \rightarrow \mathbb{R}$ is a reward function such that $R(s,a)$ is immediate reward for performing action a in state s.

The agent does not observe the true state of the system. Instead, it maintains a **belief** $b \in \triangle^n$ over the true state, with $\triangle^n$ denoting the standard $(n-1)$-simplex. For belief b, performing a and observing ω yields successor belief b' at s':

$$b'(s') = Pr(\omega|b,a)^{-1} O(a,s',\omega) \sum_{s \in S} T(s,a,s') b(s), \tag{2.13}$$

with normalizing constant $Pr(\omega|b,a)^{-1}$. Let $b'_{a\omega}$ denote the resulting belief after applying Equation 2.13 to all s'. It is convenient to refer to sets of beliefs $B \subseteq \triangle^n$, such as the set of **reachable beliefs** from an initial belief b^0 following belief updates (Equation 2.13) denoted $\mathcal{R}(b^0)$. Importantly, the belief state is a *sufficient statistic* for the entire history and initial belief—that is, no prior additional historic information could be used to improve the current belief. Thus, the belief update process also follows the Markov property.

A POMDP operates as a stochastic control process over discrete time steps up to a **horizon** $h \in \mathbb{N} \cup \{\infty\}$. At each time step the system is in a true state s, but the agent has belief state b that informs which action a is performed. Immediately, a belief-based reward signal is generated weighting each state by the belief: $R(b,a) = \sum_s b(s) R(s,a)$. The successor state s' is generated stochastically following $T(s,a,s')$; however the resulting s' is not known to the agent. Instead, an observation ω is generated from s' stochastically following $O(a,s',\omega)$ which is observed by the agent. The agent then updates its belief accordingly. This process continues until horizon h is reached. A POMDP policy describes which actions to perform based on the belief of the agent. We consider two **objective functions**: expected reward over finite and infinite horizon.

An **infinite horizon** POMDP has horizon $h = \infty$ and a **discount factor** $\gamma \in [0,1)$. The **policy** $\pi : \triangle^{|S|} \rightarrow A$ maps each belief to an action. It is stationary. Let Π denote the set of all policies. The objective is to find a policy π that maximizes the expected reward:

$$\mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t R(b^t, \pi(b^t)) \Big| \pi, b^0\Big] \tag{2.14}$$

with b^t denoting a random variable for the belief state at time t generated following T and O. This assumes an initial belief b^0 is provided. A POMDP formulation exists which need not require an initial belief; however, it requires the application of a Bellman optimality equation on all uncountably infinite belief states an infinite number of times. This is highly intractable and thus the overwhelming majority of research assume an initial belief.

For policy π, the **value** $V^\pi : \triangle^{|S|} \to \mathbb{R}$ is the expected reward at belief b with Bellman equation:

$$V^\pi(b) = R(b, \pi(b)) + \gamma \sum_{\omega \in \Omega} Pr(\omega | b, \pi(b)) V^\pi(b'_{\pi(b)\omega}) \tag{2.15}$$

and $R(b,a) = \sum_s b(s)R(s,a)$ and $b'_{\pi(b)\omega}$ following the belief update equation. As with MDPs, it is convenient to define a Q-value function $Q^\pi : \triangle^{|S|} \times A \to \mathbb{R}$ for belief b and action a as:

$$Q^\pi(b,a) = R(b,a) + \gamma \sum_{\omega \in \Omega} Pr(\omega | b, a) V^\pi(b'_{a\omega}). \tag{2.16}$$

A policy $\pi^* \in \Pi$ is **optimal** if it obtains the maximal value denoted as V^*. This optimal value can be computed by the Bellman optimality equation over each belief b:

$$V^*(b) = \max_{a \in A} \Big(R(b,a) + \gamma \sum_{\omega \in \Omega} Pr(\omega | b, a) V^\pi(b'_{a\omega}) \Big), \tag{2.17}$$

or equivalently $V^*(b) = \max_a Q^*(b,a)$. The optimal policy can be found by $\pi^*(b) = \operatorname{argmax}_a Q^*(b,a)$.

Unfortunately, infinite horizon POMDPs, with or without an initial belief, are undecidable for most problems because there are often countably infinite reachable beliefs. Luckily, Sondik [117] also proved that the solution to a discounted finite horizon POMDP can be used to approximate the infinite horizon POMDP. As such, the finite horizon POMDP will be defined with discounting. Additionally, we will assume a stationary policy; however, since individual beliefs are typically only visited once, it implicitly captures a kind of non-stationary policy instead of an explicit non-stationary policy with a time factor.

A **finite horizon** POMDP has horizon $h \in \mathbb{N}$ and a discount factor $\gamma \in [0,1]$. The stationary policy $\pi : \triangle^{|S|} \to A$ maps beliefs to actions. The objective is similar to the infinite horizon:

$$\mathbb{E}\Big[\sum_{t=0}^{h} \gamma^t R(b^t, \pi(b^t)) \Big| \pi, b^0 \Big]. \tag{2.18}$$

Smallwood and Sondik [112] proved that a finite horizon POMDP has a piecewise-linear convex (PWLC) value function. It is defined by a set of α-vectors $\Gamma = \{\alpha_1, \ldots, \alpha_r\}$ with each α-vector $\alpha_i = [\alpha_i(s_1), \ldots, \alpha_i(s_n)]^T$ assigning values of each state. Thus, a policy π can be equivalently defined by a properly defined Γ, with each $\alpha \in \Gamma$ associated with an action $a_\alpha \in A$. For policy $\pi \equiv \Gamma$, the value function $V^\pi : \triangle^{|S|} \rightarrow A$ for belief b is:

$$V^\pi(b) = \max_{\alpha \in \Gamma} \sum_{s \in S} b(s)\alpha(s). \tag{2.19}$$

As written, Equations 2.15 and 2.17 both hold in this finite horizon case as well. Therefore, we can apply Equation 2.19, as well as the belief update from Equation 2.13, to Equation 2.17 in order to obtain a tractable operator. Thus, the Bellman optimality equation can be written for a belief b as:

$$V^*(b) = \max_{a \in A} \Big(R(b,a) + \gamma \sum_{\omega \in \Omega} \max_{\alpha' \in \Gamma} \sum_{s \in S} b(s) \sum_{s' \in S} T(s,a,s') O(a,s',\omega) \alpha'(s') \Big). \tag{2.20}$$

If the initial values are set to $\alpha(s) = R^{min}/(1-\gamma)$, then we V^* weakly monotonically increases to the infinite horizon objective's values as $t \rightarrow \infty$, acting as a lower bound [79, 90].

2.2.1.1 Belief MDP

Any POMDP can be mapped to a special continuous MDP known as a *belief MDP* [63]. This is an important realization regarding the POMDP model because it allows us to prove properties about a finite or infinite state MDPs and have it hold for finite state POMDPs as well.

Formally, for any POMDP $\langle S, A, \Omega, T, O, R \rangle$, the equivalent **belief MDP** is defined by the MDP tuple $\langle B, A, \tau, \rho \rangle$:

- $B \subseteq \triangle^{|S|}$ is the set of relevant belief states,
- A is the same set of actions,
- $\tau : B \times A \times B \rightarrow [0,1]$ is the belief state transition such that $\tau(b,a,b') = Pr(b'|b,a)$ is $\tau(b,a,b') = \sum_\omega Pr(\omega|b,a)[b'_{a\omega} = b']$, with Iversen bracket $[\cdot]$, and
- $\rho : B \times A \rightarrow \mathbb{R}$ is the reward function such that $\rho(b,a) = \sum_s b(s)R(s,a)$.

The objective, policy, and value equations follow the same structure as in continuous state MDPs. Thus, the Bellman optimality equation for a belief MDP is simply:

$$V^*(b) = \max_{a \in A} \Big(\rho(b,a) + \gamma \int_{\triangle^{|S|}} \tau(b,a,b') V^*(b') db' \Big). \tag{2.21}$$

By applying the definition of τ, we obtain the equivalent Bellman optimality equation:

$$V^*(b) = \max_{a \in A} \Big(\rho(b,a) + \gamma \sum_{\omega \in \Omega} Pr(\omega|b,a) V^*(b'_{a\omega}) \Big). \tag{2.22}$$

2.2.2 Algorithms

As previously stated, solving infinite horizon POMDPs in general is undecidable [117]. Solving finite horizon POMDPs in general is PSPACE-complete in the size of the problem in beliefs resulting from the states, actions, and observations [86]. Since P $\subseteq$ PSPACE and NP $\subseteq$ PSPACE, POMDPs are very challenging to solve. This fact has limited their use in practical applications as well. Thankfully, a rich variety of approximate policy and value formulations exist, with numerous approximate algorithms for each. As described above, they take advantage of Sondik's representation of finite horizon POMDPs as an approximation to infinite horizon POMDPs.

The optimal algorithm is given by Equation 2.17. For any initial belief b^0, explore the set $\mathcal{R}(b^0)$, then apply Equation 2.17 in post order traversal. Of course, this tree of reachable beliefs is countably infinite, hence the undecidability of infinite horizon POMDPs. Sondik's approximation instead computes the reachable beliefs up to a horizon h and applies Equation 2.20 up the tree. From this foundation, three classes of approximate algorithms have been developed. Each represents the agent's policy in a different manner, as formally described in Chapter 4.

2.2.2.1 Approximate Algorithms

Point-based approaches are generally based on PBVI [90] and select a subset of the reachable beliefs $B \subseteq \mathcal{R}(b^0)$ on which to apply a slightly modified Bellman optimality update equation. Perseus [118] improves PBVI by intelligently selecting beliefs, thus applying this update less frequently. HSVI2 [113] and SARSOP [72] tightly couple belief point selection with the update equation by selecting a new belief each time corresponding to upper and lower bounds on the value function. Various techniques can be used to augment these algorithms even further, such as the σ-approximation which constrains the size of the belief points [138].

Finite state controller (FSC) algorithms assume a policy formulation as a stochastic FSC. The original formulation performed policy iteration with operations to add, merge, and prune nodes [51]. While an improvement exists BPI [95] that intelligently adds a single node, it tends to get stuck in local optima. Recently, vast improvements have been made use a nonlinear programming (NLP) method on a fixed number of controller nodes for both stochastic FSCs [3] and deterministic FSCs [70]. We will show in Chapter 4 how a more general controller family [140] can be described that generalizes many approximate policy and value forms.

Lastly, compression algorithms compress the POMDP into a smaller form. The intuition is that the true problem is actually a much smaller one over a small subset of the full POMDP space. A policy is defined within this compressed POMDP. It selects actions following the compressed

POMDP, while the action is actually affects the original POMDP. VDC uses Krylov iteration to perform a linear compression [94]. In contrast, E-PCA uses exponential-family principle components analysis to compute a non-linear compression [107].

2.3 Semi-Markov Decision Process (SMDP)

The MDP model can be generalized in another manner beyond SSP problems. Consider an MDP in which the control of the system is sojourn—actions are durative, selected between decision epochs, with rewards that are generated between these stochastic periods of time. This describes a semi-Markov decision process (SMDP). The modern use of the SMDP model has been to describe hierarchical MDP methods [122, 88, 34], though it is explored in its own right in both planning [56] and learning [18] contexts.

The **semi-Markov decision process (SMDP)** [98] is a sequential decision-making model defined by the tuple $\langle S, A, T, F, R, \rho \rangle$:

- S is a finite set of states,
- A is a finite set of actions,
- $T : S \times A \times \mathbb{R}^+ \times S \rightarrow [0,1]$ is a state transition function such that $T(s,a,\tau,s') = Pr(s'|s,a,\tau)$ is the probability of being in state s', given action a was performed in state s, and the next decision epoch has not occurred prior to τ,
- $F : S \times A \times \mathbb{R}^+ \rightarrow [0,1]$ is a cumulative distribution function for sojourn time random variable J such that $F(s,a,\tau) = Pr(J \leq \tau | s,a)$ at sojourn time τ after performing action a in state s,
- $R : S \times A \rightarrow \mathbb{R}$ is a reward function such that $R(s,a)$ is immediate reward for performing action a in state s, and
- $\rho : S \times A \times \mathbb{R}^+ \rightarrow \mathbb{R}$ is a expected reward function such that $\rho(s,a,\tau)$ is reward rate at sojourn time τ after performing action a in state s, but before the next action is performed.

SMDP policies and objectives are the same as those found in MDP. However, the value equations must now account for the time spent accruing reward while the action is being executed. There are three notions of time. First, the actual system's **natural process time** is denoted by non-negative $\tau \in \mathbb{R}^+$. Second, a **decision epoch** is denoted by time step index $t \in \mathbb{N}$, referring to the interval of time $[\tau^1 + \cdots + \tau^t, \tau^1 + \cdots + \tau^{t+1})$ within the natural process time. Third, the **sojourn time** of decision epoch t is denoted by $\tau^t \in \mathbb{R}^+$ is essentially the duration of the decision epoch. This notation is overloaded, as it refers to both this duration and the random variable that determines this duration (F detailed below). Also, this notation means natural process time τ is the sum of sojourn times: $\tau = \tau^1 + \cdots \tau^t + \epsilon$ after any t decision epochs, plus the amount remaining time $\epsilon \geq 0$ it has been since the last decision epoch t. On occasion, we will omit extraneous notation when it is clear which time is considered. The underlying stochastic process now follows $Pr(\tau, s'|s,a)$ as its actual state transition, equating to the probability the next decision epoch occurs at or before sojourn time τ and the successor state at that time is s'. Similarly, we denote $Pr(d\tau, s'|s,a)$ as its time-differential.

Interestingly, we can construct both continuous and discrete time MDP formulations from the SMDP with an appropriate assignment of F. The continuous time MDP has:

$$F(s,a,\tau) = 1 - e^{-\alpha\tau} \qquad \text{and} \qquad F(s,a,d\tau) = \alpha e^{-\alpha\tau} d\tau, \tag{2.23}$$

describing an exponentially distributed sojourn times τ at rate α. The discrete time MDP has:

$$F(s,a,\tau) = \begin{cases} 0, & \text{if } \tau \leq \triangle\tau \\ 1, & \text{if } \tau > \triangle\tau \end{cases} \qquad \text{and} \qquad F(s,a,d\tau) = \begin{cases} 1\ d\tau, & \text{if } \tau = \triangle\tau \\ 0\ d\tau, & \text{otherwise} \end{cases} \tag{2.24}$$

with $\triangle\tau > 0$ denoting the fixed discrete time step duration.

An infinite horizon SMDP has horizon $h = \infty$ and continuous time discount rate $\alpha > 0$. A policy $\pi : S \to A$ maps each state to an action. There exist optimal stationary policies for infinite horizon SMDPs [98]. The objective is to find a policy π that maximizes the expected reward over all states:

$$\mathbb{E}\Big[\sum_{t=0}^{\infty} e^{-\alpha\tau^t} \Big(R(s^t, \pi(s^t)) + \int_{\tau^t}^{\tau^{t+1}} e^{\alpha\tau^t} \rho(s^t, \pi(s^t), \tau - \tau^t) d\tau \Big) \Big| \pi \Big] \tag{2.25}$$

with s^t denoting a random variable for the state at decision epoch t, τ^t denoting the random variable for decision epoch start times within the natural process, and the combined term $\tau - \tau^t$ denoting a random variable for each decision epoch t's sojourn time.

For a policy π, the value $V^\pi : S \to \mathbb{R}$ is the expected reward at state s following:

$$V^\pi(s) = \mathcal{R}(s, \pi(s)) + \sum_{s' \in S} \int_0^\infty e^{-\alpha\tau} Pr(d\tau, s'|s, \pi(s)) V^\pi(s'), \tag{2.26}$$

with expected reward $\mathcal{R} : S \times A \to \mathbb{R}$. Similarly, the Bellman optimality equation is given by:

$$V^*(s) = \max_{a \in A} \Big(\mathcal{R}(s,a) + \sum_{s' \in S} \int_0^\infty e^{-\alpha\tau} Pr(d\tau, s'|s,a) V^*(s') \Big). \tag{2.27}$$

Again, the optimal policy is extracted by the same $\pi^*(s) = \operatorname{argmax}_a Q^*(s,a)$. SMDPs have two assumptions: (1) there exists an $\epsilon > 0$ and $\delta > 0$ such that $F(s,a,\delta) \leq 1 - \epsilon$ for all s and a, and (2) $\mathcal{R}(s,a)$ is bounded.

Intuitively, the expected reward $\mathcal{R}$ is defined as the immediate reward plus the expected sojourn time reward. This reward considers the distribution sojourn time—duration of the decision epoch—and the rewards accrued over this time, which change depending on the state in the alternate ρ':

$$\mathcal{R}(s,a) = R(s,a) + \int_0^\infty \int_0^{\tau'} \rho(s,a,\tau) d\tau F(s,a,d\tau'). \tag{2.28}$$

Now, there are variants to the SMDP definition: (1) an equivalent SMDP using a ρ', and (2) a simplifying formulation called an embedded MDP.

First, as written, ρ denotes the expected reward rate following a sojourn time τ [87]. Equivalently, the SMDP can be formulated using a different $\rho' : S \times A \times S \to \mathbb{R}$. Thus, we have an equivalence of both representations following [98]:

$$\rho(s,a,\tau) = e^{-\alpha\tau} \sum_{s' \in S} \rho'(s,a,s') T(s,a,\tau,s'). \tag{2.29}$$

Consequently, we may write an alternate form of the expected reward $\mathcal{R}(s,a)$:

$$\mathcal{R}(s,a) = R(s,a) + \int_0^\infty \int_0^{\tau'} e^{-\alpha\tau} \sum_{s' \in S} \rho'(s,a,s') T(s,a,\tau,s') d\tau F(s,a,d\tau'). \tag{2.30}$$

Thus, we can use ρ or ρ' without loss of generality.

Lastly, it is common to consider a special case in which the state transition only occurs at the start of a decision epoch. We call this the **embedded MDP**. Observe that we could consider a state transition function $T' : S \times A \times S \to [0,1]$ is given such that $Pr(\infty, s'|s,a) = T'(s,a,s')$ [98]. It defines the probability of regaining control in a successor state s', observing that the probability of regaining control on or before τ tends to 1 as $\tau \to \infty$. Intuitively this captures the embedded MDP's concern only about the decision points. Now for the state transition we can use:

$$Pr(\tau, s'|s,a) = F(s,a,\tau) T'(s,a,s'). \tag{2.31}$$

Formally, applying Equation 2.31 to Equation 2.27 results in the embedded MDP's Bellman optimality equation:

$$V^*(s) = \max_{a \in A} \Big(\mathcal{R}(s,a) + \gamma(s,a) \sum_{s' \in S} T'(s,a,s') V(s') \Big), \tag{2.32}$$

for some state-action dependent discount factor defined as:

$$\gamma(s,a) = \int_0^\infty e^{-\alpha\tau} F(s,a,d\tau). \tag{2.33}$$

By the first assumption of SMDPs we have $\gamma(s,a) \in [0,1)$, allowing for a valid contraction operator in the Banach space of value functions with Lipschitz constant $\gamma_{max} = \max_s \max_a \gamma(s,a)$. Therefore, the embedded MDP can be solved using most MDP or SSP algorithms.

2.3.1 Discrete Time SMDP

This thesis primarily considers a special form of *discrete time* SMDP used as the mathematical foundation for all widely-used hierarchical planning and learning algorithms.

Formally, the **discrete time SMDP** is a sequential decision-making model defined by the tuple $\langle S, A, T, F, R, \rho \rangle$:

- S is a finite set of states,
- A is a finite set of actions,
- $T : S \times A \times \mathbb{N} \times S \rightarrow [0,1]$ is a state transition function such that $T(s,a,\tau,s') = Pr(s'|s,a,\tau)$ is the probability of being in state s', given action a was performed in state s, and the next decision epoch has not occurred prior to τ,
- $F : S \times A \times \mathbb{N} \rightarrow [0,1]$ is a cumulative distribution function for sojourn time random variable J such that $F(s,a,\tau) = Pr(J \leq \tau | s,a)$ at sojourn time τ after performing action a in state s,
- $R : S \times A \rightarrow \mathbb{R}$ is a reward function such that $R(s,a)$ is immediate reward for performing action a in state s, and
- $\rho : S \times A \times \mathbb{N} \rightarrow \mathbb{R}$ is a expected reward function such that $\rho(s,a,\tau)$ is reward rate at sojourn time τ after performing action a in state s, but before the next action is performed.

All the same concepts apply from the general SMDP formulation. We assign the general SMDP's F to be a step function with a step size $\triangle\tau = 1$, as referenced in Equation 2.24. As a result, the integrals can be replaced by summations because the time-derivative of the state transition $Pr(d\tau, s'|s,a)$ and sojourn time CDF $F(s,a,d\tau)$ has 0 probability weight when not in $\mathbb{N}$. Also, we assign the general SMDP discount rate as $\alpha = -\log(\gamma)$ to obtain the desired discrete time discount factor $\gamma \in [0,1)$.

Thus, for a deterministic policy π, the value $V^{\pi}: S \rightarrow \mathbb{R}$ is the expected reward at state s follows from Equation 2.26 with:

$$V^{\pi}(s) = \mathcal{R}(s, \pi(s)) + \sum_{s' \in S} \sum_{\tau=1}^{\infty} \gamma^{\tau} Pr(d\tau, s' | s, \pi(s)) V^{\pi}(s'), \tag{2.34}$$

with state transition $Pr(\tau, s'|s, a)$ and expected reward $\mathcal{R}: S \times A \rightarrow \mathbb{R}$ following from Equation 2.28:

$$\mathcal{R}(s, a) = R(s, a) + \sum_{\tau'=1}^{\infty} F(s, a, d\tau') \sum_{\tau=1}^{\tau'-1} \rho(s, a, \tau). \tag{2.35}$$

We may also define ρ in terms of $\rho': S \times A \times S \rightarrow [0, 1]$, as in Equation 2.29, with:

$$\rho(s, a, \tau) = \gamma^{\tau} \sum_{s' \in S} \rho'(s, a, s') T(s, a, \tau, s'). \tag{2.36}$$

The Bellman optimality equation also follows, using Equation 2.27, with:

$$V^{*}(s) = \max_{a \in A} \Big(\mathcal{R}(s, a) + \sum_{s' \in S} \sum_{\tau=1}^{\infty} \gamma^{\tau} Pr(d\tau, s' | s, a) V^{*}(s') \Big). \tag{2.37}$$

Lastly, we also assume for simplicity that $F(s, a, 0) = 0$, even though technically the general SMDP allows for a non-one probability of immediately regaining control. None of the models discussed use a non-zero F at $\tau = 0$. Again, this SMDP assumption is to prevent an infinite amount of reward from being generated in a finite amount of time.

2.4 Hierarchies and Multiple Objectives

As researchers began to implement MDP-related models in practice, they began to realize a single monolithic solution scales very poorly. Thankfully, these large problems tend to have natural structures amenable to decomposition into subproblems. Additionally, even within a single subproblem, otherwise complex reward functions in many cases may be decomposed into multiple simpler reward functions. As a result, a variety of hierarchical and multi-objective techniques exist, both in theoretical and engineering frameworks. This section covers the widely-used approaches.

2.4.1 Hierarchical Models

Designing agents that are deployed for a long duration inevitably engage in perhaps several decision-making tasks. Each of these subproblems may be complex, stochastic, and nuanced in their own right, even considering completely locally-focused objectives that contribute to an overall global objective. Hierarchical representations attempt to naturally describe this decomposition solution and enable an overall scalable solution to large problem domains, such as those faced by long-term autonomous agents.

2.4.1.1 The Options Framework

Options are special actions available to an agent that execute a complete policy, performing the actions of the agent until it stochastically returns control [122]. Each one can execute other options within this framework, allowing for a hierarchical policy defined by the interesting stochastic combination of many options. Options *expand* the space of policies by adding rich actions—the options themselves—in contrast with other methods that seek to reduce the space of policies (e.g., the next subsection) [7]. There are two primary types of **options**: Markov and semi-Markov.

For MDP $\langle S, A, T, R\rangle$, let $\mathcal{O} = \{\mathcal{O}_1, \ldots, \mathcal{O}_k\}$ denote the set of k options. Formally, a specific **Markov option** $\mathcal{O}_i \in \mathcal{O}$ is defined by the tuple $\langle \mathcal{I}_i, \pi_i, \beta_i \rangle$:

- $\mathcal{I}_i \subseteq S$ is a set of admissible initiation states of the option,
- $\pi_i : S \rightarrow A$ is a policy for the option, and
- $\beta_i : S \rightarrow [0,1]$ is the probability of terminating the option at each state.

Let $\mathcal{O}(s) = \{\mathcal{O}_i \in \mathcal{O} | s \in \mathcal{I}_i\}$ denote the set of options available at state s. In the options framework, a distinction is made between the set of *primitive actions* $A(s)$ and the set of options $\mathcal{O}(s)$, but both are viewed as the actions available to the agent $A(s) \cup \mathcal{O}(s)$ when it is in control. In fact, options generalize primitive actions, since for any action a, a one-step option can be created with $\mathcal{I}_i = \{s \in S | a \in A(s)\}$, $\pi_i(s) = a$ for all s, and $\beta_i(s) = 1$ for all s.

The options framework operates as a special class of discrete time SMDP. The agent selects actions, either primitive or option, the option is a fixed policy that produces rewards, at discrete time steps, following $R(s,a)$ if $a \in A$ is performed when the agent is in control and $R(s, \pi_i(s))$ if option $\mathcal{O}_i$ is in control. When the option is in control, it is following the SMDP's natural process. For discrete steps $\tau \in [\tau^t, \tau^{t+1})$ during the option $\mathcal{O}_i$'s control, $\rho(s, \mathcal{O}_i, \tau)$ denotes the expected reward rate at τ, given it started from state $s \in \mathcal{I}_i$. $F(s, \mathcal{O}_i, \tau)$ denotes the probability of terminating at discrete step τ following T, π_i, and β_i.

Similarly, the options framework allows for consideration of a **semi-Markov option** defined by the tuple $\langle \mathcal{I}_i, \pi_i, \beta_i \rangle$:

- $\mathcal{I}_i \subseteq S$ is a set of admissible initiation states of the option,
- $\pi_i : \bar{H} \rightarrow A$ is a policy for the option conditioned over any histories, and
- $\beta_i : \bar{H} \rightarrow [0,1]$ is the probability of terminating the option for any such history.

The only difference is that these options can model termination based on how long the option has been executed. The underlying model is still SMDP, except ρ and F use a natural process that changes based on the now semi-Markov history since the option was executed.

Without loss of generality, we consider deterministic policies; however, the policy may also be stochastic $\pi : S \times A \rightarrow [0,1]$, if desired. We can also consider policies over options with a policy form of $\pi : S \times \mathcal{O} \rightarrow [0,1]$ because any primitive action can be viewed as a one-step option. This enables reasoning about the more powerful hierarchical execution of options. This hierarchical execution induces a semi-Markov process that preserves any history of options which executed each other before they would have otherwise terminated. In most cases, this includes even the combination of Markov options.

Typically, the standard MDP $\langle S, A, T, R \rangle$ is augmented by adding (or subsuming) the action set with $\mathcal{O}$. The objective when using options is to maximize the expected reward as in (S)MDPs. As this is canonically a reinforcement learning agent, the formulation tends to focus on the expected reward at a state $s \in S$ for an option $\mathcal{O}_i \in \mathcal{O}$:

$$\mathcal{R}(s, \mathcal{O}_i) = \mathbb{E}\Big[R(s^{\tau}, \pi_i(s^{\tau})) + R(s^{\tau+1}, \pi_i(s^{\tau+1})) + \cdots + R(s^{\tau+\tau^t}, \pi_i(s^{\tau+\tau^t}))\Big| \mathcal{E}(\mathcal{O}_i, s, \tau)\Big] \tag{2.38}$$

with s^{τ}, $s^{\tau+1}$, ..., $s^{\tau+\tau^t}$ denoting random variables for the state at each time, τ^t denoting the sojourn time for this decision epoch t, and $\mathcal{E}(\mathcal{O}_i, s, t)$ denoting the event that option $\mathcal{O}_i$ is initiated in state s at time τ. Thus the expected reward can be written recursively as:

$$\mathcal{R}(s, \mathcal{O}_i) = R(s, \pi_i(s)) + \gamma \sum_{s' \in S} T(s, \pi_i(s), s')(1 - \beta_i(s'))\mathcal{R}(s', \mathcal{O}_i) \tag{2.39}$$

with a deterministic policy for consistency; as described above, stochastic policies follow a trivial change. The SMDP state transition $Pr(d\tau, s'|s, a)$ is not defined explicitly, since the focus has been on various reinforcement learning algorithms. Instead a surrogate state-prediction function $p^{\mathcal{O}_i} : S \times S \rightarrow \mathbb{R}$ for a state s, is referred to:

$$p^{\mathcal{O}_i}(s, s') = \sum_{k=1}^{\infty} Pr(d\tau, s'|s, \mathcal{O}_i)\gamma^k \tag{2.40}$$

which captures the probability of resuming control in state s after k steps with a measure of the discount included. The Bellman equation for a deterministic policy $\pi : S \to A \cup \mathcal{O}$ over actions and options is:

$$V^\pi(s) = \begin{cases} \mathcal{R}(s, \mathcal{O}_i) + \sum_{s' \in S} p^{\mathcal{O}_i}(s, s') V^\pi(s'), & \text{if } \pi(s) = \mathcal{O}_i \in \mathcal{O} \\ R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V^\pi(s'), & \text{otherwise} \end{cases}. \tag{2.41}$$

Again, options can generalize primitive actions by immediately returning control, but conceptually preserving their separation is important for consistency in this thesis.

2.4.1.2 Hierarchical Abstract Machine (HAM)

Hierarchical abstract machines (HAMs) represent a policy by a collection of specialized state machines which select actions and can call one another, with the agent's active decisions only at particular choice machine states [87, 88]. HAMs *reduce* the space of policies, in contrast with options which expand it by adding complex option actions [7].

For MDP $\langle S, A, T, R \rangle$, let $\mathcal{H} = \{\mathcal{H}_1, \ldots, \mathcal{H}_k\}$ denote the set of k HAMs. Formally, a specific **hierarchical abstract machine (HAM)** $\mathcal{H}_i \in \mathcal{H}$ is defined by the tuple $\langle X_i, \eta_i, \eta_i^0 \rangle$:

- X_i is a finite set of machine states, each as one of four types:
  - *action*: perform an action following partial function $\pi_i : X_i \times S \to A$, such that $\pi_i(x_i, s)$ induces an immediate reward and a state transition in the MDP at state s and action machine state x_i,
  - *call*: suspends $\mathcal{H}_i$ and executes another following partial function $\pi_i : X_i \times S \to \mathcal{H}$, such that $\pi_i(x_i, s) = \mathcal{H}_j$ calls machine $\mathcal{H}_j$ in $\mathcal{H}_i$'s call machine state x_i,
  - *choice*: choose a successor machine state *instead* of following η by *defining* partial function $\pi_i : X_i \times S \to X_i$, such that the agent chooses $\pi_i(x_i, s)$ at state s and choice machine state x_i, often from a subset $X_i(x_i, s) \subseteq X_i$ of valid successors (similar to $A(s)$), and
  - *stop*: terminate $\mathcal{H}_i$ and resume the previous HAM;
- $\eta_i : X_i \times S \times X_i \to [0, 1]$ is a stochastic machine state transition such that $\eta(x_i, s, x_i') = Pr(x_i' | x_i, s)$ is the probability of a transition to x_i' from machine state x_i in state s; and
- $\eta_i^0 : X_i \to [0, 1]$ is the initial state probability such that $\eta_i^0(x_i) = Pr(x_i)$ is the probability that the initial state is x_i, also stochastically determined whenever a machine is called.

This definition's notation differs slightly from previous work in an attempt to cleanly unify notation throughout the thesis. As such, π_i is overloaded for the machine state types. For simplicity, they assume the so-called *call graph* is a tree to prevent issues regarding loops, with actions being executed with probability 1 as $t \to \infty$. Similarly, they also assume the initial machine does not have a stop machine state.

The HAM method also operates as a special class of discrete time SMDP. The SMDP defines an augmented state space $S' = \mathcal{R}(\mathcal{H}) \times S$, with $\mathcal{R}(\mathcal{H})$ denoting all reachable machine states across all machines from the initial machine's possible initial states. S' records which HAM is currently being executed including its current machine state. The decision epochs are defined by the choice machine states, with available actions $A'(s') = X_i(x_i, s)$ for $s' = \langle x_i, s \rangle$ being the successor machine state, and induce a simple state transition changing the $\mathcal{H}$ state factor. Rewards $\rho(s', \pi_i(x_i, s), \tau)$ are generated at each discrete natural process time $\tau \in [\tau_1 + \cdots + \tau_t, \tau_1 + \cdots + \tau_{t+1})$ given the starting choice machine state was at s' and the choice's action followed some π_i. $F(s', \pi_i(x_i, s), \tau)$ similarly follows from the transitions T and various η_i, as well as the choice and call machine states.

2.4.1.3 Other Hierarchical Models

Options and HAMs will be the primary focus of analysis in this thesis because they represent two of the most popular approaches, they are both related to a grounded theoretical model (SMDPs), and they represent two different methodologies: options *expand* policies and HAMs *restrict* policies. Other hierarchical approaches are lightly discussed below.

Hierarchical Task Networks (HTNs) Among the first successful hierarchy-based structures were called *hierarchical task networks (HTNs)* [38]. The idea is deceptively simple: decompose a problem into parts called tasks. Tasks can call another and essentially terminate once it is complete. The HTN representation places an emphasis on the efficient decomposition and use of tasks within the network. In many ways, HAMs share this methodology. Traditionally, state-based planning—that is, a representation of state and sequence of actions to reach a goal state—via languages such as STRIPS [41] was a more "mathematically precise" model, but did not scale well beyond toy problems, in part because of their need for complete problem specification, limiting its real-world use. Conversely, HTNs were a more pragmatic "engineering" solution that worked for practical application, but in many cases lacked a strong theoretical foundation. However, a few attempts such as ABSTRIPS [108]—which generalized STRIPS to include abstractions—were successful in melding hierarchies into a mathematically-grounded planning model. In the late 1990's, Erol [37] finally proposed a unified formal view of the state-of-the-art HTNs such as NOAH [109] and NONLIN [125]. Similarly, this thesis proposes a unified formal model of state-of-the-art hierarchical approaches.

Other Approaches HAMs are related to a number of other models. For example, early work assumed a *landmark network* which consists of a set of landmark states, the set of neighbors for each landmark, and the knowledge of the nearest landmark for any given state [62]. The agent learns the quickest paths—in terms of number of steps—among them to reach a goal in a two-level hierarchy. HAMs can be thought of as an informal generalization of this approach, since the landmark states essentially describe choice machine states.

2.4.2 Multi-Objective Models

Hierarchical models address one challenge of designing agents for long-term autonomy in the real-world: scalability via decomposition into distinct smaller problems. Another common challenge inevitably arises in these systems: they typically are concerned about more than one objective in the same problem. For example, we consider semi-autonomous vehicles that must trade-off total travel time and how much of this time is spent autonomous. However, numerous real-world domains must weight similar trade-offs, such as in manufacturing that considers time, cost, and quality of the products.

The **multi-objective MDP (MOMDP)** [103] is a sequential decision-making model defined by the tuple $\langle S, A, T, \mathbf{R} \rangle$:

- S is a finite set of states,
- A is a finite set of actions,
- $T : S \times A \times S \rightarrow [0,1]$ is a state transition function such that $T(s,a,s') = Pr(s'|s,a)$ is the probability of successor state s' given action a was performed in state s, and
- $\mathbf{R} = [R_1, \ldots, R_k]^T$ is a vector of reward functions $R_i : S \times A \rightarrow \mathbb{R}$, each denoting an immediate reward $R_i(s,a)$ for performing action a in state s.

The stochastic process operates exactly like an MDP, except that multiple immediate rewards are experienced at each time step. Notion requires k-dimensional vectors following in the natural way from rewards denoted $\mathbf{R}(s,a) = [R_1(s,a), \ldots, R_k(s,a)]^T$. The definitions of **policy** are the same, with stationary policies defined as $\pi : S \rightarrow A$, and even stochastic policies defined as $\pi : S \times A \rightarrow [0,1]$.

For **infinite horizon MOMDP**, the ideal objective is to find a policy π that maximizes all of the expected rewards over all states:

$$\mathbb{E}\Big[\sum_{t=0}^{\infty}\gamma^t \mathbf{R}(s^t,\pi(s^t))\Big|\pi\Big] \tag{2.42}$$

with s^t denoting a random variable for the state at time t generated following T, and discount factor $\gamma \in [0,1)$ assumed to be identical for all objectives for simplicity. For a policy π, the **value** function is now a vector of values $\mathbf{V}^{\pi}: S \to \mathbb{R}^k$ with Bellman equation for state s:

$$\mathbf{V}^{\pi}(s) = \mathbf{R}(s,\pi(s)) + \gamma \sum_{s' \in S} T(s,\pi(s),s')\mathbf{V}^{\pi}(s'). \tag{2.43}$$

It is rare that a single policy exists which obtains the maximal value for all objectives. Instead, we consider different objectives which explicitly trade-off the individual expected rewards. However, the evaluation of a policy over all objectives in Equation 2.43 is useful to all approaches. To resolve the trade-offs, there are two main forms of resolution: scalarization and constraint. Scalarization methods attempt to merge the k objectives into a single objective problem, which can be readily solved using the standard techniques. Constraint methods attempt to solve the k objectives as constraints, either in a sequence or simultaneously.

2.4.2.1 Scalarization and Pareto Optimality

Scalarization defines a function that maps the vector of values or rewards in Equation 2.43 to a single value or reward. Formally, a **scalarization function** $f_{\mathbf{w}}: \mathbb{R}^k \to \mathbb{R}$ is parameterized by some weight vector $\mathbf{w}$ such that $V^{\pi}_{\mathbf{w}}(s) = f_{\mathbf{w}}(\mathbf{V}^{\pi}(s))$ defines a single value function through this scalarization. An important special case is *linear scalarization* $f_{\mathbf{w}}(\mathbf{V}^{\pi}(s)) = \mathbf{w}^T\mathbf{V}^{\pi}(s)$. Within this representation, there are essentially two classes of algorithm, either the parameters $\mathbf{w}$ are given or they are unknown [129]. If parameters are given, then a single a optimal policy can be computed [84]. If parameters must be computed, then a set of policies must be computed that represent a notion of the landscape of trade-offs among optimal policies [104].

Pareto optimality is widely used to solve multi-objective problems [24]. Formally, a policy π **Pareto dominates** another policy π' if for all i, $V^{\pi}_i(s) \geq V^{\pi'}_i(s)$ and there exists some j such that $V^{\pi}_j(s) > V^{\pi'}_j(s)$. For MOMDPs, it turns out to refer to a particular class of scalarization functions that *strictly monotonically increase* [103]. When parameters $\mathbf{w}$ are unknown, we return the set of policies that are not Pareto dominated by any other policy called the **Pareto frontier**:

$$\Pi^{PF} = \{\pi \in \Pi | \forall \pi' \neq \pi, \forall i, V_i^\pi(s) \geq V_i^{\pi'}(s) \wedge \exists j \text{ s.t. } V_j^\pi(s) > V_j^{\pi'}(s)\}. \tag{2.44}$$

Interestingly, instead of computing all Pareto optimal deterministic policies, it is sufficient to instead return the convex coverage set (CCS) [103]. This realization has led to a growing number of algorithms which compute or approximate the CCS [104].

In summary, the result of scalarization methods is either a single objective function, which uses standard techniques to determine any optimal policies, or a set of policies, for example defined by a CCS representing the Pareto frontier. We will discuss scalarization within the thesis' presented models, but leave any detailed analysis of this particular multi-objective approach to future work outside this thesis.

2.4.2.2 Constraints and Lexicographic Orderings

A constrained MDP (CMDP) instead defines a single objective to maximize with a set of constraints the agent must keep under a constant value [39, 2]. Formally, for all objectives $1 \leq i \leq k-1$, let rewards be written as costs $C_i(s,a) = -R_i(s,a)$, with a *minimization* objective, and assume a constant scalar c_i is given for each. The objective in a **constrained MDP** [2] is to find a policy π:

$$\begin{aligned} &\text{maximize} \quad V_k^\pi(s^0) \\ &\text{subject to} \quad V_i^\pi(s^0) \leq c_i, \quad \forall i \in \{1, \ldots, k-1\}. \end{aligned} \tag{2.45}$$

This can be equivalently written in terms of sets of policies:

$$\Pi_i = \{\pi \in \Pi | V_i^\pi(s^0) \leq c_i\}, \forall i \in \{1, \ldots, k-1\} \quad \text{and} \quad \Pi_k = \{\pi \in \Pi_1 \cap \cdots \cap \Pi_{k-1} | V_k^*(s^0) = V_k^\pi(s^0)\}$$

$$\text{such that} \quad \pi^* \in \Pi_k \subseteq (\Pi_1 \cap \cdots \cap \Pi_{k-1}), \tag{2.46}$$

with optimal policy π^*. The CMDP was discussed for upwards of a decade [39] before it was completely described by Altman [2]. To address scalability, modern methods can employ state clustering to solve a smaller CMDP with less variables, with successful implementation on a real robot [40]. Early references to a constrained POMDP (CPOMDP) are found in work surrounding uncountably infinite state space constrained MDPs [93]. Eventually, the idea was formalized with

CPOMDP solution dynamic programming algorithms [57] and linear programming with finite state controller policies [96].

Instead of the breadth of constraints in CMDPs, they can instead be ordered. A lexicographic (ordinal) MDP (LMDP) defines an ordering over the objectives and iteratively constrains the policies. In essence, the set of policies is iteratively constrained using successor objectives as tie-breakers. Formally, the objective of a **lexicographic (ordinal) MDP (LMDP)** [43] is to find a policy π following the recursive objective for any $i > 1$:

$$\begin{aligned} \text{maximize} \quad & V_i^\pi(s^0) \\ \text{subject to} \quad & V_j^\pi(s^0) = V_j^*(s^0), \quad \forall j \in \{1, \ldots, i-1\}, \end{aligned} \tag{2.47}$$

Importantly, the V_j^* above refers to the optimal value constrained to the policies available Π_j. V_1^* is unconstrained and selects optimal values obtainable from $\Pi_1 = \Pi$. Again, this can be equivalently written in terms of sets of policies:

$$\Pi_1 = \Pi \quad \text{and} \quad \Pi_i = \{\pi \in \Pi_{i-1} | V_i^*(s^0) = V_i^\pi(s^0)\}, \forall i \in \{2, \ldots, k\},$$

$$\text{such that} \quad \pi^* \in \Pi_k \subseteq \Pi_{k-1} \subseteq \cdots \subseteq \Pi_1 = \Pi, \tag{2.48}$$

with optimal policy π^*. Obviously, ties are actually quite rare in the policy space, and leaves very little room for successor objectives to optimize their own objectives. The original work on ordinal dynamic programming for MDPs dates back to the 1970's within the field of operations research and management science. It was proven that value iteration converges to a stationary policy in finite horizon [81] and infinite horizon [114]. Within the context of reinforcement learning, the original early work by Gabor et al. [43] assumed a maximum threshold value v_i^{max} was provided. They modified the Bellman (optimality) equation to be $V_i'(s) = \min\{V_i^\pi(s), v_i^{max}\}$ in the hope of including more policies for successor objectives in the ordering. However, it is unknown if v_i^{max} is obtainable for V_i, and it is unclear how this affects the theoretical underpinnings of the MDP itself.

2.5 Conclusion

We have presented the core MDP model and a multitude of models based on it. While each model does expresses a solution to an important class of distinct problem, it also highlights the need for a unified perspective due to the quantity of these disparate approaches. In the next chapter we present policy networks as a unified model that encompasses these prior models as well as the other three novel models presented in this thesis.

CHAPTER 3
POLICY NETWORKS

This chapter defines the mathematical model that unifies all chapters in this thesis: policy networks. The model is presented with select examples, including a complete solution to a mobile home healthcare robot. A discussion about the relation to probabilistic graphical models is provided, as well as similarly convenient graphical representations of policy networks. The following chapter will discuss how policies can be generally represented within a policy network. The subsequent three chapters then present three seemingly distinct important models, each one providing a formal mapping that they are generalized by policy networks.

3.1 Introduction

Over the past decade, sequential decision-making models have been increasingly deployed in large-scale domains with high societal impact, ranging from aircraft collision avoidance [67] to autonomous vehicles [134]. While these systems have enjoyed rapid growth, they relied on a fragmented collection of specialized approaches that combine either multiple objectives [2, 66] or multiple models by hierarchical abstraction [122, 91] or by integrating their actions online [5, 134].

Each one of these solutions introduces an important reasoning capability, but to support long-term autonomy in the real world, we increasingly need to integrate multiple capabilities within one system. As Marvin Minsky observed, “the power of intelligence stems from our vast diversity, not from any single, perfect principle” [80]. It is unlikely that any single MDP model will suffice. For the sake of scalability and computational efficiency, we need new formal ways to facilitate the integration of multiple models within a single agent. To this end, we propose a novel framework called policy networks that unifies prior approaches, offers new insights, and provides a solid foundation on which to build the next generation of large-scale decision models.

We consider a home healthcare robot domain for household and eldercare scenarios [91]. The robot must perform a wide array of helpful tasks (e.g., medicine delivery and cleaning), plan safe paths around the house, and detect falls to call for help as needed [21] while operating over a long duration. This domain has many subproblems, each complex and nuanced, and they are all interrelated as part of the whole solution. Systems of this scale require an integration of multiple methods, as the range of subtasks quickly becomes too large to solve with a single monolithic MDP. Additionally, as the number of subtasks increases, the reward function becomes conflated. This complicates the design and maintenance of a weighted reward that must balance all the subtasks under one function, losing any semantic meanings of the reward in the process. These two primary concerns, scalability and conflation of reward, are alleviated by the use of a policy network.

Prior work on integrations of multiple models arose from disparate ideas, each of which extends the MDP model in a particular way. In hierarchical planning, a large problem is decomposed into essentially subtasks [122, 124]. In planning with multiple objectives, which we show to be related, solutions typically scalarize the objectives into one or maximize a primary objective subject to constraints [104, 66]. Online techniques that employ multiple models allow different models to update their states simultaneously and recommend actions for each entity in the domain [67, 5].

While these approaches have been used in modest applications, it is not clear how they relate or how to combine them to solve large-scale problems. This knowledge gap manifests itself by the lack of a unified view across all model forms, which leaves many questions unanswered. For example, how are constrained MDPs (CMDPs) related to options and can the two approaches be combined? What does it mean to perform an action if it can induce updates in multiple models? How can multiple models transfer control to one another if their state and/or action spaces are different? More generally, how can we create a principled mathematical framework that enables the integration of multiple models into a single, coherent decision-making process with well-defined properties?

We propose the notion of a policy network [139, 141] that helps us begin to answer these questions. It is a graph in which the vertices denote a set of policies and the edges denote their dependencies. A set of policies associated with a vertex refers to a state and action space that can be shared or distinct from any other vertices. A policy constraint edge enforces a restriction on a vertex's set of policies from another vertex. A policy transition edge defines a state transition in a vertex's state space or a transfer of control to another vertex. The objective is to maximize expected reward in the induced hierarchy of constrained semi-Markov decision processes following the graph's dependency structure.

Our primary contributions are: (1) a formal definition of policy networks and their properties; (2) a theoretical analysis that proves their generality, encapsulating prior models such as CMDPs and options; and (3) an implementation on a home healthcare robot acting in a real household environment.

3.2 Policy Networks

In general, policy networks are graphs in which vertices denote sets of policies for a reward function and edges denote policy dependences among them. The objective is to capture the relations among distinct decision-making components to solve large multi-objective hierarchical problems. Thus, a **policy network** is a sequential decision-making model defined by a directed graph $\langle V, E \rangle$:

- V is a set of vertices such that $v \in V$ refers to policy set Π_v and reward $R_v : S_v \times A_v \rightarrow \mathbb{R}$, and
- E is a set of edges such that $\langle v, w \rangle \in E$ forms a dependence of w on v, with optional properties:
  - policy constraint Π_{vw} enforces that $\pi_w \in \Pi_{vw}$, for the policy $\pi_w \in \Pi_w$ chosen for w; and/or
  - policy transition $T_{vw} : S_v \times A_v \times S_w \rightarrow [0, 1]$ is a partial function for $Pr(w, s'_w | v, s_v, a_v)$.

The execution of a policy network operates over discrete time steps $t \in \mathbb{N}$ as a form of *Markov multi-reward process*. Each vertex v has a state space S_v and action space A_v for its policy and reward. Its state space S_v has an initial state $s_v^0 \in S_v$. Each edge $e = \langle v, w \rangle$ makes w inherently depend on v such that when v performs an action or transitions its state it can affect w. Additional dependency properties can also be added to an edge, such as policy constraint or transition. This process is formalized in the paragraphs below.

As in (PO)MDPs, to **perform** an action is simply the act of conditioning on the action so as to induce an update in the underlying vertex v's stochastic process following the distribution $Pr(w, s'_w | v, s_v, a_v)$; this is called a **state transition**. This probability distribution describes the state transition within the state space of v (i.e., $w = v$ and $s'_w = s'_v \in S_v$) as well as across other state spaces used by other vertices (i.e., $w \neq v$ and $s'_w \in S_w$). Policy networks require full specification of $Pr(w, s'_w | v, s_v, a_v)$ via the collection of functions T_{vw}. In its simplest form, if v only transitions to itself by T_{vv}, then T_e is equivalent to a typical (PO)MDP state transition. Performing an action also induces a reward from R_v.

At each time step, a **controller** vertex v performs the action $\pi_v(s_v) \in A_v$ at its current state $s_v \in S_v$ from a policy $\pi_v \in \Pi_v \cap (\bigcap_w \Pi_{wv})$ chosen for v. Each policy network has an initial controller $v^0 \in V$. The actions performed by v may result in a state transition to a different vertex w's state space. We call this a **transfer of control**, with the controller changing from v to w who now performs actions following its policy. Obviously, only a controller vertex can transfer control. If a non-controller vertex performs a state update, transfer of control attempts instead result in a self-loop state transition.

For any edge $\langle v,w\rangle \in E$, there is an inherent dependence that w has on v. First, if v is the controller and the state spaces are shared $S_w = S_v$, then w also follows the state transition result of v (i.e., $s_w^{t+1} = s_v^{t+1}$). Second, if v is the controller and the action spaces are shared $A_w = A_v$, then w also performs the action that v performs (i.e., $a_w^t = a_v^t$); performing an action in this way also emits a reward R_w for w. Thus, if $S_w = S_v$ and $A_w = A_v$ then w performs action $a_w^t = a_v^t$ and the successor state is $s_w^{t+1} = s_v^{t+1}$. However, if $S_w \neq S_v$ and $A_w = A_v$ then w still performs action $a_w^t = a_v^t$ and induces a state transition as normal, with a caveat that any transfer of control attempt self-loops instead. Any additional dependences can be optionally added to an edge as well. This thesis primarily considers two: policy constraint and policy transition.

Policy networks may appear either deceptively simple and/or nuanced, so we will first consider a few step-by-step examples of the policy network's stochastic reward process. Not all of the details are provided at this point in the chapter, specifically the general meaning of value, optimality, stationarity, and specifics of the graphical representation referenced. *Importantly*, the purpose is mainly to provide examples of the stochastic process and interaction of models. The specifics of value, optimality, stationarity, and the graphical representation will be precisely defined in the subsequent sections. We invite the reader to re-read these examples again after each of these concepts are defined.

Example (MDP Represented as a Policy Network) Consider a policy network with $V = \{v\}$ and $E = \{e_{vv} = \langle v,v\rangle\}$, as in Figure 3.1 (a), such that e_{vv} self-loops with a transition T_{vv}. Let T_{vv} be $Pr(v, s_v' | v, s_v, a_v) = Pr(s_v' | s_v, a_v)$ define a classical MDP state transition.

The stochastic process operates with vertex v in control at each time t. In the classic case of MDP optimal control or planning, we compute an optimal stationary policy from the space of available policies $\pi_v^* \in \Pi_v$. During execution, this stationary policy performs action $\pi_v^*(s_v^t)$ at time t's state s_v^t. By definition, performing an action conditions on A_v at time t, inducing a state update following T_{vv}. The stochastic process generates successor s_v^{t+1} from T_{vv} and emits reward $R_v(s_v^t, \pi_v^*(s_v^t))$. This process repeats ad infinitum, recreating the MDP inside a policy network.

Example (Edge Dependences; Performing Action; Simple CMDP) Consider a policy network with $V = \{v, w\}$ and $E = \{e_{vv}, e_{ww}, e_{vw}, e_{wv}\}$, as in Figure 3.1 (c), with v and w having the same state-action spaces (i.e., $S_v = S_w$ and $A_v = A_w$). Let e_{vv} and e_{ww} be as in the first example for both v and w. Let the initial controller be w.

Also, let us define two dependences: e_{vw} and e_{wv}. By definition of an edge, since v and w share state-action spaces here, any performed action or state transition in one also occurs in the other.

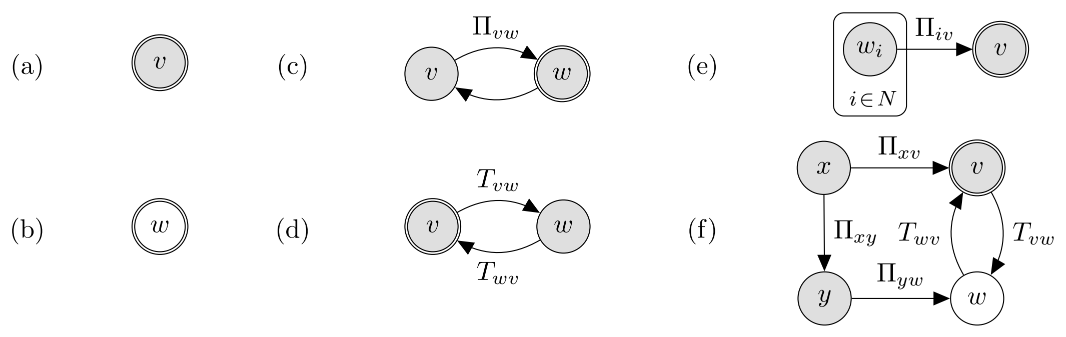


Figure 3.1: Basic examples of the graphical notation described in Section 3.4.1 that is used to represent policy networks, with each $v \in V$ following some model $v \sim MDP(S_v, A_v, T_v, R_v)$: (a) stationary vertex; (b) non-stationary vertex; (c) constraint edge; (d) transfer of control edges; (e) plate notation for a set of N constraints; and (f) mixture of the concepts.

Additionally, to add a CMDP constraint, let e_{vw} include a policy constraint defined by the set $\Pi_{vw} = \{\pi \in \Pi_v | -V_v^\pi(s^0) \leq c_v\}$ to ensure the only policies w can choose are within some constraint value c_v for V_v, as in a CMDP constraint. See the Theoretical Analysis section for details.

The stochastic process operates with vertex w in control at each time step t. It uses policy $\pi_w^* \in \Pi_w \cap \Pi_{vw}$—constrained by v's c_v—and performs action $\pi_w^*(s_w^t)$. Importantly, there is an edge from w to v. Since action spaces are shared, when w performs an action so too does v, with $a_v^t = a_w^t = \pi_w^*(s_w^t)$. By definition of performing an action, both v and w emit rewards $R_v(s_v^t, a_v^t)$ and $R_w(s_w^t, a_w^t)$, respectively. Since state spaces are shared, when w has a state transition so does v, with $s_v^{t+1} = s_w^{t+1}$. This process repeats ad infinitum, recreating a simple CMDP inside a policy network.

Example (Transferring Control; Simple Option) Consider a policy network with $V = \{v, w\}$ and $E = \{e_{vv}, e_{ww}, e_{vw}, e_{wv}\}$, as in Figure 3.1 (d), with v and w having the same state and action spaces. Again let e_{vv} and e_{ww} be as in the first example for both v and w, respectively. Also, this forms a dependence as in the second example.

Additionally, to add transfer of control, let e_{vw} include a policy transition T_{vw} such that we have $Pr(w, s'|v, s, a^+) > 0$ only for a particular transfer of control action a^+ that transfers control to w with probability 1 and performs w's recommended action $\pi_w^*(s)$ as a transition. Finally, let e_{wv} be a similarly defined transfer of control.

To create a simple Markov option, let us define w's available policy space to be a single policy $\Pi_w = \{\pi_w\}$, the option's fixed policy π_w itself. See the Theoretical Analysis section for details.

The stochastic process operates with vertex v *initially* in control at each time step t. It uses policy $\pi_v^* \in \Pi_v$ and performs action $\pi_v^*(s^t)$. Normally, if the performed action is *not* the transfer of control action (i.e., $\pi_v^*(s^t) \neq a^+$), then the controller remains v as in a normal MDP. Importantly, if the performed action *is* transfer of control action (i.e., $\pi_v^*(s^t) = a^+$), then the new controller becomes w following T_{vw}. In this case, at the next time step $t+1$, the state is now s^{t+1} and w uses its policy $\pi_w \in \Pi_w$ to performs action $\pi_w(s^{t+1})$. Of course, at any time w can also choose a^+ to transfer control back to v following T_{wv}. This process repeats ad infinitum, recreating a simple option inside the policy network.

3.2.1 Stationarity

To this point, policies π and policy sets Π have been considered without a dependence on time. Generally speaking, our goal will be to construct policy networks that have all policies fixed in place over all time. However, this need not be the case. Here we define three kinds of so-called stationarity, discuss them, and provide examples of non-stationarity to contrast with the examples above.

In the tradition of MDPs and planning, policies are commonly stationary—that is, they do not change over time. Although, even solutions computed offline can have time-varying non-stationary policies. In any case, policy networks share this policy property. Formally, if a vertex's policy $\pi_v^t = \pi_v$ for all time t, then we call it a **stationary policy**. Otherwise, we call it a **non-stationary policy**. The holistic perspective of policy networks affords a broader view of stationarity that also includes the behavior of *online* algorithms, which vary their policy over time in online planning [144] and reinforcement learning [121]. We leave the analysis of online scenarios for future work.

Since policy networks relate sets of policies to one another, the set of policies Π_v^t, as well as any Π_{wv}^t, at a time t can also remain constant or vary over time. Formally, if a vertex's policy set $\Pi_v^t = \Pi_v$ and $\Pi_{wv}^t = \Pi_{wv}$ for all constraint edges $\langle w, v\rangle$ and for all time t, then we call it a **stationary policy set**. Otherwise, we call it a **non-stationary policy set**. In a simple MDP or POMDP within a policy network, the policy set trivially remains constant. However, in a growing number of online *models*—such as MODIA used in autonomous vehicles [134] as discussed in Chapter 7—the set of policies is constantly adjusted online. While this is easily described in a policy network, their formal analysis is nuanced and specific to the assumptions for each online scenario. For this reason, we leave any analysis of these online scenarios to future work.

If a vertex's policy and policy set are both stationary, then we call it a **stationary vertex**. Otherwise, we call it a **non-stationary vertex**.

Interestingly, a non-stationary *policy set* can still retain a stationary *policy*; that is, in some cases they can be mutually exclusive stationary properties. However, as is in the case of MODIA, non-stationary policy sets can force the policy to be non-stationary, causing the vertex to be non-stationary. This is described with MODIA in Chapter 7. In any case, we now show brief examples of non-stationary policies and policy sets.

Example (Non-Stationary Policy) The first example in this section considered an MDP following an optimal stationary policy. Consider again the same policy network as above with $V = \{v\}$ and $E = \{e_{ww} = \langle w, w\rangle\}$, as in Figure 3.1 (b), such that e_{ww} self-loops with a transition T_{ww}. Let T_{ww} be $Pr(w, s'_w | w, s_w, a_w) = Pr(s'_w | s_w, a_w)$ define a classical MDP state transition.

The stochastic process operates with vertex w in control at each time t. In the classic case of MDP online planning or reinforcement learning, the policy is learned over time and thus is not stationary, but is still chosen at each time t from the space of available policies $\pi^t_w \in \Pi_w$. During execution, this non-stationary policy performs action $\pi^t_w(s^t_w)$ at time t's state s^t_w. As above, performing an action conditions on A_w at time t, inducing a state update following T_{ww}. The stochastic process generates a successor s^{t+1}_w from T_{ww} and emits reward $R_v(s^t_v, \pi^t_v(s^t_w))$. This process repeats ad infinitum, demonstrating a non-stationary policy and recreating an online planning or reinforcement learning algorithm's execution inside a policy network.

Example (Non-Stationary Policy Set) Consider a policy network with $V = \{w_i\} \cup \{v\}$ and $E = \{e_{w_i w_i}, e_{vv}, e_{w_i v}, e_{v w_i}\}$, as in Figure 3.1 (e), with w_i and v having the different state spaces but the same action spaces (i.e., $S_{w_i} \neq S_v$ and $A_{w_i} = A_v$). Specifically, let $S_v = \{s_v\}$ be a single state. Let $e_{w_i w_i}$ be a standard MDP state transition $T_{w_i w_i}$ as in the example above. Let e_{vv} have a trivial self-loop state transition T_{vv} because there is only one state. Let the initial controller be v.

Also, let us define two dependences: $e_{v w_i}$ and $e_{w_i v}$. By definition of an edge, since v and w_i share action spaces, any action performed by v induces a state transition and emits a reward in all.

Lastly, add a CMDP-like time-dependent policy constraint edge $e_{w_i v} \in E$ defined by the set $\Pi^t_{w_i v} = \{\pi \in \Pi_v | V^*_{w_i}(s^t_{w_i}) - Q^*_{w_i}(s^t_{w_i}, \pi(s_v)) \leq \underline{Q}, \forall s_v \in S_v\}$. This ensures that the only policies (actions) v can choose are within some slack constraint value $\underline{Q}$ for V_{w_i} current state $s^t_{w_i}$. In other words, it restricts actions to be within a one-step slack—that is, allowable deviation from an optimal action. This is how MODIA works, as described in Chapter 7. The nuances of slack itself are explored in Chapter 5. In any case, this particular policy constraint edge is defined in Equation 3.18.

The stochastic process operates with vertex w in control at each time step t. It uses policy $\pi^t_v \in \Pi^t_v \cap (\bigcap_{w_i} \Pi^t_{w_i v})$—constrained by v's $\underline{Q}$—and performs action $\pi^t_v(s^t_v)$. Note that $s^t_v = s_v$ for all t because $S_v = \{s_v\}$; what changes are the available actions for v to select. Importantly, there is an edge from v to each w_i. Since action spaces are shared, when v performs an action so too do all w_i, with $a^t_v = a^t_{w_i} = \pi^t_v(s_v)$. By definition of performing an action, both v and all w_i emit rewards $R_v(s_v, a_v^t)$ and $R_{w_i}(s_{w_i}^t, a_{w_i}^t)$, respectively. Since state spaces are different, each w_i has an individual state transition to some $s_{w_i}^{t+1}$ following $T_{w_i w_i}$. By construction of the policy constraint edge $e_{w_i v}$, each $\Pi_{w_i v}^{t+1}$ uses its new state $s_{w_i}^{t+1}$ to update the actions available to v via its new policy set $\Pi_v^{t+1} \cap (\bigcap_{w_i} \Pi_{w_i v}^{t+1})$. This process repeats ad infinitum, demonstrating a non-stationary policy set and recreating MODIA's (as described in Chapter 7) execution inside a policy network.

3.2.2 Underlying Markov Multi-Reward Process

One of the primary benefits of policy networks is that the underlying complex mathematics can be abstracted by simpler graphical notation. This is the same utility provided by the conventional use probability theory and Bayesian networks; we do not need to describe simple probabilities in terms of topological spaces, Boreal sets, measure theory, and σ-algebras. Just as probability theory has this deeper mathematical foundation, so too do policy networks have one for their state in the form of an underlying Markov multi-reward process.

Now that we have described the policy network model with a multitude of examples, we more concretely formalize this underlying Markov multi-reward process. The formalized stochastic process' state transition will be useful when defining optimality in the next section. Consequently, we provide a generalized definition of this formalized state transition that allows us to consider the Markov multi-reward process of any subset of vertices $X \subseteq V$. In this subset X of vertices, any transfer of control outside X, that is to vertices in $V - X$, is treated as a self-loop instead. This enables a clean statement of optimality in Section 3.3, which iteratively grows the set X towards V following a dependency graph, solving a constrained semi-MDP at each step. Thus the full underlying Markov multi-reward process is obtained when $X = V$.

First, it is useful to define a transition function μ to represent the collection of $Pr(w, s'_w | v, s_v, a_v)$ with the added ability to capture self-loops for transitions outside a given set of vertices $X \subseteq V$. Formally, let (partial) function $\mu : 2^V \times V \times (\bigcup_v S_v) \times (\bigcup_v A_v) \times V \times (\bigcup_v S_v)$ follow:

$$\mu(X, v, s_v, a_v, w, s'_w) = \begin{cases} Pr(w, s'_w | v, s_v, a_v), & \text{if } v \in X \wedge w \in X \wedge \neg(w = v \wedge s'_w = s_v) \\ Pr(w, s'_w | v, s_v, a_v) & \\ + \sum_{u \notin X} \sum_{s'_u \in S_u} Pr(u, s'_u | v, s_v, a_v), & \text{if } v \in X \wedge w \in X \wedge (w = v \wedge s'_w = s_v) \\ [w = v \wedge s'_w = s_v], & \text{if } v \notin X \end{cases} . \quad (3.1)$$

While this may look notationally complex, it performs a simple operation: (1) transition as normal for vertices in X, (2) add the extra probability weight to self-loop for transitions to vertices outside X, and (3) any vertex outside X self-loops. Observe that if $X = V$ then it reduces to $\mu(V, v, s_v, a_v, w, s'_w) = Pr(w, s'_w | v, s_v, a_v)$.

Finally we can define the policy network's underlying stochastic process. As mentioned above, the full Markov multi-reward process includes transitions among all vertices $X = V$; however, we define it here conditioned on X for use in the Section 3.3. Also, we analyze the process given some collection of policies $\pi = \langle \pi_{x_1}, \ldots, \pi_{x_{|X|}} \rangle$. We will first write the process with stationary policies and discuss non-stationarity afterward. It has the state space $\mathbf{S} = X \times S_{x_1} \times \cdots \times S_{x_{|X|}}$ denoting the current controller and the state of all vertices. The stochastic process has the Markov chain $\mathbf{S}^1$, $\mathbf{S}^2$, $\mathbf{S}^3$, ... over time $t \in \mathbb{N}$, with $\mathbf{S}^t$ denoting the random variable for the state. At each time step, the Markov chain's state transition only depends on the current state $\mathbf{s} = \langle i, s_{x_1}, \ldots, s_{x_{|X|}} \rangle$, transitioning to a successor state $\mathbf{s}' = \langle i', s'_{x_1}, \ldots, s'_{x_{|X|}} \rangle$ following the $|\mathbf{S}|$-by-$|\mathbf{S}|$ stochastic matrix $M_{X\pi}$:

$$M_{X\pi}(\mathbf{s}, \mathbf{s}') = \prod_{v \in X} M^v_{X\pi}(\mathbf{s}, \mathbf{s}'), \tag{3.2}$$

multiplying each vertex v's individual state transition probability $M^v_{X\pi}$ from state s_v to vertex w's state s'_w, given that i is currently in control and i' is in control next:

$$M^v_{X\pi}(\mathbf{s}, \mathbf{s}') = \begin{cases} \mu(X, i, s_i, \pi_i(s_i), v, s'_v), & \text{if } i' = v \\ [s'_v = s'_{i'}], & \text{if } i' \neq v \wedge \exists \langle i', v \rangle \in E \wedge S_v = S_{i'} \\ \mu(\{v\}, v, s_v, \pi_i(s_i), v, s'_v), & \text{if } i' \neq v \wedge \neg(\exists \langle i', v \rangle \in E \wedge S_v = S_{i'}) \wedge \exists \langle i, v \rangle \in E \wedge A_v = A_i \\ [s'_v = s_v], & \text{otherwise} \end{cases}. \tag{3.3}$$

Again, this notational complexity belies the simpler meaning of Equation 3.3: (1) transition as normal if the vertex is in control (as in MDPs), (2) copy the transition of the connected vertex when it is the controller and updates its state (as in options, CMDPs, LMDPs, and TMDPs), (3) self-transition within the vertex's own state space when a connected vertex is the controller and performs a shared action (as in MODIA), and (4) self-loop when not in control ("paused") and not connected to a controller vertex that shares actions (as in SAS). These concepts were described in the definition of policy networks. This is merely the formal statement of this behavior.

At each time step, a reward is emitted for the controller vertex i after performing the action $\pi_i(s_i)$. Additionally, a reward is emitted for any connected vertex to i that shares the same action space. Formally, from controller i, the set $X^R_i \subseteq X$, denoting all vertices $v \in X^R_i$ that emit rewards $R_v(s_v, \pi_i(s_i))$, is:

$$X^R_i = \Big\{ v \in X | v = i \vee (\exists \langle i, v \rangle \in E \wedge A_v = A_i) \Big\}. \tag{3.4}$$

3.2.3 Relative Markov Reward Process

To analyze a single vertex v's reward, we need not consider the Markov multi-reward process in its entirety. Instead, we consider just the local region directly connected to the vertex that shares the same action space. Only this subset of vertices can induce a reward R_v to be emitted for the vertex.

In an effort to simplify notation, any variable or function defined here that has an *overline* or *underline* is *relative* to a vertex v, the set of vertices X, and the policies π. We assume these three are given to define the relative Markov reward process below. For example, $\overline{X}$ and $\underline{X}$ are subsets of X relative to v. Similarly, $\overline{M}$ and $\underline{M}$ are matrices, and $\underline{\lambda}$ is a function, with states defined using X and their probabilities defined relative to v and π.

The relative Markov reward process for $v \in V$ operates under the vertices $\overline{X} \subseteq X$:

$$\overline{X} = \Big\{ w \in X | w = v \vee (\exists \langle w, v \rangle \in E \wedge A_w = A_v) \Big\} \tag{3.5}$$

with the other vertices defined as $\underline{X} = X - \overline{X}$, forming a partition $\{\overline{X}, \underline{X}\}$ over X. We partition $M_{X\pi}$ into two distinct Markov chains with $|\mathbf{S}|$-by-$|\mathbf{S}|$ stochastic matrices $\overline{M}$ and $\underline{M}$. The latter $\underline{M}$ describes transitions within the other vertices in $\underline{X}$ and how control is returned to $\overline{X}$. The former $\overline{M}$ describes transitions within the local vertices in $\overline{X}$ surrounding and including v.

The two stochastic matrices are linked by a **summarization function** $\underline{\lambda} : \mathbf{S} \times \mathbf{S} \rightarrow [0,1]$ at the stochastic transition boundary between $\overline{X}$ and $\underline{X}$. As the focus is on rewards generated in $\overline{X}$, $\underline{\lambda}$ is defined using $\underline{M}$ to define $\overline{M}$. This summarization function $\underline{\lambda}$ equals the probability of returning to the particular states in $\overline{X}$ vertices from any starting state in X after stochastically traversing through states in $\underline{X}$. Trivially, if the starting state is already in $\overline{X}$ then $\underline{\lambda}$ is simply probability 1 at the starting state.

Formally, let $|\mathbf{S}|$-by-$|\mathbf{S}|$ stochastic matrix $\underline{M}$ be defined for starting state $\mathbf{s} = \langle i, s_{x_1}, \ldots, s_{x_{|X|}} \rangle \in \mathbf{S}$ and resulting state $\mathbf{s}' = \langle i', s'_{x_1}, \ldots, s'_{x_{|X|}} \rangle \in \mathbf{S}$ as:

$$\underline{M}(\mathbf{s}, \mathbf{s}') = \begin{cases} M_{X\pi}(\mathbf{s}, \mathbf{s}'), & \text{if } i \in \underline{X} \\ [\mathbf{s}' = \mathbf{s}], & \text{if } i \in \overline{X} \end{cases} . \tag{3.6}$$

Intuitively, $\underline{M}$ is defined to: (1) transition normally while inside $\underline{X}$, and (2) absorbing via a self-loop when arriving (or starting) in a state in $\overline{X}$.

Next, let the summarization function $\underline{\lambda}\colon \mathbf{S}\times\mathbf{S}\to[0,1]$ be defined by:

$$\underline{\lambda}(\mathbf{s},\mathbf{s}') = \lim_{z\to\infty} \underline{M}^{(z)}(\mathbf{s},\mathbf{s}'), \tag{3.7}$$

with $\underline{M}^{(z)}$ denoting the z-th power of stochastic matrix $\underline{M}$. Intuitively, $\underline{\lambda}^{w}$ is defined to capture the probability of starting in a state in $\underline{X}$ and returning control to a state in $\overline{X}$, namely after traversing the non-reward generating states in $\underline{X}$.

Lastly, let $|\mathbf{S}|$-by-$|\mathbf{S}|$ stochastic matrix $\overline{M}$ be defined as:

$$\overline{M}(\mathbf{s},\mathbf{s}') = \begin{cases} M_{X\pi}(\mathbf{s},\mathbf{s}') + \sum_{\hat{\mathbf{s}}\in\mathbf{S}} [\hat{i}\in\underline{X}] M_{X\pi}(\mathbf{s},\hat{\mathbf{s}})\underline{\lambda}(\hat{\mathbf{s}},\mathbf{s}'), & \text{if } i\in\overline{X}\wedge i'\in\overline{X} \\ 0, & \text{if } i\in\overline{X}\wedge i'\in\underline{X} \\ [\mathbf{s}'=\mathbf{s}], & \text{if } i\in\underline{X} \end{cases}. \tag{3.8}$$

with $\hat{\mathbf{s}} = \langle \hat{i}, \hat{s}_{x_1}, \ldots, \hat{s}_{x_{|X|}} \rangle \in X$. Intuitively, $\overline{M}$ is defined to: (1) transition normally while inside $\overline{X}$, while summarizing the transitions through $\underline{X}$ by which other outside successors $\hat{\mathbf{s}}$ are possible (from $\mathbf{s}$ to $\hat{\mathbf{s}}$) and only considering which resulting state $\mathbf{s}'$ (from $\hat{\mathbf{s}}$ to $\mathbf{s}'$) is possible to end up in after travelling through $\underline{X}$ back to $\overline{X}$; (2) transitions to other outside vertices $\overline{X}$ are summarized, instead integrated into (1), and thus not considered; and (3) ignore the effects of starting in other outside vertices' states by self-looping.

As stated, we assume that the policy network is well-defined, and that control is returned to v with probability 1 as time tends to infinity. Formally, for each *relative* Markov reward process, control is always returned to $\overline{X}$ with probability 1 as time tends to infinity, computed by summing over the combination of states for $\underline{\lambda}$, with successor controller being in $\overline{X}$:

$$\sum_{\hat{\mathbf{s}}\in\mathbf{S}} [\hat{i}\in\overline{X}]\underline{\lambda}(\mathbf{s},\hat{\mathbf{s}}) = 1,$$

for any initial state $\mathbf{s}\in\mathbf{S}$. Technically, this assumption need only be true for the directed acyclic graph described in the next section, as some stationary policies may avoid transferring control to some vertices entirely. While it is possible to analyze induced Markov chains that do not have this property, we will leave this analysis to future work, as it complicates defining the constrained semi-MDPs in the next section (e.g., $\mathcal{R}$ can be unbounded).

Additionally, there are other equivalent representations to compute these Markov chains and the state transition probabilities. The approach presented here merely attempts to break down each relevant consideration in a step-by-step manner. The decomposition approach here of the full Markov multi-reward process and an individual vertex v's relative Markov reward process will, again, prove useful in describing the constrained semi-MDP in the next section (e.g., F, T, and ρ).

3.3 Optimality Criterion

Now that the policy network's stochastic process is defined, we want to define a notion of optimality. We leverage the well-established formalism of CMDPs and SMDPs to the define values and objective for policy networks.

Specifically, we state that a policy network induces a hierarchy of constrained semi-Markov decision processes (CSMDP). Each CSMDP is dependent on its ancestors following a dependency graph. Thus, we will see that policy networks can be solved by a simple algorithm: start at the furthest vertices from the initial controller, iteratively solve each vertex's CSMDP down the dependency graph, and terminate once the initial controller vertex is reached.

To get to this point, however, we must first define the dependency graph, the CSMDP components (time, state, action, transition, reward, and constraints), and the infinite horizon objective function.

3.3.1 Dependency Graph

In order to crisply define an objective function, for any given vertex, we must define which *other vertices* affect it via performing action, transfer of control, and constraints. To this end, we introduce a dependency graph and ancestors of vertices.

From an initial controller, we derive a graph describing the direct constraint or transfer dependencies among two vertices. Formally, the **dependency graph** $\langle V_d, E_d \rangle$ is a directed acyclic graph (DAG), with $V_d \subseteq V$ and $E_d \subseteq E$ that admits the most paths from each reachable vertex leading to the initial controller v^0. Following the dependency graph, we let vertex v's **ancestors** be $\mathcal{A}_v \subset V_d$ and its **parents** be $\mathcal{P}_v \subseteq \mathcal{A}_v$.

In general, for any given policy network there can be multiple valid dependency graphs. A useful special case is when the policy network follows a tree structure, which simply produces a single unique dependency graph. Formally, if we convert edges to undirected edges (collapsing duplicates and self-loops) and produce an undirected tree, then this is the special case. The policy networks discussed in this chapter happen to be this special case, but in general it need not be the case.

Finally, the dependency graph can be either unspecified or specified a priori. For the former, any of the valid dependency graphs can be computed from the general directed graph $\langle V, E \rangle$. A simple technique to do this is presented in Algorithm 1. For the latter, we are given a dependency graph

$\langle V_d, E_d \rangle$. This not only allows us to skip the step of computing one, but also allows for a simpler alternate graphical representation when designing policy networks.

3.3.2 Relative Constrained Semi-MDP

We construct the constrained semi-MDP (CSMDP) relative to a vertex v, building off of Sections 3.2.2 and 3.2.3. Instead of considering the entire Markov multi-reward process, we only consider the vertices relevant to v via $X = \{v\} \cup \mathcal{A}_v$, resulting in $\overline{X}$ and $\underline{X}$ from Equation 3.5. From this smaller relative Markov reward process $\overline{M}$, we assume ancestor policies are stationary, with $\pi = \langle \pi_v, \pi_{x_1}, \ldots, \pi_{x_{|\mathcal{A}_v|}} \rangle$ only varying in the choice of π_v. The objective is to solve for an optimal policy $\pi_v^* \in \Pi_v$ that maximizes the expected reward for R_v. To this end, we now formalize each element of the induced CSMDP: time, state, action, transition, reward, and constraints.

As in the previous section, for notational simplicity in the definitions that follow, we use an *overline* to refer to the relative vertex v's CSMDP, given vertex v, the set of vertices X, and the policies π. For example, $\overline{\mathbf{S}}$, $\overline{T}$, $\overline{F}$, $\overline{\rho}$, and $\overline{\mathcal{R}}$ are all defined for a given v, X, and π.

Relative Times In discrete time SMDPs [98], as detailed Chapter 2, there are three notions of time which policy networks share for the relative vertex v: (1) natural process time $\overline{\tau} \in \mathbb{N}$, (2) decision epoch $\overline{t} \in \mathbb{N}$, and (3) sojourn time $\overline{\tau}^{\overline{t}} \in \mathbb{N}$. A vertex v's relative decision epochs are when it is a controller, and its relative sojourn times are the duration between being in control.

Relative State Given $X = \{v\} \cup \mathcal{A}_v$, Equation 3.5 defines the relevant vertices $\overline{X}$ which, in turn, defines the relative state space. We must consider v's own state space S_v and any relevant parent w state space S_w, should it ever gain control. This is formally defined as $\overline{\mathbf{S}} = \overline{X} \times S_{x_1} \times \cdots \times S_{x_{|\overline{X}|}}$ with an individual relative state $\overline{\mathbf{s}} = \langle \overline{i}, \overline{s}_{x_1}, \ldots, \overline{s}_{x_{|\overline{X}|}} \rangle \in \overline{\mathbf{S}}$.

Relative Action We are focused on v's control for the CSMDP. Thus the action set is simply A_v.

Relative State Transitions Considering vertices $X = \{v\} \cup \mathcal{A}_v$ yields a full specification of v's relative state transition with $\overline{M}$. From $\overline{M}$ any desired probabilistic information can be extracted. We provide the three common extracted SMDP probabilities: (1) $Pr(\overline{\tau}, \overline{\mathbf{s}}' | \overline{\mathbf{s}}, \pi_v(\overline{s}_v))$, (2) $\overline{F}(\overline{\mathbf{s}}, \pi_v(\overline{s}_v), \overline{\tau})$, and (3) $\overline{T}(\overline{\mathbf{s}}, \pi_v(\overline{s}_v), \overline{\tau}, \overline{\mathbf{s}}')$.

First, $Pr(\overline{\tau}, \overline{\mathbf{s}}' | \overline{\mathbf{s}}, \pi_v(\overline{s}_v))$ defines the probability that v is in control on or before time $\overline{\tau}$ and when control was returned at this time it ended in state $\overline{\mathbf{s}}'$, after performing action $\pi_v(\overline{s}_v)$ in state $\overline{\mathbf{s}}$. To compute this, let us define a helpful $|\mathbf{S}|$-by-$|\mathbf{S}|$ matrix $\overline{U}$ as:

$$\overline{U}(\mathbf{s}, \mathbf{s}') = \begin{cases} \overline{M}(\mathbf{s}, \mathbf{s}'), & \text{if } i \neq v \\ [\mathbf{s}' = \mathbf{s}], & \text{if } i = v \end{cases}$$

which captures: (1) the probability of transitioning when v is not in control, and (2) self-looping at whatever state it regains control. $\overline{U}$ is used to define the full state transition, as it preserves the state it was in for all future $\overline{\tau}$ after control was returned to v. Formally, for $\overline{\mathbf{s}}$ with $\overline{i} = v$, we have:

$$Pr(\overline{\tau}, \overline{\mathbf{s}}' | \overline{\mathbf{s}}, \pi_v(\overline{s}_v)) = \begin{cases} (\overline{M}\overline{U}^{(\overline{\tau}-1)})(\mathbf{s}, \mathbf{s}'), & \text{if } \overline{\tau} > 0 \wedge \overline{i}' = v \\ 0, & \text{otherwise} \end{cases} \tag{3.9}$$

with $\mathbf{s}$ assigned to $i = \overline{i}$, $s_u = \overline{s}_u$ for all $u \in \overline{X}$, and $s_w = s_w^0$ for all $w \in \underline{X}$; let $\mathbf{s}'$ be assigned similarly. Also note that $\overline{U}^{(\overline{\tau}-1)}$ is the $(\overline{\tau}-1)$-th power of $\overline{U}$, with $\overline{U}^0 = I$. Intuitively, this begins with the CSMDP Markov reward process $\overline{M}$, which may or may not preserve v's control, continuing to follow $\overline{M}$ until control is returned to v, after which the resulting state is essentially stored via $\overline{U}$'s self-loop.

Second, $\overline{F}(\overline{\mathbf{s}}, \pi_v(\overline{s}_v), \overline{\tau})$ defines the probability that v is in control on or before time $\overline{\tau}$ after performing action $\pi_v(\overline{s}_v)$ in state $\overline{\mathbf{s}}$. It can be computed using $Pr(\overline{\tau}, \overline{\mathbf{s}}' | \overline{\mathbf{s}}, \pi_v(\overline{s}_v))$:

$$\overline{F}(\overline{\mathbf{s}}, \pi_v(\overline{s}_v), \overline{\tau}) = \sum_{\overline{\mathbf{s}}' \in \overline{\mathbf{S}}} Pr(\overline{\tau}, \overline{\mathbf{s}}' | \overline{\mathbf{s}}, \pi_v(\overline{s}_v)). \tag{3.10}$$

Third, $\overline{T}(\overline{\mathbf{s}}, \pi_v(\overline{s}_v), \overline{\tau}, \overline{\mathbf{s}}')$ defines the probability that the successor state is $\overline{\mathbf{s}}'$ given control had not yet been returned to v prior to $\overline{\tau}$, after v had performed action $\pi_v(\overline{s}_v)$ in state $\overline{\mathbf{s}}$. To compute this, let us define a helpful $|\mathbf{S}|$-by-$|\mathbf{S}|$ matrix $\overline{W}$ as:

$$\overline{W}(\mathbf{s}, \mathbf{s}') = \begin{cases} \overline{M}(\mathbf{s}, \mathbf{s}')/\eta(\mathbf{s}), & \text{if } \eta(\mathbf{s}) > 0 \wedge i' \neq v \\ 0, & \text{if } \eta(\mathbf{s}) > 0 \wedge i' = v \\ [\mathbf{s}' = \mathbf{s}], & \text{otherwise} \end{cases}$$

with normalization term $\eta(\mathbf{s}) = \sum_{\mathbf{s}''} [i'' \neq v] \overline{M}(\mathbf{s}, \mathbf{s}'')$. This captures: (1) the probability of transitioning only within $\overline{X} - \{v\}$, since we are given that v is not in control up to $\overline{\tau}$ steps, (2) transitions to

v in control must be zero, and (3) any other case is considered a self-loop. $\overline{W}$ is used to define the state transition behavior when v is not in control. Formally, for $\overline{s}$ with $\mathbf{s}$ with $\overline{i}=v$, we have:

$$\overline{T}(\overline{\mathbf{s}},\pi_v(\overline{s}_v),\overline{\tau},\overline{\mathbf{s}}')=\overline{W}^{(\overline{\tau})}(\mathbf{s},\mathbf{s}') \tag{3.11}$$

with $\mathbf{s}$ assigned to $i=\overline{i}$, $s_u=\overline{s}_u$ for all $u\in\overline{X}$, and $s_w=s_w^0$ for all $w\in\underline{X}$; let $\mathbf{s}'$ be assigned similarly. Again note that $\overline{W}^{(\overline{\tau})}$ is the $\overline{\tau}$-th power of $\overline{U}$, with $\overline{W}^0=I$.

Relative Rewards The discrete time CSMDP for v has two rewards: (1) immediate reward R_v, and (2) expected reward gained at rate $\overline{\rho}:\overline{\mathbf{S}}\times A_v\times\mathbb{N}\to\mathbb{R}$. Specifically, as used in SMDP Equations 2.28 and 2.35, $\overline{\rho}(\overline{\mathbf{s}},\pi_v(\overline{s}_v),\overline{\tau})$ is defined as the reward rate at sojourn time $\overline{\tau}$ after action $\pi_v(\overline{s}_v)$ was performed in state $\overline{\mathbf{s}}$, but before the next action is performed. However, as shown in SMDP Equations 2.29 and 2.36, we can use an alternate $\overline{\rho}':\overline{\mathbf{S}}\times A_v\times\overline{\mathbf{S}}\to\mathbb{R}$ to define $\overline{\rho}$. Let $\overline{\rho}'(\overline{\mathbf{s}},\pi_v(\overline{s}_v),\overline{\mathbf{s}}'))=R_v(\overline{s}_v',\pi_{\overline{i}'}(\overline{s}_{\overline{i}'}))$. By discrete time SMDP reward rate Equation 2.36:

$$\overline{\rho}(\overline{\mathbf{s}},\pi_v(\overline{s}_v),\overline{\tau})=\gamma_v^{\overline{\tau}}\sum_{\overline{\mathbf{s}}'\in\overline{\mathbf{S}}}R_v(\overline{s}_v',\pi_{\overline{i}'}(\overline{s}_{\overline{i}'}'))\overline{T}(\overline{\mathbf{s}},\pi_v(\overline{s}_v),\overline{\tau},\overline{\mathbf{s}}'), \tag{3.12}$$

with the discount factor $\gamma_v\in[0,1)$. By discrete time SMDP expected reward Equation 2.35:

$$\overline{\mathcal{R}}(\overline{\mathbf{s}},\pi_v(\overline{s}_v))=R_v(\overline{s}_v,\pi_v(\overline{s}_v))+\sum_{\overline{\tau}'=1}^{\infty}\overline{F}(\overline{\mathbf{s}},\pi_v(\overline{s}_v),d\overline{\tau}')\sum_{\overline{\tau}=1}^{\overline{\tau}'-1}\overline{\rho}(\overline{\mathbf{s}},\pi_v(\overline{s}_v),\overline{\tau}). \tag{3.13}$$

Relative Constraints The constraints for vertex v's CSMDP follow directly from the parents, restricting the space of available policies. Formally, this is defined by the set $\mathcal{C}_v\subseteq\{v\}\cup\mathcal{P}_v$:

$$\mathcal{C}_v=\Big\{w\in\{v\}\cup\mathcal{P}_v\Big|\exists\langle w,v\rangle\in E\wedge\exists\Pi_{wv}\Big\}. \tag{3.14}$$

This set of v's direct parent vertices, possibly including a self-constraint, defines the constraints for the relative CSMDP. Formally, the vertex v's chosen policy $\pi_v\in\Pi_v$ to use must satisfy:

$$\pi_v\in\Pi_v\cap\Big(\bigcap_{w\in\mathcal{C}_v}\Pi_{wv}\Big). \tag{3.15}$$

In some cases, the intersection of the policy constraint edge sets can produce an empty set of policies. This is equivalent to an overly-constrained CMDP [2]. As in the CMDP case, policy networks will return infeasible for such a scenario, as no such solution exists. Formally, we call a policy network's CSMDP **infeasible** if:

$$\Pi_v \cap \Big(\bigcap_{w \in \mathcal{C}_v} \Pi_{wv} \Big) = \emptyset. \tag{3.16}$$

Additionally, an important consequence of the dependency graph's topological ordering is that some constraints may not be considered, such as if two vertices constrain each other directly. In these cases, the dependency graph will breaks this tie and solves them in a topologically valid manner. As such, we also consider the scenario in which a constraint is not satisfiable to be infeasible.

We consider two main forms of policy constraint edges Π_{wv} in this thesis. Both forms are in terms of a bound on regret from expected value—as defined in the next section—up to an allotted slack—allowable regret, the deviation from this optimal expected value.

First, if the state and action spaces are shared, then v can choose among all policies that were within slack of optimal for w's objective. Formally, for edge $\langle w, v \rangle \in E$, given $S_v = S_w$, $A_v = A_w$, and slack $\delta_{wv} \geq 0$, the policy constraint Π_{wv} is:

$$\Pi_{wv} = \Big\{ \pi \in \Pi_w \cap \Big(\bigcap_{u \in \mathcal{C}_w} \Pi_{uw} \Big) \Big| \overline{V}^*_w(\bar{\mathbf{s}}^0_w) - \overline{V}^\pi_w(\bar{\mathbf{s}}^0_w) \leq \delta_{wv} \Big\}, \tag{3.17}$$

with vertex w's CSMDP value function and initial state denoted by $\overline{V}_w$ and $\bar{\mathbf{s}}^0_w$, respectively. This type of constraint is used in TMDPs. Chapter 5 covers this and other related forms in detail.

Second, if only the action spaces are shared, then v can choose among all policies that have approved actions—actions that were within a local one-step slack of optimal for w's objective at a particular state. Formally, for edge $\langle w, v \rangle \in E$, given $A_v = A_w$, local one-step slack $\eta_{wv} \geq 0$, and state $\bar{\mathbf{s}}_w \in \overline{\mathbf{S}}_w$, the policy constraint Π_{wv} is:

$$\Pi_{wv} = \Big\{ \pi \in \Pi_v \Big| \overline{V}^*_w(\bar{\mathbf{s}}_w) - \overline{Q}^*_w(\bar{\mathbf{s}}_w, \pi(s_v)) \leq \eta_{wv}, \forall s_v \in S_v \Big\}. \tag{3.18}$$

with vertex w's CSMDP value and Q-value functions denoted by $\overline{V}_w$ and $\overline{Q}_w$, respectively. This type of constraint is used in MODIA. Chapter 7 covers the online use of this in detail.

3.3.3 Objective Function

The **infinite horizon** objective of a policy network is defined recursively over each successive relative CSMDP. Formally, vertex v's objective to find a policy $\pi_v \in \Pi_v$ that maximizes the expected reward starting from initial state $\overline{\mathbf{s}}^0$ subject to its ancestral constraints:

$$\begin{aligned} &\text{maximize} && \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma_v^{\overline{\tau}^t} \Big(R_v(\overline{s}_v^t, \pi_v(\overline{s}_v^t)) + \sum_{\overline{\tau}=\overline{\tau}^t}^{\overline{\tau}^{t+1}} \gamma^{-\overline{\tau}^t} \overline{\rho}(\overline{s}_v^t, \pi(\overline{s}_v^t), \overline{\tau} - \overline{\tau}^t) \Big) \Big| \pi_v, \overline{\mathbf{s}}^0 \Big] \\ &\text{subject to} && \pi_v \in \Pi_v \cap \Big(\bigcap_{w \in \mathcal{C}_v} \Pi_{wv} \Big) \end{aligned} \tag{3.19}$$

with $\overline{s}_v^t$ denoting the random variable for the state of v at its decision epoch t, $\overline{\tau}^t$ denoting the random variable for decision epoch start times, the combined term $\overline{\tau} - \overline{\tau}^t$ denoting the random variable for each decision epoch t's sojourn time, and the stationary ancestor policies π_w and policy sets Π_w used in $\overline{\rho}$ and Π_{wv}.

For a policy $\pi_v \in \Pi_v$, the **value** $\overline{V}^{\pi} : \overline{\mathbf{S}} \rightarrow \mathbb{R}$ is the expected reward at state $\overline{\mathbf{s}}$ follows the SMDP Bellman equation (as defined Equations 2.26 and 2.34):

$$\overline{V}^{\pi}(\overline{\mathbf{s}}) = \overline{\mathcal{R}}(\overline{\mathbf{s}}, \pi_v(\overline{s}_v)) + \sum_{\overline{\mathbf{s}}' \in \overline{\mathbf{S}}} \sum_{\overline{\tau}=1}^{\infty} \gamma^{\overline{\tau}} Pr(d\overline{\tau}, \overline{\mathbf{s}}' | \overline{\mathbf{s}}, \pi_v(\overline{s}_v)) \overline{V}^{\pi}(\overline{\mathbf{s}}'). \tag{3.20}$$

A policy $\pi_v^* \in \Pi_v$ is **optimal** if it obtains the maximal value $\overline{V}^*(\overline{\mathbf{s}}^0)$. This optimal value can be computed, subject to the constraints, by the Bellman optimality equation over each state $\overline{\mathbf{s}}$:

$$\overline{V}^*(\overline{\mathbf{s}}) = \max_{a_v \in A_v} \Big(\overline{\mathcal{R}}(\overline{\mathbf{s}}, a_v) + \sum_{\overline{\mathbf{s}}' \in \overline{\mathbf{S}}} \sum_{\overline{\tau}=1}^{\infty} \gamma^{\overline{\tau}} Pr(d\overline{\tau}, \overline{\mathbf{s}}' | \overline{\mathbf{s}}, a_v) \overline{V}^{\pi}(\overline{\mathbf{s}}') \Big). \tag{3.21}$$

3.3.4 Algorithm

Algorithm 1 follows directly from the definitions (i.e., dependency graph) and equations (e.g., Equaion 3.21) above. As stated above, we consider stationary policies here, leaving algorithms for non-stationary policies to future work. To **solve** a policy network means to compute optimal stationary policies for each CSMDP following the dependency graph. Optionally, we have included computing a dependency graph; it can instead be given. As it is a DAG, it defines a topological ordering $\mathbf{x}$. ComputeReversePostOrderDFS(V, E, v^0) returns vertex ordering $\mathbf{x}$ by a simple reverse post-order traversal depth-first search (DFS). Path(V_d, E_d, w, v) returns the set of paths between w and v on $\langle V_d, E_d \rangle$. SolveRelativeCSMDP(v, $\mathcal{A}_v$, $\mathbf{\Pi}^*$) solves the relative CSMDP.

Algorithm 1 An algorithm to compute the collection of optimal CSMDP policies for the vertices of a policy network.

```
Require: $\langle V, E\rangle$: The policy network.
Require: $v^0$: The initial controller.
 1: procedure ComputeDependencyGraph($V$, $E$, $\mathbf{x}$)
 2:     $V_d \leftarrow \{v \in V | \exists x_i = v\}$
 3:     $E_d \leftarrow \{\langle v, w\rangle \in E | \exists x_i = v \wedge \exists x_j = w \wedge j > i\}$
 4:     return $\langle V_d, E_d\rangle$
 5: procedure SolvePolicyNetwork($V_d$, $E_d$, $\mathbf{x}$, $v^0$)
 6:     $\mathbf{\Pi}^* \leftarrow \{\}$
 7:     for $v \leftarrow x_1, \ldots, x_k$ do
 8:         $\mathcal{A}_v \leftarrow \{w \in V' | \exists p \in$ Path($V_d$, $E_d$, $w$, $v$) $\}$
 9:         $\pi^*_v \leftarrow$ SolveRelativeCSMDP($v$, $\mathcal{A}_v$, $\mathbf{\Pi}^*$)
10:         $\mathbf{\Pi}^* \leftarrow \mathbf{\Pi}^* \cup \{\pi^*_v\}$
11:     return $\mathbf{\Pi}^*$
12: $\mathbf{x} \leftarrow$ ComputeReversePostOrderDFS($V$, $E$, $v^0$)
13: $\langle V_d, E_d\rangle \leftarrow$ ComputeDependencyGraph($\mathbf{x}$)
14: return SolvePolicyNetwork($V_d$, $E_d$, $\mathbf{x}$, $v^0$)
```

3.3.5 Conditioning

To this point, the relative Markov reward process to v, and its corresponding relative CSMDP, assume all ancestor vertices are stationary—that is, both the policy and policy set are given. We can generalize this concept by allowing any set of vertices to be given for computing vertex v's relative CSMDP. This added flexibility will be useful in the next section. Thus we formalize a notion of conditioning in policy networks.

A vertex $v \in V$ is **conditioned** on by a vertex $w \in V$, denoted $v|w$, if w is a stationary vertex. We can condition on any number of vertices, such as $v|a, b, c$. To this point, we have considered a vertex v to be conditioned on its ancestors $v|\mathcal{A}_v$. Conditioning is necessary in most cases to compute the relative CSMDP value $\overline{V}_v^{\pi_v}(\bar{\mathbf{s}}^0)$.

3.3.6 Independence

The relative CSMDP for vertex v depends on the ancestors $\mathcal{A}_v$, namely their stationary policies for transfer of control transition edges and policy sets for constraint edges. Hence, if a policy network does not initially have any stationary vertices, then solving it (e.g., with Algorithm 1) requires computing all ancestors' relative CSMDPs to obtain their optimal policies. Consider the case in which a vertex v's parents $w \in \mathcal{P}_v$ were stationary vertices—we have their policy π_w and policy set $\Pi_w \cap (\bigcap_u \Pi_{uw})$. By Equation 3.19, we can compute its value $\overline{V}_v^*(\bar{\mathbf{s}}^0)$ if we also know: (1) the summarization $\overline{\lambda}_v(\bar{\mathbf{s}}, \bar{\mathbf{s}}')$ for states $\mathbf{s} = \langle w, \ldots\rangle$ as defined by the policy transition edges which give control from w to other ancestors, and (2) the policy set available to v as defined by the policy constraint edges $\Pi_v \cap (\bigcap_w \Pi_{wv})$. In this case, we have all necessary information to solve for π_v^* without needing any other information about other ancestors. This means we can avoid computing v's relative CSMDP's optimal value and optimal policy for potentially many distant ancestors of v. This is the motivation for defining a more general notion of independence in policy networks.

To accomplish this goal, we must make an assumption regarding stationary vertices to discuss independence. For example, observe that the constraint edges $\langle w, v\rangle$ require knowledge of the value $\overline{V}_w^*(\bar{\mathbf{s}}^0)$ in order to compute the policy constraint Π_{wv}. Also, observe that the effects of any transfer of control to w must be known in $\underline{\lambda}_v$ when starting in states with w in control. Thus, given a stationary policy for w, we need to assume we can compute its value, and consequently all relevant components specific to w. Formally, the **stationary information assumption** assumes that: if $w \in V$ is stationary with policy π_w and policy set Π_w, then we know its value $\overline{V}_w^{\pi_w}(\bar{\mathbf{s}}^0)$, any policy constraints Π_{wu}, and any $\underline{\lambda}_v(\mathbf{s}, \mathbf{s}')$ for $\mathbf{s} = \langle w, \ldots\rangle$.

This assumption has been trivially true up to this point because we have $v|\mathcal{A}_v$ and all information had already been computed for ancestors. Cases in which a subset of ancestors are given also follow naturally. First, Chapter 5 discusses the case of policy constraint edges Π_{wv} and independence therein. Second, the case in which $\underline{\lambda}_v$ must be known starting at stationary controller vertex w (i.e., $\mathbf{s} = \langle w, \ldots, \rangle$) is straight-forward in policy networks with *tree* dependency graphs—nearly every policy network in this thesis. It is simply $\underline{\lambda}_v(\mathbf{s}, \mathbf{s}') = \underline{\lambda}_w(\mathbf{s}_w, \mathbf{s}'_w)$ with all state factors in $\mathbf{s}_w$ assigned to match the corresponding factors in $\mathbf{s}$. In any case, with the ability to compute components of stationary vertices, we may now define a general notion of independence.

A vertex $v \in V$ is **independent** of a vertex $w \in V$, denoted as $v \perp w$, if v's optimal value $\overline{V}^*(\bar{\mathbf{s}}^0)$ can be computed without w. Specifically, $v \perp w$ if: (1) $w \notin \{v\} \cup A_v$, (2) for all paths $p = \langle w, \ldots, u, \ldots, v\rangle = \textsc{Path}(V_d, E_d, w, v)$ there exists a stationary vertex u. Importantly, independence is *not commutative*, $v \perp w \neq w \perp v$. Also, this definition implicitly defines the policy network equivalent of *d-separation*; however, any such *causality* of sorts is much simpler in policy networks as opposed to Bayesian networks. Once independence $v \perp w$ is established, we can compute the value of v without the need for any such vertex w. Potential algorithms that analyze independence and exploit it to quickly solve policy networks is a topic of interest for future work.

3.4 Graphical and Notational Representations

Policy networks continue the tradition set by probabilistic graphical models (PGMs) with clean and powerful graphical and notational representations. This allows for rich complex decision-making with many objectives and levels of abstraction to be easily described and analyzed.

We first define a general graphical notation for any form of policy network; it does not assume a dependency graph is given. Next, we define a simpler, more specific, graphical notation for certain forms of policy network; it assumes the dependency graph is given. Finally, we define a simple graphical notation for policy network values, in a similar style to the convenient notation used in probability theory.

3.4.1 Graphical Representation

Figure 3.1 covers basic policy network notation. Each vertex as $v \in V$ is a circle and each directed edge $e \in E$ as an arrow. Edges are directed and denote their policy dependencies by any relevant variables. For example, policy constraints are denoted by their policy set Π_e and policy transitions are denoted by their function T_e. When it is not ambiguous, it may suffice to denote parameters instead, such as slack δ_e, an options' initiation set I_e, or termination function β_e. Here vertices and edges are lowercase, and their sets are uppercase; however, this need not be the case in general. The initial controllers v^0 are denoted by double-lined circles. Stationary vertices are filled-in—e.g., solved by *offline* algorithms. In contrast, non-stationary vertices which are *not* filled-in—e.g., solved by *online* algorithms. Plate notation may be used to easily group sets of similarly defined vertices.

Any vertex v which follows a standard MDP, POMDP, etc. model uses the notation $v \sim \mathit{MDP}(\cdot)$, $v \sim \mathit{POMDP}(\cdot)$, etc. in the tradition of PGMs. Intuitively, the "$\sim$" symbol refers to "selecting" or "choosing" a policy from a policy space. This notation is used for convenience. It completely describes the vertex's policy set Π_v, reward R_v, and an implicit self-loop edge with transition T_{vv}.

Formally, this extra notation means $v \sim \mathit{MDP}(S_v, A_v, T_v, R_v)$ defines v with the set of policies given by $\Pi_v = \{\pi : S_v \rightarrow A_v\}$ with reward $R_v : S_v \times A_v \rightarrow \mathbb{R}$. Also, it defines the *implicit edge*—that is, merely not graphically drawn—self-looping edge $e = \langle v, v \rangle \in E$ with policy transition $T_e = T_v$. The same formal definition holds for $v \sim \mathit{SSP}(S_v, A_v, T_v, C_v, s_v^0, s_v^g)$, except with costs such that $R_v = -C_v$.

POMDPs are similarly defined as $v \sim \mathit{POMDP}(S, A, \Omega, T, O, R)$ because they are simply a special form of continuous state MDP called a *belief MDP* [63]. Formally, the set of policies is given by $\Pi_v = \{\pi : \triangle^{|S_v|} \rightarrow A_v\}$ with reward $R_v : \triangle^{|S_v|} \times A_v \rightarrow \mathbb{R}$ such that $R_v(b_v, a_v) = \sum_{s_v} b_v(s_v) R(s_v, a_v)$. The implicit edge $e = \langle v, v \rangle \in E$ has policy transition $T_e = \tau_v$ with $\tau_v : \triangle^{|S_v|} \times A_v \times \triangle^{|S_v|} \rightarrow [0, 1]$ following the belief MDP state transition as in Equations 2.21 and 2.22.

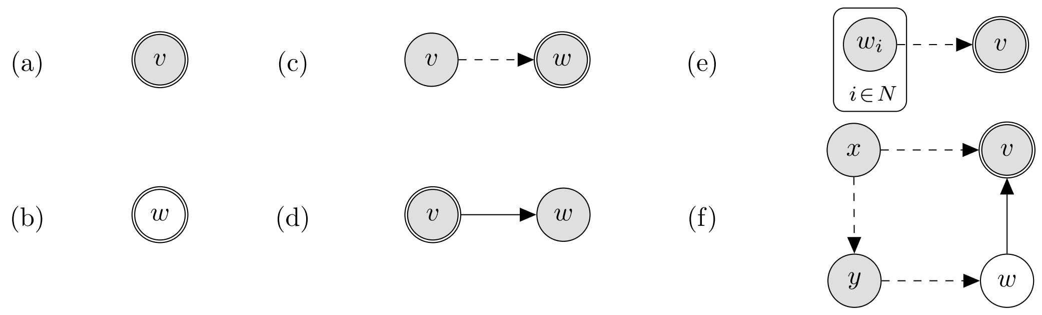


Figure 3.2: Similar basic examples as shown in Figure 3.1 of the *alternate* graphical notation described in Section 3.4.2 that is used to more quickly represent policy networks. Each $v \in V$ follows some model $v \sim MDP(S_v, A_v, T_v, R_v)$: (a) stationary vertex; (b) non-stationary vertex; (c) constraint edge; (d) transfer of control edge; (e) plate notation for a set of N constraints; and (f) mixture of the concepts. Conveniently this alternate form explicitly defines the dependency graph.

3.4.2 Alternate Graphical Representation

The previous section above describes a lossless graphical representation for any policy network. It makes no assumption about the dependency graph selected and allows for rich complex interactions among vertices. While this notation is general, it can be a bit overwhelming to view, and lacks a means to enforce a particular dependency graph to be used in the optimality criterion for the policy network. We now provide an alternate graphical representation that is not as general but is simpler and allows the dependency graph to be specified in the graphical form itself.

Formally, the graphical notation is nearly identical to Section 3.4.1 above, except that we render the dependency graph $\langle V_d, E_d \rangle$ instead of the full policy network graph $\langle V, E \rangle$. Each vertex $v \in V_d$ is a circle as before, referring to the policy space Π_v and reward R_v. Stationary vertices are filled-in; non-stationary vertices are not filled-in. However for each edge $e \in E_d$, we now *assume bidirectionality* in E. Formally, if $\langle w, v \rangle \in E_d$ then $\{\langle w, v \rangle, \langle v, w \rangle\} \subseteq E$. Moreover the edge $e = \langle w, v \rangle$ can be either: (1) a transition edge, governed by bidirectional partial functions T_{wv} and T_{vw}; or (2) a constraint edge Π_{wv}, unidirectional in the constraint, and assumed to follow either Equation 3.17 or 3.18 based on if the state space is shared or not. To quickly delineate the two, we use a solid line for a transition edge and a dotted line for constraint edge. Also this restricted graphical notation of an edge allows us to omit explicitly stating T_e or Π_e on each edge, leveraging a solid or dotted line instead. The remaining notation surrounding the model—$v \sim MDP(\cdot)$, $v \sim SSP(\cdot)$, $v \sim POMDP(\cdot)$, etc.—is exactly as detailed above.

Figure 3.2 shows example notation using this alternate graphical representation. All vertices are equivalent to those corresponding in Figure 3.1 except the mixture of concepts (f). In Figure 3.2 (f), the bidirectionality would require that edges $\{\langle y, x \rangle, \langle w, y \rangle, \langle v, x \rangle\}$ be added to E in Figure 3.1 (f).

3.4.3 Notational Representation

Following in the tradition of probability theory, we provide a notational representation of value for policy networks. Essentially, policy networks seek to compute value for collections of vertices. We can capture this simply by a vector of values. We can also embed the properties of conditioning and independence that were established previously in the formulation of value.

Let $V = \{v\}$ denote a vertex whose policy is chosen from some $v \sim \langle \Pi_v, R_r \rangle$ or $v \sim MDP(S, A, T, R)$. We simply write the **value** for a policy $\pi \in \Pi_v$ as:

$$Va(v = \pi) = \begin{bmatrix} \overline{V}_v^{\pi}(\bar{\mathbf{s}}_v^0) \end{bmatrix}.$$

with $\overline{V}_v$ denoting the value following Equation 3.20 and its initial state $\bar{\mathbf{s}}_v = \langle v, s_{x_1}^0, \ldots, x_{x_{|X_v|}}^0 \rangle$. When the policy π is omitted, we assume the optimal policy is chosen to compute the **optimal value**:

$$Va(v) = \begin{bmatrix} \overline{V}_v^{*}(\bar{\mathbf{s}}_v^0) \end{bmatrix}.$$

We can apply the useful definition of conditioning as defined in Section 3.3.5 to this notation. Let $V = \{v, w\}$ and $E_d = \{\langle w, v \rangle\}$. For a given stationary vertex w, the **conditional value** is:

$$Va(v|w) = \begin{bmatrix} \overline{V}_v^{*}(\bar{\mathbf{s}}_v^0 | w) \end{bmatrix},$$

with $\overline{V}_v^{*}(\bar{\mathbf{s}}_v^0|w)$ denoting computing $\overline{V}_i^{*}$ in Equation 3.20 using π_w and any other components offered by the stationary vertex under the stationary information assumption. Conditional value operates for any set or listed vertices, as it is defined in Section 3.3.5. Importantly, if we condition on a vertex $w \in A_v$, then we cannot condition that vertex on anything else; it is already assigned a stationary policy and policy set.

Most vertices are, in fact, dependent on their ancestors to compute their value. In Section 3.3, we actually compute $Va(v|\mathcal{A}_v)$, since hidden in the definition of $\overline{V}_v^{*}$ (Equation 3.19) contains a dependence on all ancestors $\mathcal{A}_v$.

Fortunately, we can also apply the definition of **independence** as defined Section 3.3.6:

$$Va(v|w) = Va(v) \qquad \text{if } v \perp w.$$

Observe that $Va(v|V - \mathcal{A}_v) = Va(v)$ by definition of the dependency graph and relative CSMDP.

We can consider multiple vertices' values simultaneously. Let $V = \{v, w\}$. The **joint value** is:

$$Va(v,w) = \begin{bmatrix} Va(v|\mathcal{A}_v) \\ Va(w|\mathcal{A}_w) \end{bmatrix}.$$

This notation leverages block matrix notation. Hence, we can chain this definition any number of times. For example, if $V = \{u, v, w\}$, then:

$$Va(u,v,w) = \begin{bmatrix} Va(u|\mathcal{A}_u) \\ Va(v,w|\mathcal{A}_v \cup \mathcal{A}_w) \end{bmatrix} = \begin{bmatrix} Va(u|\mathcal{A}_u) \\ Va(v|\mathcal{A}_v \cup \mathcal{A}_w) \\ Va(w|\mathcal{A}_v \cup \mathcal{A}_w) \end{bmatrix} = \begin{bmatrix} Va(u|\mathcal{A}_u) \\ Va(v|\mathcal{A}_v) \\ Va(w|\mathcal{A}_w) \end{bmatrix}.$$

Lastly, we can leverage the matrix notation by using standard basis vectors $e_1 = [1, 0]^T$ and $e_2 = [0, 1]^T$. Let $V = \{v, w\}$. We may also write the joint value in the form:

$$Va(v,w) = e_1^T \, Va(v|\mathcal{A}_v) + e_2^T \, Va(w|\mathcal{A}_w).$$

In any case, this notational form of value, conditioning, and independence is simply an efficient way to represent value. The rich mathematical manipulations in probability theory are not necessarily needed here, given the more focused graphical structure to define value. We leave any detailed analysis of this form to future work.

3.5 Theoretical Analysis

We now show the generality of policy networks by proving that they can encapsulate various models such as the options framework and CMDPs. Additionally, this section also serves as a demonstration of the design of policy networks to provide guidance for how to create them. Proposition 1 begins with a statement regarding policy networks' generality beyond (PO)MDPs.

Proposition 1. *Policy networks generalize MDPs.*

Proof. For any MDP $\langle S_v, A_v, T_v, R_v\rangle$, we must construct an equivalent policy network. Let $V = \{v\}$ with $v \sim \langle \Pi_v, R_v\rangle$ for $\Pi_v = \{\pi : S_v \to A_v\}$. Let $E = \{e = \langle v, v\rangle\}$ with transition $T_e = T$ such that $Pr(v, s'_v | v, s_v, a_v) = T_e(s_v, a_v, s'_v) = T_v(s_v, a_v, s'_v)$. Or, using equivalent notation, simply let the vertex be written as $v \sim MDP(S_v, A_v, T_v, R_v)$. Let the initial controller be $v^0 = v$.

We have $X = V = \{v\}$. By Equation 3.2, for any policy π, $M_{X\pi} = M^v_{X\pi}$ for the state space $\mathbf{S} = \{v\} \times S_v$ with $\mathbf{s} = \langle v, s_v\rangle \in \mathbf{S}$. By Equation 3.3, $M^v_{X\pi}(\overline{\mathbf{s}}, \overline{\mathbf{s}}') = \mu(V, v, s_v, \pi_v(s_v), v, s'_v)$ as $i^t = v$ for all t. By Equation 3.1 and definition of $T_e = T$ above, $\mu(V, v, s_v, \pi_v(s_v), v, s'_v) = Pr(v, s'_v | v, s_v, \pi_v(s_v)) = T(s_v, \pi_v(s_v), s'_v)$.

By Equation 3.5, $\overline{X} = \{v\}$. This implies $\underline{X} = \emptyset$, obviating the need for $\underline{M}$ and $\underline{\lambda}$. By Equation 3.8, only the first case applies, and $[\hat{i} \in \underline{X}] = 0$. Thus, we arrive at $\overline{M}(\mathbf{s}, \mathbf{s}') = M_{X\pi}(\mathbf{s}, \mathbf{s}') = T_v(s_v, \pi_v(s_v), s'_v)$.

The relative CSMDP has states $\overline{\mathbf{S}} = \mathbf{S} = \{v\} \times S_v$, with $\overline{\mathbf{s}} = \langle v, \overline{s}_v\rangle \in \overline{\mathbf{S}}$, and actions A_v. By Equation 3.9, $Pr(\overline{\tau}, \overline{\mathbf{s}}' | \overline{\mathbf{s}}, \pi_v(\overline{s}_v)) = \overline{M}(\mathbf{s}, \mathbf{s}') = T_v(\overline{s}_v, \pi_v(\overline{s}_v), \overline{s}'_v)[\overline{s}'_v = \overline{s}_v]^{\overline{\tau}-1}$ for $\overline{\tau} > 0$ and $Pr(\overline{\tau}, \overline{\mathbf{s}}' | \overline{\mathbf{s}}, \pi_v(\overline{s}_v)) = 0$ for $\overline{\tau} = 0$. Therefore, the time-differential state transition is $Pr(d\overline{\tau}, \overline{\mathbf{s}}' | \overline{\mathbf{s}}, \pi_v(\overline{s}_v)) = T_v(\overline{s}_v, \pi_v(\overline{s}_v), \overline{s}'_v)$ $[\overline{\tau} = 1]$. By Equation 3.10, we have $\overline{F}(\overline{\mathbf{s}}, \pi_v(\overline{s}_v), d\overline{\tau}) = [\overline{\tau} = 1]$, akin to Equation 2.24.

By Equation 3.13, $\overline{\mathcal{R}}(\overline{\mathbf{s}}, \pi_v(\overline{s}_v)) = R_v(\overline{s}_v, \pi_v(\overline{s}_v)) + 0 = R_v(\overline{s}_v, \pi_v(\overline{s}_v))$, noting that $\overline{F}$ obviates the need to consider $\overline{\rho}$. Also, $\mathcal{P}_v = \emptyset$ implies by Equation 3.14 that $\mathcal{C}_v = \emptyset$. In other words, there are no instances of transfer of control nor are there any constraints.

By Equation 3.20, with $\overline{\mathbf{s}} = \langle v, \overline{s}_v\rangle$, we have:

$$\overline{V}^{\pi}(\overline{\mathbf{s}}) = R_v(\overline{s}_v, \pi_v(\overline{s}_v)) + \gamma_v \sum_{\overline{s}'_v \in S_v} T_v(\overline{s}_v, \pi_v(\overline{s}_v), \overline{s}'_v)\overline{V}^{\pi}(\overline{\mathbf{s}}') \tag{3.22}$$

with $\overline{\mathbf{s}}' = \langle v, \overline{s}'_v\rangle$. Hence Equation 3.20 becomes the Bellman equation for the MDP in Equation 2.2.

Thus, we have represented any MDP's states, actions, transition, reward, and value equations with an policy network. $\square$

We consider two related cases of *policy constraint edges*: CMDPs in Proposition 2 and MODIA in Chapter 7's Proposition 26. Here, constraints limit the space of policies from parent vertices to a child controller vertex. CMDPs represent a policy network with a shared both state and action space, constrained offline with stationary policies. MODIA represents a policy network with a different state space but a shared action space—illustrating how performing action can simultaneously affect many models—constrained online with non-stationary policy sets.

Proposition 2. *Policy networks generalize CMDPs.*

Proof. For any CMDP $\langle S, A, T, R, \mathbf{C}, \mathbf{c} \rangle$ with k constraints, we must construct an equivalent policy network. Please see Figure 3.3 (a) for the graphical representation. Let $V = \{v_0, \ldots, v_k\}$ with $v_0 \sim$ *MDP*(S, A, T, R) and $v_j \sim$ *MDP*$(S, A, T, -C_j)$ for all $j \in K = \{1, \ldots, k\}$. Policies for $v_j \in V$ can be deterministic $\Pi_j = \{\pi : S \to A\}$ or stochastic $\Pi_j = \{\pi : S \times A \to [0, 1]\}$; the analysis is identical. Thus, without loss of generality, we will write the value equation for deterministic policies here; using stochastic policies merely introduces an outer summation of a stochastic action selector for value function in Equations 3.20 and 3.21. Let the initial controller be $v^0 = v_0$.

Let $E = \{\langle v_j, v_j \rangle, \forall j \in \{0\} \cup K\} \cup \{\langle v_j, v_0 \rangle, \langle v_0, v_j \rangle, \forall j \in K\}$. Each edge $\langle v_j, v_j \rangle \in E$ are the self-loops with policy transition edge $T_{v_j v_j}(s_j, a_j, s'_j) = Pr(v'_j, s'_j | v_j, s_j, a_j) = T(s_j, a_j, s'_j)[v'_j = v_j]$. Each edge $\langle v_j, v_0 \rangle \in E$ has the policy constraint defined in Equation 3.17 with a slack of $\delta_{v_j v_0} = \overline{V}^*_{v_j}(\overline{\mathbf{s}}^0_{v_j}) + c_j$, such that $\Pi_{v_j v_0} = \{\pi \in \Pi_{v_j} | \overline{V}^*_{v_j}(\overline{\mathbf{s}}^0_{v_j}) - \overline{V}^\pi_{v_j}(\overline{\mathbf{s}}^0_{v_j}) \leq \delta_{v_j v_0}\}$. Notationally we must include the sub-script v_j in $\overline{V}_{v_j}$ to delineate each value.

We have the dependency graph $\langle V_d, E_d \rangle$ with $V_d = V$ and $E_d = \{\langle v_j, v_0 \rangle, \forall j \in K\}$. Following the optimality criterion in Section 3.3, we solve each relative CSMDP following the topological ordering of the dependency graph. Thus, cases $j \in K$ can be solved independently first, followed by case $j = 0$.

Case $\mathbf{j} \in \mathbf{K}$: For each vertex v_j with $j \in K$, we have $X = \{v_j\}$. This case follows almost identically from Proposition 1. We include the details here for completeness and to highlight the minor variations. By Equation 3.2, for any policy π, $M_{X\pi} = M^{v_j}_{X\pi}$ for the state space $\mathbf{S} = \{v_j\} \times S$ with $\mathbf{s} = \langle v_j, s_{v_j} \rangle \in \mathbf{S}$. By Equation 3.3, $M^{v_j}_{X\pi}(\mathbf{s}, \mathbf{s}') = \mu(\{v_j\}, v_j, s_{v_j}, \pi_{v_j}(s_{v_j}), v_j, s'_{v_j})$ as $i^t = v_j$ for all t. By Equation 3.1, the definition of $T_{v_j v_j} = T$ above that places all probability weight on preserving v_j's control ($i' = v'_j = v_j$), and $X = \{v_j\}$, we have $\mu(\{v_j\}, v_j, s_{v_j}, \pi_{v_j}(s_{v_j}), v'_j, s'_{v'_j}) = Pr(v'_j, s'_{v'_j} | v_j, s_{v_j}, \pi_{v_j}(s_{v_j})) = T(s_{v_j}, \pi_{v_j}(s_{v_j}), s'_{v'_j})[v'_j = v_j]$.

By Equation 3.5, $\overline{X} = \{v_j\}$. This implies $\underline{X} = \emptyset$, obviating the need for $\underline{M}$ and $\underline{\lambda}$. By Equation 3.8, only the first case applies, and $[\hat{i} \in \underline{X}] = 0$. Thus, we arrive at $\overline{M}(\mathbf{s}, \mathbf{s}') = M_{X\pi}(\mathbf{s}, \mathbf{s}') = T(s_{v_j}, \pi_{v_j}(s_{v_j}), s'_{v_j})$, removing the unnecessary $[v'_j = v_j] = 1$ since, again, $i^t = v_j$ by $\mathbf{S} = \{v_j\} \times S$.

The relative CSMDP has states $\overline{\mathbf{S}} = \mathbf{S} = \{v_j\} \times S$, with $\overline{\mathbf{s}} = \langle v_j, \overline{s}_{v_j} \rangle \in \overline{\mathbf{S}}$, and actions A. By Equation 3.9, $Pr(\overline{\tau}, \overline{\mathbf{s}}' | \overline{\mathbf{s}}, \pi_{v_j}(\overline{s}_{v_j})) = \overline{M}(\mathbf{s}, \mathbf{s}') = T(\overline{s}_{v_j}, \pi_{v_j}(\overline{s}_{v_j}), \overline{s}'_{v_j})[\overline{s}'_{v_j} = \overline{s}_{v_j}]^{\overline{\tau} - 1}$ for $\overline{\tau} > 0$ with $\overline{i}' = v_j$, and also simply $Pr(\overline{\tau}, \overline{\mathbf{s}}' | \overline{\mathbf{s}}, \pi_{v_j}(\overline{s}_{v_j})) = 0$ for $\overline{\tau} = 0$. Therefore, the time-differential state transition just follows T, with $Pr(d\overline{\tau}, \overline{\mathbf{s}}' | \overline{\mathbf{s}}, \pi_{v_j}(\overline{s}_{v_j})) = T(\overline{s}_{v_j}, \pi_{v_j}(\overline{s}_{v_j}), \overline{s}'_{v_j})[\overline{\tau} = 1]$. By Equation 3.10, $\overline{F}(\overline{\mathbf{s}}, \pi_{v_j}(\overline{s}_{v_j}), d\overline{\tau}) = [\overline{\tau} = 1]$, akin to Equation 2.24.

By Equation 3.13, $\overline{\mathcal{R}}(\overline{\mathbf{s}}, \pi_{v_j}(\overline{s}_{v_j})) = -C_j(\overline{s}_{v_j}, \pi_{v_j}(\overline{s}_{v_j})) + 0 = -C_j(\overline{s}_{v_j}, \pi_{v_j}(\overline{s}_{v_j}))$, noting that $\overline{F}$ obviates the need to consider $\overline{\rho}$. Also, $\mathcal{P}_{v_j} = \emptyset$ implies by Equation 3.14 that $\mathcal{C}_{v_j} = \emptyset$.

By Equation 3.20, with $\overline{\mathbf{s}} = \langle v_j, \overline{s}_{v_j} \rangle$, we have:

$$\overline{V}^{\pi}_{v_j}(\overline{\mathbf{s}}) = -C_j(\overline{s}_{v_j}, \pi_{v_j}(\overline{s}_{v_j})) + \gamma_{v_j} \sum_{\overline{s}'_{v_j} \in S} T(\overline{s}_{v_j}, \pi_{v_j}(\overline{s}_{v_j}), \overline{s}'_{v_j}) \overline{V}^{\pi}_{v_j}(\overline{\mathbf{s}}') \tag{3.23}$$

with $\overline{\mathbf{s}}' = \langle v_j, \overline{s}'_{v_j} \rangle$. Hence Equation 3.20 becomes the Bellman equation for the CMDP's constraints in Equations 2.43 and 2.45. Thus, for each constraint $j \in K$, we can compute $\overline{V}^{*}_{v_j}(\overline{\mathbf{s}}^0_{v_j})$ and the policy constraint edges $\Pi_{v_j v_0}$.

Case j = 0: For vertex v_0, we have $X = \{v_0\} \cup \mathcal{P}_{v_0} = V$. By Equation 3.2, for policies π, $M_{X\pi} = \prod_{v_j \in V} M^{v_j}_{X\pi}$ for the state space $\mathbf{S} = V \times S \times \cdots \times S$ with $\mathbf{s} = \langle i, s_{v_0}, s_{v_1}, \ldots, s_{v_k} \rangle \in \mathbf{S}$. By Equation 3.3, at time $t = 0$, $M^{v_0}_{X\pi}(\mathbf{s}, \mathbf{s}') = \mu(V, v_0, s_{v_0}, \pi_{v_0}(s_{v_0}), v_j, s'_{v_j})$. By Equation 3.1 and definition of $T_{v_j v_j} = T$ above, $\mu(V, v_0, s_{v_0}, \pi_{v_0}(s_{v_0}), v_0, s'_{v_0}) = Pr(v_0, s'_{v_0} | v_0, s_{v_0}, \pi_{v_0}(s_{v_0})) = T(s_{v_0}, \pi_{v_0}(s_{v_0}), s'_{v_0})[i' = v_0]$; the probability of another controller $v_j \neq v_0$ is zero: $\mu(V, v_0, s_{v_0}, \pi_{v_0}(s_{v_0}), v_0, s'_{v_0}) = 0$. Also, each other v_j has $M^{v_j}_{X\pi}(\mathbf{s}, \mathbf{s}') = [s'_{v_j} = s'_{v_j}]$; informally, they copy the state of the successor controller, which must be s'_{v_0}. Thus, by Equation 3.2, $M_{X\pi}(\mathbf{s}, \mathbf{s}') = T(s_{v_0}, \pi_{v_0}(s_{v_0}), s'_{v_0})[i' = v_0]$. Therefore, at time $t = 1$, we have $i^1 = v_0$. The same logic as above applies in succession for all time t.

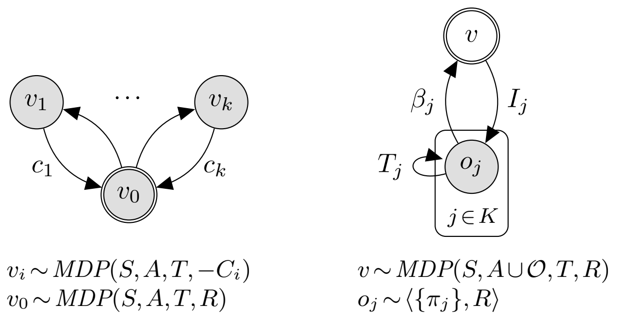
By Equation 3.5, $\overline{X} = V$. This implies $\underline{X} = \emptyset$, again obviating the need for $\underline{M}$ and $\underline{\lambda}$. By Equation 3.8, only the first case applies, and $[\hat{i} \in \underline{X}] = 0$. Thus, we arrive at $\overline{M}(\mathbf{s}, \mathbf{s}') = M_{X\pi}(\mathbf{s}, \mathbf{s}') = T(s_{v_0}, \pi_{v_0}(s_{v_0}), s'_{v_0})[i' = v_0]$.

The relative CSMDP has states $\overline{\mathbf{S}} = \mathbf{S} = V \times S \times \cdots \times S$, with $\overline{\mathbf{s}} = \langle i, \overline{s}_{v_0}, \overline{s}_{v_1}, \ldots, \overline{s}_{v_k} \rangle \in \overline{\mathbf{S}}$, and actions A. By Equation 3.9, $Pr(\overline{\tau}, \overline{\mathbf{s}}' | \overline{\mathbf{s}}, \pi_{v_0}(\overline{s}_{v_0})) = \overline{M}(\mathbf{s}, \mathbf{s}') = T(\overline{s}_{v_0}, \pi_{v_0}(\overline{s}_{v_0}), \overline{s}'_{v_0})[i' = v_0][\overline{s}'_{v_0} = \overline{s}_{v_0}]^{\overline{\tau} - 1}$ for $\overline{\tau} > 0$ and also simply $Pr(\overline{\tau}, \overline{\mathbf{s}}' | \overline{\mathbf{s}}, \pi_{v_0}(\overline{s}_{v_0})) = 0$ for $\overline{\tau} = 0$. Therefore, the time-differential state transition follows T and enforces v_0 is the controller: $Pr(d\overline{\tau}, \overline{\mathbf{s}}' | \overline{\mathbf{s}}, \pi_{v_0}(\overline{s}_{v_0})) = T(\overline{s}_{v_0}, \pi_{v_0}(\overline{s}_{v_0}), \overline{s}'_{v_0})[i' = v_0][\overline{\tau} = 1]$. By Equation 3.10, $\overline{F}(\overline{\mathbf{s}}, \pi_{v_0}(\overline{s}_{v_0}), d\overline{\tau}) = [\overline{\tau} = 1]$, akin to Equation 2.24.

By Equation 3.13, $\overline{\mathcal{R}}(\overline{\mathbf{s}}, \pi_{v_0}(\overline{s}_{v_0})) = R(\overline{s}_{v_0}, \pi_{v_0}(\overline{s}_{v_0})) + 0 = R(\overline{s}_{v_0}, \pi_{v_0}(\overline{s}_{v_0}))$, noting that $\overline{F}$ obviates the need to consider $\overline{\rho}$. By Equation 3.20, with $\overline{\mathbf{s}} = \langle i, \overline{s}_{v_0}, \overline{s}_{v_1}, \ldots, \overline{s}_{v_k} \rangle = \langle v_0, \overline{s}_{v_0}, \overline{s}_{v_0}, \ldots, \overline{s}_{v_0} \rangle$, we have:

$$\overline{V}^{\pi}_{v_0}(\overline{\mathbf{s}}) = R(\overline{s}_{v_0}, \pi_{v_0}(\overline{s}_{v_0})) + \gamma_{v_0} \sum_{\overline{s}'_{v_0} \in S} T(\overline{s}_{v_0}, \pi_{v_0}(\overline{s}_{v_0}), \overline{s}'_{v_0}) \overline{V}^{\pi}_{v_0}(\overline{\mathbf{s}}') \tag{3.24}$$

with $\overline{\mathbf{s}}' = \langle i', \overline{s}'_{v_0}, \overline{s}'_{v_1}, \ldots, \overline{s}'_{v_k} \rangle = \langle v_0, \overline{s}'_{v_0}, \overline{s}'_{v_0}, \ldots, \overline{s}'_{v_0} \rangle$. This assignment of state $\overline{\mathbf{s}}$ and successor $\overline{\mathbf{s}}$ is a consequence of: (1) the initial controller is $v^0 = v_0$, and (2) all time steps preserve v_0 as the controller with $i^t = v_0$. The probability of any other controller is zero, and all states copy the state of v_0. Thus,



v_1
$\cdots$
v_k
c_1
v_0
c_k
$v_i \sim MDP(S,A,T,-C_i)$
$v_0 \sim MDP(S,A,T,R)$
v
β_j
I_j
T_j
o_j
$j \in K$
$v \sim MDP(S,A\cup\mathcal{O},T,R)$
$o_j \sim \langle\{\pi_j\},R\rangle$

(a) CMDP (b) Options

Figure 3.3: Two examples of policy networks: (a) a constrained MDP, traditionally for a planning agent; and (b) the options framework, traditionally for a reinforcement learning agent.

such successors are omitted in the summation by replacing the sum over S and assigning the $\bar{\mathbf{s}}$ and $\bar{\mathbf{s}}'$ as stated.

Importantly, $\mathcal{P}_{v_0} = \{v_1,\ldots,v_k\}$ implies by Equation 3.14 that $\mathcal{C}_{v_0} = \mathcal{P}_{v_0}$. Therefore, we have our desired set of k constraints. By Equation 3.15, any policy π_{v_0} chosen for v_0 must be subject to the constraints: for each $v_j \in \mathcal{C}_{v_0}$, $\overline{V}^*_{v_j}(\bar{\mathbf{s}}^0_{v_j}) - \overline{V}^{\pi_{v_0}}_{v_j}(\bar{\mathbf{s}}^0_{v_j}) \leq \delta_{v_j v_0} = \overline{V}^*_{v_j}(\bar{\mathbf{s}}^0_{v_j}) + c_j$. Equivalently we can write each constraint as: $-\overline{V}^{\pi_{v_0}}_{v_j}(\bar{\mathbf{s}}^0_{v_j}) \leq c_j$. This is identical to the CMDP constraints in Equation 2.45.

Hence both cases show that Equation 3.20 becomes Equations 2.45 and 2.43: (1) the Bellman equation for the CMDP's constraints, and (2) the constrained Bellman equation for the CMDP's primary objective.

Thus, we have represented any CMDP's states, actions, transition, reward, constraints, and value equations within a policy network. $\square$

Next, we consider two related cases of *policy transition edges*: the options framework in Proposition 3 and SAS in Chapter 6's Proposition 22. Here, transfer of control happens between parent and child vertices, both online (options) and offline (SAS). Options represent a policy network with a shared state space and shared action space, learning online with a non-stationary policy. SAS represents a policy network with different state space and different action space—illustrating how how different models can interact—planning offline with a stationary policies.

Proposition 3. *Policy networks generalize options.*

Proof. For any MDP $\langle S,A,T,R\rangle$ and set of options $\mathcal{O} = \{\mathcal{O}_1 \ldots, \mathcal{O}_k\}$ with $\mathcal{O}_j = \langle \mathcal{I}_j, \pi_j, \beta_j\rangle$, we must construct an equivalent policy network. Please see Figure 3.3 (b) for the graphical representation. As v is traditionally a reinforcement learning agent, we can represent the vertex's policy as non-stationary. In both offline planning and online learning, we are concerned with computing an optimal policy using options. We consider both Markov options and semi-Markov options; both follow the same logic save for a variation in their state space.

In both Markov and semi-Markov cases, let $V = \{v, o_1, \ldots, o_k\}$, with $K = \{1, \ldots, k\}$. Let $E = \{\langle v, v\rangle\} \cup \{\langle o_j, o_j\rangle, \forall j \in K\} \cup \{\langle v, o_j\rangle, \langle o_j, v\rangle, \forall j \in K\}$. Each edge has a policy transition as well, defined by the partial functions T_{vv}, $T_{o_j o_j}$, $T_{v o_j}$, and $T_{o_j v}$. The specific assignment of these four policy transition edges is described in the two cases below.

We have the dependency graph $\langle V_d, E_d\rangle$ with $V_d = V$ and $E_d = \{\langle o_j, v\rangle, \forall j \in K\}$. Following the optimality criterion in Section 3.3, we solve each relative CSMDP following the topological ordering of the dependency graph. Thus, cases o_j can be solved independently first, followed by case v.

First, we precisely define the Markov and semi-Markov option cases. Once defined, the remainder of the proof applies to both these cases. Namely, we detail the o_j vertices and then, finally, we detail the v vertices.

Step 1: Below we define the policy networks for: (1) *Markov* and (2) *semi-Markov options*.

Case Markov Options: Let $v \sim MDP(S, A_v, T_{vv}, R_v)$ and for each option $\mathcal{O}_j$ let $o_j \sim \langle \Pi_{o_j}, R_v\rangle$ with $A_v = A \cup \mathcal{O}$ and only the option policy $\Pi_{o_j} = \{\pi_j\}$ available to o_j, following $\pi_j : S \to A_v$. Without loss of generality in MDPs, actions can be defined for each state (i.e., $A_v(s_v)$), handling any invalid execution of options in states. In policy networks, we can either define Π_v to omit any such invalid policies, or define a policy constraint that removes them: $\Pi_{vv} = \{\pi \in \Pi_v | \pi(s_v) \in A \cup \{\mathcal{O}_j | s_v \in \mathcal{I}_j(s_v)\}, \forall s_v \in S\}$. Let R_v be $R_v(s_v, a_v) = R(s_v, a_v)$ if $a_v \in A$ and $R_v(s_v, \mathcal{O}_j) = R(s_v, \pi_{o_j}(s_v))$ if $a_v = \mathcal{O}_j \in \mathcal{O}$.

For v's state transitions, let $T_{vv}(s_v, a_v, s'_v) = T(s_v, a_v, s'_v)$ if $a_v \in A$ as normal. Also, for option actions $a_v = \mathcal{O}_j \in \mathcal{O}$, let $T_{v o_j}(s_v, \mathcal{O}_j, s'_{o_j}) = (1 - \beta_j(s'_{o_j}))T(s_v, \pi_{o_j}(s_v), s'_{o_j})$ transfer control to the option o_j—when allowed by v's reduced policy space from $\mathcal{I}_j$. Interestingly, we also must allow the transfer to immediately fail after the option performs one action, as represented by $T_{vv}(s_v, \mathcal{O}_j, s'_v) = \beta_j(s'_v)T(s_v, \pi_{o_j}(s_v), s'_v)$.

Similarly for each option o_j's state transitions, let $T_{o_j o_j}(s_{o_j}, a_{o_j}, s'_{o_j}) = (1 - \beta_j(s'_{o_j}))T(s_{o_j}, a_{o_j}, s'_{o_j})$ if $a_{o_j} \in A$ preserve controller o_j following $(1 - \beta_j(s'_{o_j}))$. Conversely, we let $T_{o_j v}(s_{o_j}, a_{o_j}, s'_v) = \beta_j(s'_{o_j})$ $T(s_{o_j}, a_{o_j}, s'_v)$ transfer control back to v stochastically following $\beta_j(s'_{o_j})$. If desired, options can be trivially defined to transfer control to other options, as done with v. However, to preserve the original options framework definition, we assume here that each fixed option policy π_j does not execute another option. Formally, for all $s_{o_j} \in S$, $\pi_j(s_{o_j}) \notin \mathcal{O}$.

Case Semi-Markov Options: Semi-Markov options have a similar structure, except the shared state space is $\bar{H}$ instead of S. Thus, the state is $\bar{h} = \langle s^0, a^0, s^1, a^1, \ldots, a^{t-1}, s^t\rangle \in \bar{H}$, with a t particular for the state $\bar{h}$. Let $v \sim MDP(\bar{H}, A_v, T_{vv}, R_v)$ and for each option $\mathcal{O}_j$ let $o_j \sim \langle \Pi_{o_j}, R_v \rangle$ with $A_v = A \cup \mathcal{O}$. Only the option policy $\Pi_{o_j} = \{\pi_j\}$ available to o_j follows $\pi_j : \bar{H} \to A_v$.

For v's state transitions, let $T_{vv}(\bar{h}_v, a_v, \bar{h}'_v) = T(s^0_v, a_v, s'_v)[\bar{h}'_v = \langle s'_v \rangle]$ if $a_v \in A$ as normal, essentially ignoring the option's time component in the state space. Also, for option actions $a_v = \mathcal{O}_j \in \mathcal{O}$, let $T_{vo_j}(\bar{h}_v, \mathcal{O}_j, \bar{h}'_{o_j}) = (1 - \beta_j(s'_{o_j}))T(s^0_v, \pi_{o_j}(s^0_v), s'_{o_j})[\bar{h}'_{o_j} = \langle s'_{o_j} \rangle]$ transfer control to the option o_j—when allowed by v's reduced policy space from $\mathcal{I}_j$. Interestingly, we also must allow the transfer to immediately fail after the option performs one action, as represented by $T_{vv}(\bar{h}_v, \mathcal{O}_j, \bar{h}'_v) = \beta_j(s'_v)T(s^0_v, \pi_{o_j}(s^0_v), s'_v)[\bar{h}'_v = \langle s'_v \rangle]$.

Similarly for each option o_j's state transitions, let $T_{o_j o_j}(\bar{h}_{o_j}, a_{o_j}, \bar{h}'_{o_j}) = (1 - \beta_j(s'_{o_j}))T(s^t_{o_j}, a_{o_j}, s'_{o_j})$ $[\bar{h}'_{o_j} = \langle s^0_{o_j}, a^0_{o_j}, \ldots, a^{t-1}_{o_j}, s^t_{o_j}, a_{o_j}, s'_{o_j} \rangle]$ if $a_{o_j} \in A$ preserve controller o_j following $(1 - \beta_j(s'_{o_j}))$. Conversely, let $T_{o_j v}(\bar{h}_{o_j}, a_{o_j}, \bar{h}'_v) = \beta_j(s'_{o_j})T(s^t_{o_j}, \pi_{o_j}(s^t_{o_j}), s'_v)[\bar{h}'_v = \langle s'_v \rangle]$ transfer control back to v stochastically following $\beta_j(s'_{o_j})$. If desired, options can be trivially defined to transfer control to other options, as described above.

<u>Step 2</u>: Below we evaluate the optimality criterion for: (1) any vertex o_j and (2) vertex v. We focus the remainder of the proof on the Markov case, as the semi-Markov case below is nearly identical.

Case $\mathbf{o_j}$: For each vertex o_j with $j \in K$, we have $X = \{o_j\}$. This case follows almost identically from Propositions 1 and 2. We include the details here for completeness and to highlight the minor variations. By Equation 3.2, for any policy π, $M_{X\pi} = M^{o_j}_{X\pi}$ for the state space $\mathbf{S} = \{o_j\} \times S$ with $\mathbf{s} = \langle o_j, s_{o_j} \rangle \in \mathbf{S}$. By Equation 3.3, $M^{o_j}_{X\pi}(\mathbf{s}, \mathbf{s}') = \mu(\{o_j\}, o_j, s_{o_j}, \pi_{o_j}(s_{o_j}), o_j, s'_{o_j})$ as $i^t = o_j$ for all t given $X = \{o_j\}$. By Equation 3.1, the definition of $T_{o_j o_j}$ above that places all probability weight on preserving o_j's control ($i' = o'_j = o_j$) if the action is in A. It also shifts all probability weight on preserving o_j's control by the definition of μ and given $X = \{o_j\}$. Specifically, there are two piecewise cases for μ, either $s'_{o_j} \neq s_{o_j}$ or $s'_{o_j} = s_{o_j}$. In the first case, $\mu(\{o_j\}, o_j, s_{o_j}, \pi_{o_j}(s_{o_j}), o'_j, s'_{o'_j}) = Pr(o'_j, s'_{o'_j} | o_j, s_{o_j}, \pi_{o_j}(s_{o_j})) = (1 - \beta_j(s'_{o'_j}))T(s_{o_j}, \pi_{o_j}(s_{o_j}), s'_{o'_j})[o'_j = o_j \wedge s'_{o_j} \neq s_{o_j}]$. In the second case, $\mu(\{o_j\}, o_j, s_{o_j}, \pi_{o_j}(s_{o_j}), o'_j, s'_{o'_j}) = Pr(o'_j, s'_{o'_j} | o_j, s_{o_j}, \pi_{o_j}(s_{o_j})) = ((1 - \beta_j(s'_{o'_j}))T(s_{o_j}, \pi_{o_j}(s_{o_j}), s'_{o'_j}) + \sum_{s'_v \in S} \beta_j(s'_{o'_j})T(s_{o_j}, \pi_{o_j}(s_{o_j}), s'_v))[o'_j = o_j \wedge s'_{o_j} = s_{o_j}]$. Let this piecewise state transition be denoted simply by $T'(s_{o_j}, \pi_{o_j}(s_{o_j}), s'_{o_j})$ below, obviating redundant and unnecessary terms such as $[o'_j = o_j]$.

By Equation 3.5, $\overline{X} = \{o_j\}$. This implies $\underline{X} = \emptyset$, obviating the need for $\underline{M}$ and $\underline{\lambda}$. By Equation 3.8, only the first case applies, and $[\hat{i} \in \underline{X}] = 0$. Thus, we arrive at $\overline{M}(\mathbf{s}, \mathbf{s}') = M_{X\pi}(\mathbf{s}, \mathbf{s}') = T'(s_{o_j}, \pi_{o_j}(s_{o_j}), s'_{o_j})$.

The relative CSMDP has states $\overline{\mathbf{S}} = \mathbf{S} = \{o_j\} \times S$, with $\overline{\mathbf{s}} = \langle o_j, \overline{s}_{o_j} \rangle \in \overline{\mathbf{S}}$, and actions A. By Equation 3.9, $Pr(\overline{\tau}, \overline{\mathbf{s}}' | \overline{\mathbf{s}}, \pi_{o_j}(\overline{s}_{o_j})) = \overline{M}(\mathbf{s}, \mathbf{s}') = T'(\overline{s}_{o_j}, \pi_{o_j}(\overline{s}_{o_j}), \overline{s}'_{o_j})[\overline{s}'_{o_j} = \overline{s}_{o_j}]^{\overline{\tau} - 1}$ for $\overline{\tau} > 0$ with $\overline{i}' = o_j$, and also simply $Pr(\overline{\tau}, \overline{\mathbf{s}}' | \overline{\mathbf{s}}, \pi_{o_j}(\overline{s}_{o_j})) = 0$ for $\overline{\tau} = 0$. Therefore, the time-differential state transition just follows T, with $Pr(d\overline{\tau}, \overline{\mathbf{s}}' | \overline{\mathbf{s}}, \pi_{o_j}(\overline{s}_{o_j})) = T'(\overline{s}_{o_j}, \pi_{o_j}(\overline{s}_{o_j}), \overline{s}'_{o_j})[\overline{\tau} = 1]$. By Equation 3.10, $\overline{F}(\overline{\mathbf{s}}, \pi_{o_j}(\overline{s}_{o_j}), d\overline{\tau}) = [\overline{\tau} = 1]$, akin to Equation 2.24.

By Equation 3.13, $\overline{\mathcal{R}}(\overline{\mathbf{s}}, \pi_{o_j}(\overline{s}_{o_j})) = R(\overline{s}_{o_j}, \pi_{o_j}(\overline{s}_{o_j})) + 0 = R(\overline{s}_{o_j}, \pi_{o_j}(\overline{s}_{o_j}))$, noting that $\overline{F}$ obviates the need to consider $\overline{\rho}$. Also, $\mathcal{P}_{o_j} = \emptyset$ implies by Equation 3.14 that $\mathcal{C}_{o_j} = \emptyset$.

By Equation 3.20, with $\overline{\mathbf{s}} = \langle o_j, \overline{s}_{o_j} \rangle$, we have:

$$\overline{V}^{\pi}_{o_j}(\overline{\mathbf{s}}) = R(\overline{s}_{o_j}, \pi_{o_j}(\overline{s}_{o_j})) + \gamma_{o_j} \sum_{\overline{s}'_{o_j} \in S} T'(\overline{s}_{o_j}, \pi_{o_j}(\overline{s}_{o_j}), \overline{s}'_{o_j}) \overline{V}^{\pi}_{o_j}(\overline{\mathbf{s}}') \tag{3.25}$$

with $\overline{\mathbf{s}}' = \langle o_j, \overline{s}'_{o_j} \rangle$. Hence Equation 3.20 becomes the Bellman equation (Equation 2.2). Importantly, the option policy π_j is the *only* policy available for optimization following Equation 3.19 given that $\Pi_{o_j} = \{\pi_j\}$. Thus, for each option o_j, we can compute the value of the option $\overline{V}^{*}_{o_j}(\overline{\mathbf{s}}^0_{o_j}) = \overline{V}^{\pi_j}_{o_j}(\overline{\mathbf{s}}^0_{o_j})$.

Case v: For vertex v_0, we have $X = \{v_0\} \cup \mathcal{P}_{v_0} = V$. By Equation 3.2, for policies π, $M_{X\pi} = \prod_{o_j \in V} M^{o_j}_{X\pi}$ for the state space $\mathbf{S} = V \times S \times \cdots \times S$ with $\mathbf{s} = \langle i, s_v, s_{o_1}, \ldots, s_{o_k} \rangle \in \mathbf{S}$. The next two paragraphs detail the state transition following Equation 3.3 and $\mu(V, v, s_v, a_v, w, s'_w) = Pr(w, s'_w | v, s_v, a_v)$, by Equation 3.1.

For vertex v, by Equation 3.3, only the first (1) and second (2) cases can be non-zero: (1a) if $i' = v$, $i = v$, and $\pi_v(s_v) \in A$, then $M^v_{X\pi}(\mathbf{s}, \mathbf{s}') = \mu(V, v, s_v, \pi_v(s_v), v, s'_v) = T(s_v, \pi_v(s_v), s'_v)$ follows the state transition as normal; (1b) if $i' = v$, $i = v$, and $\pi_v(s_v) \in \mathcal{O}$, then $M^v_{X\pi}(\mathbf{s}, \mathbf{s}') = \mu(V, v, s_v, \pi_{o_j}(s_v), v, s'_v) = \beta_j(s'_v) T(s_v, \pi_{o_j}(s_v), s'_v)$ considers if transfer of control to the option fails; (1c) if $i' = v$ and $i = o_j$ then $M^v_{X\pi}(\mathbf{s}, \mathbf{s}') = \mu(V, o_j, s_{o_j}, \pi_{o_j}(s_{o_j}), v, s'_v) = \beta_j(s'_{o_j}) T(s_{o_j}, \pi_{o_j}(s_{o_j}), s'_v)$ follows the state transition giving control back to v; and (2) if $i' = o_j$ then $M^v_{X\pi}(\mathbf{s}, \mathbf{s}') = [s'_v = s'_{o_j}]$ makes v copy the state transition of the option o_j in control.

For each option vertex o_j, by Equation 3.3, only the first (1), second (2), and fourth (4) cases can be non-zero: (1a) if $i' = o_j$ and $i = o_j$ then $M^{o_j}_{X\pi}(\mathbf{s}, \mathbf{s}') = \mu(V, o_j, s_{o_j}, \pi_{o_j}(s_{o_j}), o_j, s'_{o_j}) = (1 - \beta_j(s'_{o_j})) T(s_{o_j}, \pi_{o_j}(s_{o_j}), s'_{o_j})$ follows the state transition as normal; (1b) if $i' = o_j$ and $i = v$ then $M^{o_j}_{X\pi}(\mathbf{s}, \mathbf{s}') = \mu(V, v, s_v, \pi_{o_j}(s_v), o_j, s'_{o_j}) = (1 - \beta_j(s'_v)) T_{v o_j}(s_v, \pi_{o_j}(s_v), s'_{o_j})$ follows the state transition as normal; (2) if $i' = v$ yields $M^{o_j}_{X\pi}(\mathbf{s}, \mathbf{s}') = [s'_{o_j} = s'_v]$ so that o_j copies the state transition of controlling vertex v; and (4) if $i' \neq v$ and $i' \neq o_j$, $M^{o_j}_{X\pi}(\mathbf{s}, \mathbf{s}') = [s'_{o_j} = s_{o_j}]$ pauses until control is transferred back to v. Since the options only have one policy π_j to choose from, it does not matter if they update following other options. If desired, a simple edges that fully connect all options to each other will ensure they all use the latest state, but this is not necessary for this proposition.

Thus, by Equation 3.2, we have now defined the full Markov chain $M_{X\pi}(\mathbf{s},\mathbf{s}')$ as the state transition following T and transfer of control following $\mathcal{I}_j$ and β_j.

By Equation 3.5, $\overline{X}=V$. This implies $\underline{X}=\emptyset$, again obviating the need for $\underline{M}$ and $\underline{\lambda}$. By Equation 3.8, only the first case applies, and $[\hat{i}\in\underline{X}]=0$. Thus, we arrive at the state transition $\overline{M}(\mathbf{s},\mathbf{s}')=M_{X\pi}(\mathbf{s},\mathbf{s}')=M^{v}_{X\pi}(\mathbf{s},\mathbf{s}')M^{o_1}_{X\pi}(\mathbf{s},\mathbf{s}')\cdots M^{o_k}_{X\pi}(\mathbf{s},\mathbf{s}')$. For example, the probability that v is in control as normal is computed by: $\overline{M}(\mathbf{s},\mathbf{s}')=T(s_v,\pi_v(s_v),s'_v)[s'_{o_1}=s'_v]\cdots[s'_{o_k}=s'_v]$. Also, as another example, the probability that o_j is in control and does not return control to v is computed by: $\overline{M}(\mathbf{s},\mathbf{s}')=[s'_v=s'_{o_j}][s'_{o_1}=s_{o_1}]\cdots(1-\beta_j(s'_{o_j}))T(s_{o_j},\pi_{o_j}(s_{o_j}),s'_{o_j})\cdots[s'_{o_k}=s_{o_k}]$.

The relative CSMDP has states $\overline{\mathbf{S}}=\mathbf{S}=V\times S\times\cdots\times S$, with $\overline{\mathbf{s}}=\langle i,\overline{s}_v,\overline{s}_{o_1},\ldots,\overline{s}_{o_k}\rangle\in\overline{\mathbf{S}}$, and actions A. By Equation 3.9, we have $Pr(\overline{\tau},\overline{\mathbf{s}}'|\overline{\mathbf{s}},\pi_v(\overline{s}_v))=(\overline{M}\overline{U}^{\overline{\tau}-1})(\mathbf{s},\mathbf{s}')\ [\overline{\tau}>0\wedge\overline{i}'=v]$. By construction of $\overline{M}$ and $\overline{U}$, $Pr(d\overline{\tau},\overline{\mathbf{s}}'|\overline{\mathbf{s}},\pi_v(\overline{s}_v))$ is exactly the probability that the option terminates after $\overline{\tau}$ steps in state $\overline{\mathbf{s}}'$, as stated in the original paper [122].

By Equation 3.13, $\overline{\mathcal{R}}(\overline{\mathbf{s}},\pi_v(\overline{s}_v))=R(\overline{s}_v,\pi_v(\overline{s}_v))+\mathbb{E}[R(\overline{s}^{\tau^t+1}_{o_j},\pi_{o_j}(\overline{s}^{\tau^t+1}_{o_j}))+\cdots]$. Since the underlying $\overline{M}$ is constructed to be identical to that of the SMDP for options, we may write $\mathcal{R}(\overline{\mathbf{s}},\pi_v(\overline{s}_v))$ using Equation 2.38 and the state transition using Equation 2.40 $p^{\mathcal{O}_j}$. By Equation 3.20, with $\overline{\mathbf{s}}=\langle i,\overline{s}_v,\overline{s}_{o_1},\ldots,\overline{s}_{o_k}\rangle$ and $\overline{\mathbf{s}}'=\langle i',\overline{s}'_v,\overline{s}'_{o_1},\ldots,\overline{s}'_{o_k}\rangle$, we have:

$$\overline{V}^{\pi}_{v}(\overline{\mathbf{s}})=\mathcal{R}(\overline{\mathbf{s}}_v,\pi_v(\overline{s}_v))+\sum_{\overline{s}'_v\in S}p^{\mathcal{O}_j}(\overline{s}_v,\overline{s}'_v)\overline{V}^{\pi}_{v}(\overline{\mathbf{s}}'). \tag{3.26}$$

Finally, by Equation 3.14 that $\mathcal{C}_v=\emptyset$, as there are no constraint edges Π_{o_jv}. However, trivially if we included Π_{vv} as described above, then it merely removes options at select states following the $\mathcal{I}_j$. This is equivalent to defining an action set for each state $A(s)$.

Hence both cases show that Equation 3.20 forms an equivalent SMDP, in Markov reward process, resulting in the identical expected reward and value to that of the options framework.

Thus, we have represented any collection of options's states, actions, transition, reward, constraints, and value equations within a policy network. □

3.6 Evaluation

Home healthcare robots serve in household and eldercare scenarios, providing solutions to a wide array of helpful tasks ranging from cleaning to medicine delivery [102]. Surveys conducted by Broadbent *et al.* [21] analyzed and ranked the desired tasks the robot could do to help improve the lives of the elderly. Both elderly people and healthcare staff were surveyed. We focus on a robot

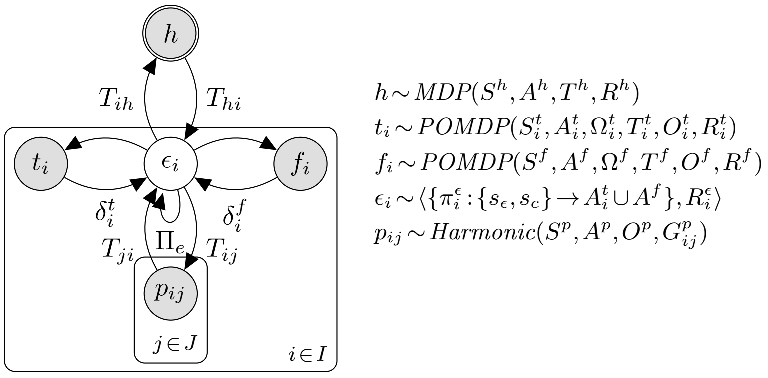

h
T_{ih}
T_{hi}
t_i
ϵ_i
f_i
δ_i^t
δ_i^f
Π_e
T_{ji}
T_{ij}
p_{ij}
$j \in J$
$i \in I$
$h \sim MDP(S^h, A^h, T^h, R^h)$
$t_i \sim POMDP(S_i^t, A_i^t, \Omega_i^t, T_i^t, O_i^t, R_i^t)$
$f_i \sim POMDP(S^f, A^f, \Omega^f, T^f, O^f, R^f)$
$\epsilon_i \sim \langle \{\pi_i^\epsilon : \{s_\epsilon, s_c\} \rightarrow A_i^t \cup A^f\}, R_i^\epsilon \rangle$
$p_{ij} \sim Harmonic(S^p, A^p, O^p, G_{ij}^p)$

Figure 3.4: The policy network for the home healthcare robot.

solution that captures three top-ranked needs: (1) medicine notification and delivery; (2) cleaning; and (3) monitoring and helping with falls.

The few mobile healthcare robots that exist tend towards hand-engineered decision-making systems that work well for their specific implementation [49]. One of the notable exceptions of a general model-based approach involves an early form of hierarchical POMDP [91]. They partitioned a single POMDP's action space into smaller groups, solving a collection of identical POMDPs with differing reduced action spaces. While successful, this early seminal work lacked the generality of a policy network, as policy networks can allow different models to be integrated (state and action spaces), can handle multiple objectives (as in CMDPs), and can leverage a grounding in SMDPs (as in options).

The objective of this robot demonstration is to show how policy networks can be used to solve these large problems. It also implicitly suggests how to think about using policy networks to break up the problem into each sub-component for use within the framework. The demonstration is evaluated by illustrating that a policy network consists of compact easy-to-solve sub-problems, as compared to an otherwise intractable large monolithic (PO)MDP.

Problem Description Consider a healthcare robot with a set of *high-level tasks* it must continuously complete. The *medicate task* is selected to complete by the high-level and requires navigating to the bathroom, retrieving medicine, finding the patient, and delivering it to them. The *clean task* is also selected and requires moving any out-of-place objects back in place while vacuuming. The *monitor task* must operate at all times, reactively interrupting any other task, and requires monitoring and detecting a fall of an elderly person. If confident in the detection, the robot should check on the person and call for a healthcare professional's help. The *low-level path planning* must take special care to safely traverse the house.

Policy Network Vertex Name	V	$\lvert S\rvert$	$\lvert A\rvert$	$\lvert\Omega\rvert$	$\lvert\Gamma\rvert$	Time
High-Level Task Selector	h	16	4	—	—	<0.1
Medicate Task	t_1	289	13	5	1224	159.8
Clean Task	t_2	145	13	3	612	14.0
Fall Monitor/Assist Task	f_i	2	3	2	22	<0.1
Low-Level Path Planner	p_{ij}	17766	9	—	—	0.92

Table 3.1: The problem sizes and run times for each model.

Policy Network Solution Figure 3.4 shows the policy network for the healthcare robot. We provide a description of each vertex below. Also, Table 3.1 shows the problem sizes and results of solving these problems using *nova* [136], with value iteration (VI), point-based VI (PBVI) [90], and harmonic functions [133], for MDP, POMDP, and path planning models, respectively. PBVI has a policy size denoted as $|\Gamma|$. Harmonic function path planning (denoted $Harmonic(\cdot)$) is equivalent to a special class of SSP with uniform state transitions, goals $G^p_{ij} \subset S^p$, and cost of 1 for obstacles $O^p \subset S^p$. We now outline a concise description of each of the five models below, due to space constraints, and will provide all source code with the model details.

The high-level task h handles issues $I = \{t_1, t_2, f_i, \emptyset\}$ with $S^h = 2^I$ and $A^h = I$. Let $\emptyset$ denote a "complete" or "no-op" state and action here. The high-level h transfers control by T_{hi} to the start state of the corresponding task when selected as an action. Let a set of regions $\mathcal{R}$ (e.g., kitchen, bathroom, and bedroom) be given for the map ($|\mathcal{R}| = 12$ in Figure 3.5). The medicate task t_1 has $S^t_1 = \mathcal{R} \times \mathcal{R} \times \{Y, N\} \cup \{\emptyset\}$, for the robot and person regions, as well as if the medicine is carried or not. $A^t_1 = \mathcal{R}$ refer to navigation to a region by the path planner by T_{ij} and T_{ji}. $\Omega^t_1 = \{Y, N\} \times \{Y, N\} \cup \{\emptyset\}$ refers to detection of a person or not, holding medicine or not, and completion. The clean task t_2 is similarly defined with $S^t_2 = \mathcal{R} \times \mathcal{R} \cup \{\emptyset\}$ and $A_2 = \mathcal{R}$ as it searches for a location to clean. $\Omega^t_2 = \{Y, N, \emptyset\}$ refers to detection of a person or not and completion. The monitor task f_i has state space $S^f = \{Y, N\}$ for if the person has fallen and needs help. $A^f = \{call, ask, \emptyset\}$ denotes calling for help, asking if the person needs help, and no-op. $\Omega^f = \{Y, N\}$ refers to detecting a fall or not. The executor ϵ follows MODIA, with a preference for *call* and *ask* over region navigation actions $\mathcal{R}$. T_{ih} transfers control back to h when in a task complete state s_c. The path planner p_{ij} navigates between regions $J = \mathcal{R} \times \mathcal{R}$ in the occupancy grid map. The paths are modeled by solutions to Laplace's equation called harmonic functions [29, 133]. Figure 3.5 shows a full implementation of this policy network on a real robot.

Discussion We observe the policy networks naturally model this home healthcare robot domain—it clearly breaks the problem into tractable subproblems, with clearly defined objectives, and links them together in a theoretically sound manner. Compare this with a traditional *monolithic POMDP* of just t_1, t_2, and f_i. $S = \mathcal{R} \times (\mathcal{R} \times \{Y, N\} \cup \{\emptyset\}) \times (\mathcal{R} \cup \{\emptyset\}) \times \{Y, N\}$ for the robot, patient (holding medicine or not), and clean locations, plus if a fall exists or not, yielding $|S| = 7800$. $A = \mathcal{R} \cup A^f$ and $\Omega = \Omega_1^t \times \Omega_2^t \times \Omega^f$, yielding $|A| = 15$ and $|\Omega| = 30$. These monolithic POMDPs grow *exponentially* in their state, action, and observation spaces as the number of tasks increase, and are shown to be intractable [134]. Also, their reward is diluted as it weighs all tasks in one model and objective, losing semantic meanings in its conflation. Policy networks address these problems by growing *linearly* in the number of tasks and preserving semantic meanings of task objectives.

3.7 Conclusion

We now revisit the questions posed in the introduction. First, how are CMDPs related to options and can these two models be combined? In policy networks, they are different types of edges between collections of distinct models: policy constraints and policy transitions. By simply adding any desired vertices and corresponding edges, we can easily combine both ideas, as evident by the previous section. Second, what does it mean to perform an action? In policy networks, it means conditioning on an action so as to induce a state update in any models that share the same action space. Third, how do these operate when the state and/or action spaces are different? Following the definition of performing an action, any shared action space induces state updates in the collection of models, as in options or more generally SMDPs. With different action spaces, the policies can still affect one another through transfer of control, treated as an abstraction or macro-action. Finally, is there a principled mathematical model that enables the integrated design of multiple models with these concepts? We show in this chapter that policy networks form such a model. They provide a formal approach to generalize select state-of-the-art models and use them within a single reasoning system. The implementation of policy networks demonstrates that they can be used effectively to model and solve a challenging home healthcare robot problem.

Policy networks are fully defined in a complete general form in this thesis; however, this chapter represents ongoing work that will be expanded in a number of ways in future work. First, the nuances and complexity of the policy network's presentation will be streamlined and refined. Second, multiple other models, such as perhaps MAXQ [34] and abstract MDPs (AMDPs) [47], will be proven to be generalized by policy networks. Third, the properties and benefits for using policy networks, such as how conditioning and independence can improve solving them, will be further developed and evaluated. Fourth, more demonstrations and solutions using policy networks will be created, this includes policy network solutions in the domains of home healthcare robots and autonomous vehicles, as well as search-and-rescue robots and delivery robots. Overall, this work lays the foundation for the scalable integration of multiple models in support of reasoning about complex real-world domains for long-term autonomy.

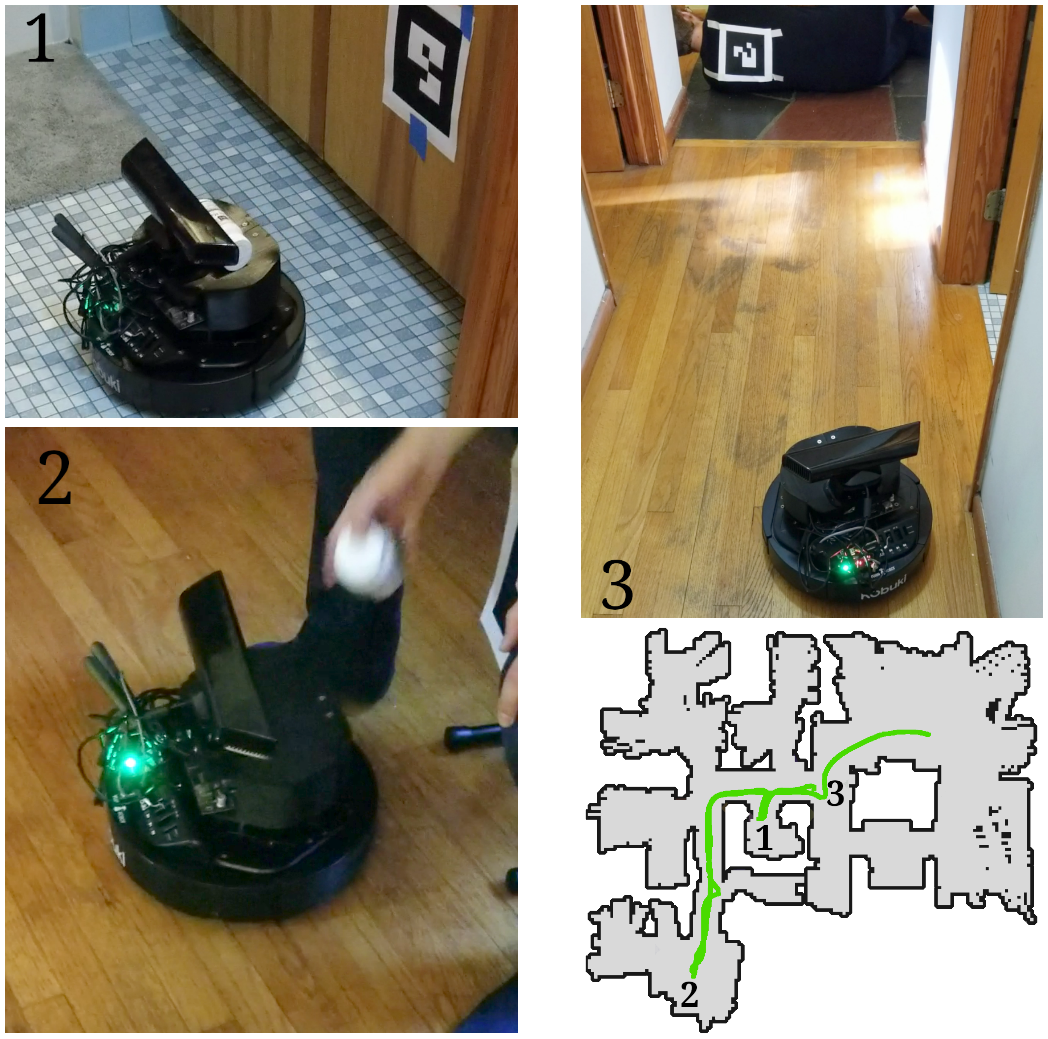


Figure 3.5: Experiments with the home healthcare robot using this policy network in the real household shown above. Three highlights are shown: (1) medicine retrieval for task t_1, (2) medicine delivery completion with transfer $t_1 \to h \to t_2$, and (3) interruption of cleaning task t_2 by detecting a fall with task f_i and calling for assistance.

CHAPTER 4
CONTROLLER FAMILY POLICIES

Policy networks represent a unified general perspective on decision-making with multiple models integrated together. The notion of a policy is crucial to a policy network, as each vertex refers to a policy set for a reward. However, the actual policy computed is almost always approximating the true solution to the MDP or POMDP. This is particularly evident in POMDPs, for which solvers really return a set of α-vectors for belief-points, a finite state controller (FSC), or even filter beliefs through a compression function. How does this representation of policy work with a policy network?

In this chapter, we present a family or class of policies called the controller family. We formally prove that it encapsulates popular POMDP policy forms and their value functions. We include an example of creating novel policy forms within the family, and implement the form in simulation and on a real robot to illustrate its benefits. This controller family policy form is therefore usable in the vertices of a policy network, capturing all common representations of policy under one unified formulation.

4.1 Introduction

The partially observable Markov decision process (POMDP) is one of the most general single agent decision-making models [63]. Over the past two decades, POMDP solution formulations and approximate algorithms have enjoyed rapid growth and increased interest, specifically point-based methods [90], finite state controller policies [3], and compression techniques [107]. With the improved tractability these solutions afford, POMDPs are increasingly used in real world deployed robotic applications ranging from aircraft collision avoidance systems [67] to self-driving cars [134]. However, POMDPs are PSPACE-complete, requiring improved techniques to be developed to facilitate more widespread use. Towards this goal, we provide a novel formulation of POMDP value and policy that unifies the state-of-the-art approaches, gives new insights, and provides a solid foundation on which to build the next generation of solutions which use these improved policy representations.

The current generation of POMDP solutions individually arose from disparate ideas about how to explore small sets of reachable beliefs, define node-based abstractions, or find reduced models representing a similar problem. Each method resulted in a distinct form of policy and value function representation, on top of which an algorithm was constructed. While each approach is successful in its own right, it is not clear how they are related to one another. This knowledge gap manifests itself by the lack of a unified view on the POMDP value equations, policies, and resulting algorithms, limiting the potential of new approaches and leaving many questions unanswered. For example, can we integrate point-based and FSC approaches in a principled manner? Is there a general method to introduce compression into point-based or FSC policy forms? What methods can automatically select the abstracted states (nodes)? In general, what is the relation between expanding more belief points, adding FSC nodes, or exploring the number of bases in a compressed model? From a foundational perspective, we might even ask what the mathematical reason behind the existence of value iteration and policy iteration? Finally, is there an underlying principled framework to design POMDP solutions? This chapter aims at taking steps towards providing answers to these questions.

We present the *controller family* of policy forms as a general formulation of policy and value [140]. It consists of a set of nodes, an action selector function, a node selector function, and a function approximator with a specific form of value function acting as a constraint. These placeholder nodes and functions can be assigned to specific values or left unconstrained. We show how various policy forms emerge by constraining these elements.

As a motivational analogy, consider exponential family distributions which generalize popular distributions such as binomial, exponential, Dirichlet, and Gaussian. The exponential family presents a probability density function with placeholder functions (e.g., sufficient statistic function, natural parameter, and partition function). Constraining these placeholder functions in particular ways produces the specific popular distributions. In the same manner, we define controller family policies with a value function using placeholder functions. Constraining these placeholder functions in particular ways produces specific popular approximate policy forms: point-based, FSC, value-directed compression (VDC), and exponential-family principle components analysis compression (E-PCA). This chapter defines the controller family and rigorously proves it encapsulates these policy forms.

Our primary contributions are: (1) a formal statement of the controller family (Section 4.2); (2) a detailed theoretical analysis mapping the three widely-used policy forms to the controller family (Section 4.3); (3) a discussion of controller family policies within policy networks and the analogies to probablistic graphical models (Section 4.4); and (4) a novel belief-integrated FSC-based policy form with a robot demonstration of its execution and an analysis of its improvement over a vanilla FSC-based policy form (Section 4.5).

4.2 Controller Family Policies

The **controller family** are policies defined by the form $\pi = \langle X, \psi, \eta, \sigma \rangle$. X is a set of r controller nodes, acting as an internal memory for the agent and referring to compact relevant aspects (e.g., belief points, state features, or compressed beliefs). $\psi : X \times \triangle^{|S|} \times A \rightarrow [0,1]$ denotes the stochastic selection of action a at node x and belief b with $\psi(x,b,a) = Pr(a|x,b)$. $\eta : X \times \triangle^{|S|} \times A \times \Omega \times X \rightarrow [0,1]$ denotes the stochastic selection of successor node x' given at node x and belief b, action a was performed yielding observation ω with $\eta(x,b,a,\omega,x') = Pr(x'|x,b,a,\omega)$. Lastly, $\sigma : X \times \triangle^{|S|} \rightarrow \mathbb{R}$ denotes a function approximator of V. Commonly, $\sigma(x,b) = \sum_s b(s)V(x,s)$, with node x's α-vector $V(x,s)$, is used to approximate infinite horizon with the finite horizon α-vectors. Generally, we will see that approximate algorithms assume $V(x,b) = \sigma(x,b)$ to compute their values.

The objective is to find a controller family policy π that maximizes the expected reward:

$$\mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t R(b^t, \pi^t(b^t)|\pi, b^0\Big] \tag{4.1}$$

with b^t denoting a random variable for the belief state at time t generated following T, O, and π^t denoting a random variable for the policy's selected action generated following its controller node over time X from ψ and η. As such, we obtain the **value** of a controller family policy, which depends on the controller node x and the belief b:

$$V(x,b) = \sum_{a \in A} \psi(x,b,a) \Big[R(b,a) + \gamma \sum_{\omega \in \Omega} Pr(\omega|b,a) \sum_{x' \in X} \eta(x,b,a,\omega,x')\sigma(x', b'_{a\omega})\Big] \tag{4.2}$$

with $R(b,a) = \sum_s b(s)R(s,a)$ and $b'_{a\omega}$ following the belief update equation. We often assume initial $x^0 \in X$ and $b^0 \in \triangle^{|S|}$. Also, it is convenient to define $Q(x,b,a)$ by the equation in $[\cdot]$. In the most general form, the **objective** is to compute values for *all components*, including the computation of V, to maximize the function approximator $\sigma(x,b)$ for all x and b. Given an initial node and belief, the objective is: $\max_{X,\psi,\eta,\sigma,V} \sigma(x^0, b^0)$ subject to the definition of $V(x,b)$ in Equation 4.2 and any extra constraints on $\langle X, \psi, \eta, \sigma \rangle$. We provide a general non-linear programming (NLP) formulation of this optimization in Table 4.1. Extra constraints added to the NLP, which are described in the next section, will produce the various approximate policy forms.

We define a controller family policy with a function approximator. Finite stochastic controllers (FSCs) and function approximators have been explored in various forms in-depth before. Critically, this chapter presents this *unified formulation* and a *novel perspective* on the formal core of POMDP algorithm *policy and value representations*. We are not encompassing any specific algorithmic nu-

Function: $\textsc{SolveControllerFamilyPolicyPOMDP}(\cdot)$

Given: $S,\ A,\ \Omega,\ T,\ O,\ R,\ X,\ \sigma,\ x^0 \in X,\ b^0 \in \triangle^{|S|}$

Variables: $\psi(x,b,a),\ \forall x,b,a \qquad \eta(x,b,a,\omega,x'),\ \forall x,b,a,\omega,x' \qquad V(x,b),\ \forall x,b$

Maximize: $V(x^0,b^0)$

Bellman Constraints:

$$V(x,b) = \sum_{a\in A} \psi(x,b,a)\Big[R(b,a) + \gamma \sum_{\omega\in\Omega} Pr(\omega|b,a) \sum_{x'\in X} \eta(x,b,a,\omega,x')\sigma(x',b'_{a\omega})\Big]$$

Function Approximation Constraint: $V(x,b) = \sigma(x,b),\ \forall x,b$

Probability Constraints:

$\psi(x,b,a) \geq 0,\ \forall x,b,a \qquad \sum_{a\in A} \psi(x,b,a) = 1,\ \forall x,b$

$\eta(x,b,a,\omega,x') \geq 0,\ \forall x,b,a,\omega,x' \qquad \sum_{x'\in X} \eta(x,b,a,\omega,x') = 1,\ \forall x,b,a,\omega$

Table 4.1: A general NLP formulation using an unconstrained controller family policy for POMDPs.

ances or details. Our work establishes important new links among the various POMDP *policy and value representations* that are *used* by different algorithms. Early work informally described similar concepts, but lacked any formal results as modern algorithms were not yet developed [83] [19]. Other work compares the models themselves rather than the POMDP's policy and value forms [16] [13]. Surveys have derived algorithmic commonalities among strictly point-based approaches [110]. The remaining literature discussed in the chapter compares algorithms, namely in terms of performance, whereas we find the common threads underlying their design.

At a high-level, we observe that X is a free set and ψ, η, and σ are free functions. If we condition them on the value function and policy, in an appropriate manner, we can enforce a particular structure to focus the resulting policy form. A major contribution of this chapter are the formal proofs that select state-of-the-art approximate policy forms are actually specific instances of a broad family of policies.

4.3 Theoretical Analysis

We now prove various policy forms are members of the controller family. Each time we: (1) define its normal policy form and value, (2) represent its policy as a controller family policy, and (3) prove the resulting value equation is identical.

4.3.1 Optimal Policy Formulations

Importantly, Equation 4.2 does not enable an "improved optimal equation" beyond what is achievable with Equation 2.17. The optimal solution can always be expressed by the original Bellman optimality equation, but the controller family equation does encapsulate it. Interestingly, we can construct policies that are not representable by a simple mapping of belief to action; however, no such policy can ever obtain values higher than that of the optimal formulation. This added flexibility is instead exploited in approximations. These simple but necessary facts are proven in Proposition 4.

Proposition 4. *The optimality policy form is a member of the controller family.*

Proof. We must write Equations 2.17 and 2.20 using Equation 4.2. Let $X = \triangle^{|S|}$, $\psi(x,b,a) = [Q(x,b,a) \geq Q(x,b,a')\forall a']$, $\eta(x,b,a,\omega,x') = [x' = b'_{a\omega}]$, and $\sigma(x,b) = V(x,b)$ with Iverson bracket $[\cdot]$, $b'_{a\omega}$ resulting from the belief update equation, and $b \equiv x$ for each. This produces Equation 2.17. For finite horizon, let $X = \mathcal{R}(b^0)$ with ψ as above. Let $\eta(x,b,a,\omega,x') = [\sum_{s'} b'_{a\omega}(s')V(x',s') \geq \sum_{s'} b'_{a\omega}(s') V(x'',s')\forall x'']$ (i.e., an argmax) with α-vector $V(x,s)$ for belief/node x. Let $\sigma(x,b) = \sum_s b(s)V(x,s)$. This produces Equation 2.20. Also, we can write a policy $\pi : \triangle^{|S|} \rightarrow A$ as the controller family policy $\pi_c = \langle X,\psi,\eta,\sigma\rangle$. Let X, η, and σ be as above. Let $\psi(x,b,a) = [\pi(x) = a]$ since $x = b$. We can add arbitrary stochasticity to any beliefs $x = b$, both in action ψ and successor η, representing policies unobtainable otherwise. $\square$

Insights (1) The *node selection* X is the policy's domain (e.g., beliefs). (2) We write ψ and η as maximizations by constraining them by parts of the equation. (3) Equation 2.19 can be interpreted as: $\eta \equiv \max_\alpha$ and $\sigma \equiv \sum_s b(s)\alpha(s)$.

4.3.2 Point-Based Policy Formulations

Point-based policies avoid the exponential growth of reachable belief points by exploring a subset $B \subseteq \mathcal{R}(b^0)$. Given any belief $b \in \triangle^{|S|}$, we can extract the policy's action $\pi(b) \mapsto a_\alpha$ using $\alpha = \operatorname{argmax}_{\alpha'} \sum_s b(s)\alpha'(s)$. We denote a point-based policy by $\pi = \langle B,\Gamma\rangle$. The point-based update equation, given previous α-vectors Γ', is [90]:

$$\Gamma_{a\omega} = \{[V^{\alpha'}_{a\omega}(s_1),\ldots,V^{\alpha'}_{a\omega}(s_n)]^T, \forall \alpha' \in \Gamma'\}, \qquad \Gamma_b = \{R(\cdot,a) + \sum_{\omega\in\Omega} \operatorname*{argmax}_{\alpha'\in\Gamma_{a\omega}} \sum_{s\in S} b(s)\alpha'(s), \forall a \in A\},$$

$$\Gamma = \{\operatorname*{argmax}_{\alpha\in\Gamma_b} \sum_{s\in S} b(s)\alpha(s), \forall b \in B\} \tag{4.3}$$

with $V^{\alpha'}_{a\omega}(s) = \gamma \sum_{s'} O(a,s',\omega)T(s,a,s')\alpha'(s')$ and vector $R(\cdot,a) = [R(s_1,a),\ldots,R(s_n,a)]^T$. Each initial $\alpha(s) = \min_{s'} \min_{a'} R(s',a')/(1-\gamma)$ to ensure α-vectors weakly monotonically increase [79].

The original forms of point-based methods apply Equation 2.20 on a fixed grid over the belief simplex [79]. Point-based value iteration (PBVI) in Equation 4.3 originally operated over all beliefs, selecting them by solving linear programs to find "witness" (i.e., improvable in value) beliefs [145]. The tractable general incarnation of PBVI explores reachable beliefs and interleaves belief updates with belief expansion techniques [90]. Perseus does all belief expansion initially then intelligently orders the beliefs to do less updates overall, as an α-vector can improve many beliefs simultaneously [118]. HSVI2 [113] and SARSOP [72] both have a tighter interleaving of update and expansion, maintaining lower and upper bounds to test convergence and cleverly selecting action-observation pairs to tighten these bounds. Recent work suggests a modular approach to mix the algorithms' components, which can be competitive in some cases [110].

The formal mapping between point-based and controller family policies is in Proposition 5.

Proposition 5. *Point-based policies are a member of the controller family.*

Proof. We must write Equation 4.3 using Equation 4.2. Let $X = B$ with each $x \in X$ corresponding to a point-based belief $x \in B$, distinct from the current belief $b \in \mathcal{R}(b^0)$. Let $\sigma(x,b) = \sum_s b(s)V(x,s) = \sum_s x(s)V(x,s)$ always requiring $x = b \in B$. This is the critical assumption that makes point-based algorithms work: to compute the values the only beliefs that matter are in B. Since the objective only optimizes values over X, it removes the dependence on $\mathcal{R}(b^0)$. We rewrite Equation 4.2, noting $Pr(\omega|b,a)$ cancels with $b'_{a\omega}$, as:

$$V(x,s) = \sum_{a \in A} \psi(x,b,a) \Big[R(s,a) + \sum_{\omega \in \Omega} \sum_{x' \in X} \eta(x,b,a,\omega,x') \gamma \sum_{s' \in S} T(s,a,s') O(a,s',\omega) V(x',s') \Big]$$

with $\sum_s b(s)$ moved outside the summations such that $V(x,b) = \sum_s b(s)V(x,s)$. Simply reference x and x' here as α and α' to rename $\alpha(s) = V(x,s)$ and $\alpha'(s') = V(x',s')$. Recognize $V^{\alpha'}_{a\omega}(s)$ to obtain:

$$\alpha(s) = \sum_{a \in A} \psi(x,b,a) \Big[R(s,a) + \sum_{\omega \in \Omega} \sum_{x' \in X} \eta(x,b,a,\omega,x') V^{\alpha'}_{a\omega}(s) \Big].$$

As Proposition 4, $\psi(x,b,a) = [Q(x,b,a) \geq Q(x,b,a') \forall a']$ and $\eta(x,b,a,\omega,x') = [\sum_s b(s) V^{\alpha'}_{a\omega}(s) = \sum_s b(s) V^{\alpha''}_{a\omega}(s) \forall \alpha'']$, both equivalent to argmax. This produces Equation 4.3. Optionally, we could use the more flexible probabilistic softmax function, such as:

$$\eta(x,b,a,\omega,x') = \frac{\exp\{\sum_s b(s) V^{\alpha'}_{a\omega}(s)/\tau\}}{\sum_{x''} \exp\{\sum_s b(s) V^{\alpha''}_{a\omega}(s)/\tau\}} \tag{4.4}$$

with softmax *temperature* $\tau \to 0^+$. Also, we can write a point-based policy $\pi_p = \langle B, \Gamma \rangle$ as a controller family policy $\pi_c = \langle X, \psi, \eta, \sigma \rangle$ as in Proposition 4. □

Insights (1) Node selection X is the set of explored beliefs. (2) *Node iteration*—iterating to improve the set of selected nodes—is what most point-based approaches rely on. (3) *Node (successor) selector* η and *action selector* ψ can be written with softmax, enabling easy derivatives [19].

4.3.3 Finite State Controller Policy Formulations

Finite state controller (FSC) methods instead describe a policy as an FSC that is executed from an initial belief [51]. An FSC policy is defined by $\pi = \langle X, \hat{\psi}, \hat{\eta} \rangle$. X is a set of nodes. $\hat{\psi} : X \times A \rightarrow [0,1]$ and $\hat{\eta} : X \times A \times \Omega \times X \rightarrow [0,1]$ ignore any dependence on an explicitly maintained belief and follow the FSC alone. Policy iteration (PI) is commonly used with FSC policy representations. PI alternates between policy evaluation and policy improvement steps, though techniques exist to perform them simultaneously [3]. Evaluating policy π requires solving a system of equations:

$$V(x,s) = \sum_{a \in A} \hat{\psi}(x,a) \Big[R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') \sum_{\omega \in \Omega} O(a,s',\omega) \sum_{x' \in X} \hat{\eta}(x,a,\omega,x') V(x',s') \Big] \tag{4.5}$$

with $V(x,b) = \sum_s b(s) V(x,s)$, $x^0 \in X$, and $b^0 \in \triangle^{|S|}$. This formula is derived from the recognition and use of a cross-product MDP formed from states and controller nodes.

The original PI defines a policy as bounded regions on the belief simplex (e.g., resulting from Γ) and converts it to an FSC for policy evaluation [117]. This proved intractably complex. Instead the policy itself can be represented as an FSC. Policy improvement performs the Bellman update in Equation 2.20, assigns these α-vectors as new potential nodes, then adds, merges, or prunes them [51]. To avoid exponential growth, bounded PI (BPI) explores single new node at each iteration, at the cost of getting stuck in local optima [95]. Point-based PI (PBPI) does Hansen's PI but uses the point-based update in Equation 4.3 [59]. Recent work casts both evaluation and improvement as one non-linear program (NLP) [3]. A dual formulation exists, but only for deterministic FSCs [70].

The straight-forward but necessary mapping from FSC to the controller family is in Proposition 6.

Proposition 6. *FSC policies are a member of the controller family.*

Proof. We must write Equation 4.5 using Equation 4.2. Let X be the same in both. Let $\psi(x,b,a) = \hat{\psi}(x,a)$ and $\eta(x,b,a,\omega,x') = \hat{\eta}(x,a,\omega,x')$ simply ignore the current maintained b. Let $\sigma_c(x,b) = \sum_s b(s) V(x,s)$. Rewrite Equation 4.2, apply the definition of $Pr(\omega|b,a)$, and recognize we can again pull $\sum_s b(s)$ outside the summations to obtain Equation 4.5 with $V(x,b) = \sum_s b(s) V(x,s)$. For deterministic FSCs, we also ensure $\hat{\psi}$ and $\hat{\eta}$ are 0 or 1 via parameters $y_x \in A$ and $z_{xa\omega} \in X$ such that we also have $\hat{\psi}(x,a) = [a = y_x]$ and $\hat{\eta}(x,a,\omega,x') = [x' = z_{xa\omega}]$. Also, we must write an FSC policy $\pi_f = \langle X_f, \psi_f, \eta_f \rangle$ as a controller family policy $\pi_c = \langle X_c, \psi_c, \eta_c, \sigma_c \rangle$, which can be done as above. □

Insights (1) Node iteration to improve X occurs in BPI and PBPI. (2) Nodes are constrained (i.e., fixed-sized controller) in the NLP solution, but it does *simultaneous value and policy iteration* because V, ψ, and η are free variables. (3) Parameters can be added to a controller family's elements with a fixed structure encoding these parameters in the elements (e.g., y for ψ, and z for η, above).

4.3.4 Compression Policy Formulations

Compression techniques construct a mapping from the original POMDP to a smaller reduced model. There are two standard general approaches that we consider: value directed compression (VDC) [94] and exponential family principle components analysis (E-PCA) [107]. Both approaches attempt to find a function $f : \triangle^{|S|} \rightarrow \triangle^r$ that projects beliefs in n dimensional space to a lower $r \leq n$ dimensional space. In PCA and linear VDC, this takes the form of a matrix $F \in \mathbb{R}^{n \times r}$ that acts as a change of basis. In general, this function f is applied to a set of beliefs $B \subseteq \triangle^{|S|}$ to produce an approximate POMDP described by $\tilde{B}$, $\tilde{T} : \tilde{B} \times A \times \tilde{B} \rightarrow [0,1]$, and $\tilde{R} : \tilde{B} \times A \rightarrow \mathbb{R}$. We prove that all these forms are representable in the controller family.

Each approximate belief $\tilde{b} \in \triangle^r$ is projected from some original belief $b \in \triangle^{|S|}$ by $\tilde{b}^T = b^T F$ for linear and $\tilde{b} = f(b)$ for non-linear. For the linear case, a belief b can be reconstructed as $\hat{b} \approx b$ with $\hat{b} = (F^\dagger)^T \tilde{b}$; orthonormal columns yields $\hat{b} = F\tilde{b}$. For the non-linear case, a belief b can be reconstructed with a function inverse $\hat{b} = f^{-1}(\tilde{b})$. In general, we consider these beliefs to be non-negative and renormalized, as both VDC and E-PCA papers discuss. If they are not, then the resulting compressed model would not be a valid (PO)MDP, as beliefs and state transitions must be probabilities over $\triangle^{|S|}$. It does not necessarily affect the change of basis, and without it algorithms can run into issues. In the cases when it does still work, it falls outside the proper definition of a (PO)MDP.

Value Directed Compression VDC compresses the beliefs with either a lossless or lossy f creating a smaller POMDP on which any solver can be applied [94]. To focus our analysis, we consider the *linear* VDC using F as it is the favored form to extend [76] [126] [130]. Similar logic applies for general approaches using non-linear f. The original model forms lossless compression F by *Krylov iteration*. Let $R^a \in \mathbb{R}^{|S|}$ be defined as $R^a_i = R(s_i, a)$. Let $T^{a\omega} \in \mathbb{R}^{n \times n}$ be defined as $T^{a\omega}_{ij} = Pr(s_j, \omega | s_i, a) = T(s_i, a, s_j) O(a, s_j, \omega)$. Krylov iteration methods start with $F_{1j} = R^a_j$. Then, for each a and ω, they iteratively assign $F_{\cdot i+1} = T^{a\omega} F_{\cdot i}$ if it is linearly independent for all $F_{\cdot 1}$, …, $F_{\cdot i-1}$ [94]. Lossy variants can compute F by linear programming or truncated Krylov iteration. Truncated Krylov iteration selects up to a fixed $r \leq n$ and removes the largest error column in F at each step. Other lossy variants employ non-negative matrix factorization (NMF) over a given fixed set of beliefs, either by orthogonal NMF to ensure $FF^{\dagger} \approx I$ [76] or by locality preserving NMF to ensure Lipschitz continuity is preserved in the compressed model [126].

The VDC compressed problem requires that $R^a = F\tilde{R}^a$ and $T^{a\omega}F = F\tilde{T}^{a\omega}$. Since the columns of F are linearly independent by construction, we take the pseudoinverse to derive the compressed model:

$$\tilde{R}^a = F^{\dagger}R^a \qquad \text{and} \qquad \tilde{T}^{a\omega} = F^{\dagger}T^{a\omega}F. \tag{4.6}$$

The policy $\tilde{\pi} : \triangle^r \to A$ is be evaluated with $\tilde{V}^{\tilde{\pi}} : \triangle^r \to \mathbb{R}$ by:

$$\tilde{V}^{\tilde{\pi}}(\tilde{b}) = \tilde{b}^T \tilde{R}^{\tilde{\pi}(\tilde{b})} + \gamma \sum_{\omega \in \Omega} \tilde{V}^{\tilde{\pi}}(\tilde{b}^T \tilde{T}^{\tilde{\pi}(\tilde{b})\omega}). \tag{4.7}$$

Lemma 1. *The linear VDC policy is a member of the controller family.*

Proof. We must write Equation 4.7 as Equation 4.2. Let $X = \hat{B} \subseteq \triangle^{|S|}$ be all possible reconstructed beliefs, with each $x = \hat{b}$. For example, if starting at $x = b^0$, $X = \hat{\mathcal{R}}(b^0)$ are all possible reconstructed beliefs following $T^{a\omega}$ and F. Let $\psi(x, b, a) = [\tilde{\pi}(\tilde{b}) = a]$, since VDC uses a policy evaluation form. Let $\eta(x, b, a, \omega, x') = [\hat{b}^T T^{a\omega} = x']$. Let $\sigma(x, b) = V(x, \hat{b})/\|\hat{b}\|_1 = \hat{b}^T\hat{\alpha}/\|\hat{b}\|_1$, with the α-vector dot product, and $\hat{\alpha} = F\tilde{\alpha}$. By the definition of $\sigma(x, b)$, the only beliefs ever visited are the reconstructed $\hat{b}$ corresponding to x. Apply to Equation 4.2:

$$V(x, \hat{b}) = R(\hat{b}, \tilde{\pi}(\tilde{b})) + \gamma \sum_{\omega \in \Omega} Pr(\omega|\hat{b}, \tilde{\pi}(\tilde{b}))V(x', \hat{b}^T T^{\tilde{\pi}(\tilde{b})\omega})/\|\hat{b}^T T^{\tilde{\pi}(\tilde{b})\omega}\|_1$$

with $x = \hat{b}$, $\tilde{b}^T = \hat{b}^T F$, and $x' = \hat{b}^T T^{\tilde{\pi}(\tilde{b})\omega}$ from η's definition. It is simple to show that $\|\hat{b}^T T^{\tilde{\pi}(\tilde{b})\omega}\|_1 = Pr(\omega|\hat{b}, \tilde{\pi}(\tilde{b}))$. Apply this fact with the definitions of R (as vectors) and V:

$$V(x, \hat{b}) = \hat{b}^T R^{\tilde{\pi}(\tilde{b})} + \gamma \sum_{\omega \in \Omega} \hat{b}^T T^{\tilde{\pi}(\tilde{b})\omega} F\tilde{\alpha}.$$

Apply definition of $\hat{b}$, and then recognize both $\tilde{R}$ and $\tilde{T}$:

$$\begin{aligned} V(x, \hat{b}) &= \tilde{b}^T F^{\dagger} R^{\tilde{\pi}(\tilde{b})} + \gamma \sum_{\omega \in \Omega} \tilde{b}^T F^{\dagger} T^{\tilde{\pi}(\tilde{b})\omega} F\tilde{\alpha} \\ &= \tilde{b}^T \tilde{R}^{\tilde{\pi}(\tilde{b})} + \gamma \sum_{\omega \in \Omega} \tilde{b}^T \tilde{T}^{\tilde{\pi}(\tilde{b})\omega} \tilde{\alpha}. \end{aligned}$$

With $\tilde{b}$ we know x and $\hat{b}$, so we can rename $V(x, \hat{b}) = \tilde{V}(\tilde{b})$. Recognize $\tilde{V}(\tilde{b}) = \tilde{b}^T\tilde{\alpha}$ to obtain Equation 4.7. Also, we can rewrite any policy $\tilde{\pi} : \triangle^r \to A$ as the controller family policy $\pi = \langle X, \psi, \eta, \sigma \rangle$ by assigning the elements be as above. □

Exponential Family Principle Component Analysis E-PCA computes a non-linear compression of the reachable beliefs using E-PCA [107] and applies *fitted value iteration* [48] on the resultant reduced belief MDP. E-PCA is used over PCA to better represent the fact that beliefs are probabilities. This generalized form requires a *link function* $\ell : \triangle^{|S|} \rightarrow \triangle^{|S|}$. The exponential link function is given by $\ell(F\tilde{b}) = \exp\{F\tilde{b}\}$ with $F \in \mathbb{R}^{n \times r}$. It enables us to recover a belief $b \approx \hat{b}$ with $\hat{b} = \exp\{F\tilde{b}\}$. Thus, the link function acts an inverse $f^{-1}(\tilde{b}) = \hat{b} = \ell(F\tilde{b})$. In general, the link function determines the type of exponential family random variable. A loss function is then defined by minimizing the generalized Bregman divergence between b and $\hat{b}$. For the choice of an exponential link function, we have a Poisson belief error model, and equates to minimizing unnormalized KL divergence. The solutions to F and $\tilde{B}$ are convex optimization problems solvable using the partial derivatives of this loss function. The process for $\tilde{B}$ is also used to project belief b to the compressed belief space $\tilde{b}$, defining $f(b) = \tilde{b}$. Fitted value iteration (FVI) is used in the E-PCA compressed belief MDP with a averager function approximator. E-PCA policy $\tilde{\pi} : \tilde{B} \rightarrow A$ is determined from value $\tilde{V} : \tilde{B} \rightarrow \mathbb{R}$:

$$\tilde{V}(\tilde{b}) = \max_{a \in A} \tilde{R}(\tilde{b}, a) + \gamma \sum_{\tilde{b}' \in \tilde{B}} \tilde{T}(\tilde{b}, a, \tilde{b}')\tilde{V}(\tilde{b}') \tag{4.8}$$

with k-nearest neighbors computing the nearest $k > 0$ belief neighbors such that $w : \tilde{B} \times \triangle^r \rightarrow [0, 1]$ and $w(\tilde{b}, \tilde{b}') = (1/k)$ if $\tilde{b}$ is one of k closest beliefs from $\tilde{B}$ for $\tilde{b}'$, with $w(\tilde{b}, \tilde{b}') = 0$ otherwise. The reward and state transition are defined, with $\hat{b} = f^{-1}(\tilde{b})$ and $\tilde{b}'_{a\omega} = f(\hat{b}'_{a\omega})$, as:

$$\tilde{R}(\tilde{b}, a) = \sum_{s \in S} \hat{b}(s) R(s, a) \qquad \text{and} \qquad \tilde{T}(\tilde{b}, a, \tilde{b}') = \sum_{\omega \in \Omega} Pr(\omega | \hat{b}, a) w(\tilde{b}', \tilde{b}'_{a\omega}).$$

Lemma 2. *The E-PCA policy is a member of the controller family.*

Proof. We must write Equation 4.8 as Equation 4.2. Let $X = \tilde{B}$ be all compressed beliefs considered, with $\hat{b} = f^{-1}(\tilde{b})$. Let $\psi(x, b, a) = [Q(\hat{b}, a) \geq Q(\hat{b}, a') \forall a']$, since E-PCA's FVI uses an optimality form. Let $\eta(x, b, a, \omega, x') = w(\tilde{b}', \tilde{b}'_{a\omega})$, noting that from $x = \tilde{b}$ we compute $\hat{b}$, then $\hat{b}'_{a\omega}$, and finally $\tilde{b}'_{a\omega}$. Let $\sigma(x, b) = V(x, \hat{b})$ with $\hat{b}$ computed from $x = \tilde{b}$. By the definition of $\sigma(x, b)$, the only beliefs ever visited are the reconstructed $\hat{b}$ corresponding to x. Apply this to Equation 4.2 to yield:

$$V(x, \hat{b}) = \max_{a \in A} \sum_{s \in S} \hat{b}(s) R(s, a) + \gamma \sum_{x' \in X} \sum_{\omega \in \Omega} Pr(\omega | \hat{b}, a) w(\tilde{b}', \tilde{b}'_{a\omega}) V(x', \hat{b}')$$

with the definition of R and reordering of summations. Rename $V(x,\hat{b})=\tilde{V}(\tilde{b})$ since $\tilde{b}$ uniquely determines x and $\hat{b}$. Recognize $\tilde{R}$ and $\tilde{T}$ to obtain Equation 4.8. Also, we can rewrite any policy $\tilde{\pi}:\triangle^r\rightarrow A$ as the controller family policy $\pi=\langle X,\psi,\eta,\sigma\rangle$ by assigning the elements as above. $\square$

Compression Policy Formulations Proposition 7 formally combines the above lemmas, showing linear and non-linear compression forms are generalized and remarks on other subsumed methods.

Proposition 7. *Compression policies are members of the controller family.*

Proof. By Lemmas 1 and 2, we find that the controller family generalizes compression equations and policy representations. Since VDC generalizes state aggregation [17], model minimization [45], and linear predictive state representations (PSR) [78], so too does the controller family. Additionally, since the proof is independent of ℓ, controller family generalizes any choice of ℓ for E-PCA. $\square$

Insights (1) Node selection *is the compression technique itself*, since X is the compressed or reconstructed beliefs. (2) Function approximators σ make value constant over the belief; instead, the value's belief (i.e., b in $V(x,b)$) is replaced with node's reconstructed belief. (3) Node selection are thus assigned to ensure the node's belief properly updates.

4.4 Policy Networks with Controller Family Policies

The controller family as defined here represents a general form of policy and value for POMDPs. As the POMDP generalizes the discrete MDP, this policy form can also generalize controllers for use in an MDP. Policy networks define vertices as a policy set for a reward function. Typically, we define a policy as either deterministic or stochastic, with this policy set being a subset or equal to the full space of such policies. A natural extension is to define this policy set more generally with a controller family policy. This allows the value, objective, and policy to be as general as possible.

Formally, a policy network $\langle V,E\rangle$ using a controller family set of policies, for any $v\in V$ we are given the fixed controller nodes X and function approximator $\sigma_v:X_v\times\triangle^{|S_v|}\rightarrow\mathbb{R}$. The policy set for v is simply $\Pi_v=\{\langle\psi_v,\eta_v\rangle\}$ for action selector $\psi_v:X_v\times\triangle^{|S_v|}\times A_v\rightarrow[0,1]$ and node successor selector $\eta_v:X_v\times\triangle^{|S_v|}\times A_v\times\Omega_v\times X_v\rightarrow[0,1]$. In other words, it is any assignment of action selector ψ_v and node successor selector η_v.

The policy network Bellman equation in Equation 3.20 can be generalized using a controller family policy form. Two important points regarding the state must be first defined. First, any finite MDP state can be generalized by a POMDP belief state. Mechanically, the semantic change necessary is to consider reachable beliefs $\overline{\mathbf{B}}$ instead of just reachable states $\overline{\mathbf{S}}$ (i.e., $\overline{\mathbf{s}}=\overline{\mathbf{b}}$ and $\overline{\mathbf{S}}=\overline{\mathbf{B}}$).

Domain		BI-FSC-Based Policy			FSC-Based Policy	
Name	$\lvert N\rvert$	Time	ADR	$\lvert B\rvert$	Time	ADR
Aloha-10	10	109.2	**523.0**	50	**15.0**	163.0
Grid-4x3	5	**16.4**	**0.83**	10	18.8	0.58
Hallway2	7	155.4	0.25	10	**67.2**	0.27
Tiger	3	**13.8**	**11.7**	6	20.7	-20.0

Table 4.2: Results using new policy form with same NLP algorithm and same number of FSC nodes ($|N|$). Metrics: Time (seconds, 10 trials) and average discounted reward (ADR, 100 trials). Bold indicates statistically significant improvement.

Both consider transfer of control to ancestors in the same manner as described in Chapter 3. Second, the controller nodes $\overline{\mathbf{x}} \in \overline{\mathbf{X}}$ are essentially considered part of the state, except its stochasticity is governed by the η_v after each action is performed and observation is made. These actions are performed following the ancestors, which result in a belief MDP state transition (i.e., observation emitted) in the same manner as described in Chapter 3. In summary, while we use POMDP notation here for consistency, it is equivalent to an underlying belief MDP. For example, for vertex $v \in V$, the most general value equation $\overline{V}$ for node $\overline{\mathbf{x}}$ and belief $\overline{\mathbf{b}}$ would be:

$$\overline{V}^{\pi}(\overline{\mathbf{x}},\overline{\mathbf{b}}) = \sum_{a\in A_v} \psi(\overline{x}_v,\overline{b}_v,a)\Big[\overline{\mathcal{R}}_v(\overline{\mathbf{x}},\overline{\mathbf{b}},a) + \sum_{\overline{\mathbf{x}}'\in \overline{X}}\sum_{\overline{\mathbf{b}}'\in\overline{\mathbf{B}}}\sum_{\overline{\tau}=1}^{\infty}\gamma^{\overline{\tau}} Pr(d\overline{\tau},\overline{\mathbf{x}}',\overline{\mathbf{b}}'|\overline{\mathbf{x}},\overline{\mathbf{b}},a)\overline{\sigma}^{\pi}(\overline{\mathbf{x}}',\overline{\mathbf{b}}')\Big]. \quad (4.9)$$

We will leave the detailed analysis of this complex general form to future work.

4.5 Evaluation

Now we demonstrate the efficacy of the controller family as a tool for describing novel policy and value representations. Importantly, the main contribution of this chapter remains the formulation of controller family and theoretical analysis, rather than this new example of a controller family policy. Recall, the controller family is *not an algorithm*; it is a representation of policy and value. This is included to provide evidence of its usefulness and guidance for how to create new forms of policy.

Intuitively, we define a belief-integrated FSC as a member of the controller family with some FSC nodes with some belief point nodes. FSC nodes follow the free stochastic successor node selector (e.g., NLP). Beliefs nodes follow an argmax (e.g., PBVI). The belief points used, however, are reconstructed from compressed beliefs (e.g., E-PCA). The resulting value equation is amenable to node iteration via PBVI's expand step followed by E-PCA, and policy iteration via an NLP. Formally, a **belief-integrated FSC (BI-FSC)** is a controller family policy $\pi = \langle X, \psi, \eta, \sigma\rangle$ that has

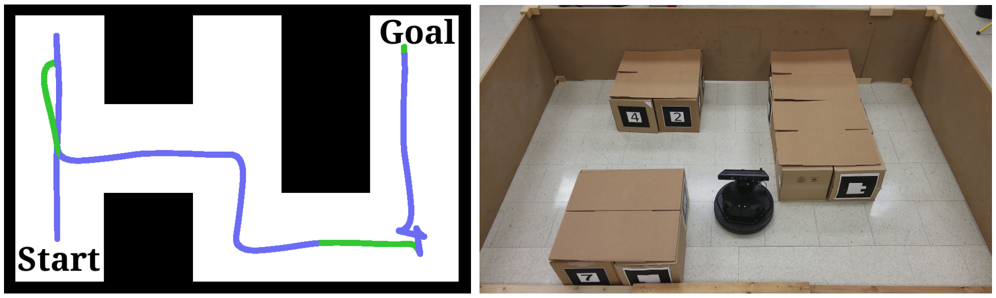


Figure 4.1: Robot experiment. Path: FSC (blue), belief (green) actions.

its nodes X represent both FSC nodes *and* compressed belief-points. Let $X = N \cup \tilde{B}$ with N a set of FSC nodes and $\tilde{B} \subseteq \triangle^{|S|}$. Let $\lambda \in [0,1]$ be a weight between both approaches. Let $\eta(x,b,a,\omega,x')$ be either: $(1-\lambda)\bar{\eta}(x,b,a,\omega,x')$ if $x,x' \in N$; λsoftmax over $\tilde{B}$ if $x \in N$ and $x' \in \tilde{B}$; or $\bar{\eta}(x,b,a,\omega,x')$ if $x \in \tilde{B}$ and $x' \in N$. We can use a Taylor series approximation of softmax (e.g., the 0-th order is $1/|\tilde{B}|$). Let $\sigma(x,b)$ be either: $\sum_s b(s)V(x,s)$ if $x \in N$; or $\sum_s \hat{b}'_{a\omega}(s)V(x,s)$ if $x \in \tilde{B}$ and $\hat{b} = f^{-1}(x)$.

Understanding the BI-FSC Policy Form What does this novel policy form look like in practice? Figure 4.1 shows results from real robot POMDP navigation. The POMDP has S as a 7×5 grid, A as eight directions and stop, and Ω as the "bump" sensor. We solve the BI-FSC with an NLP using SNOPT [44] on the NEOS Server [32]. The traversed path includes intentional "feeling" for walls with successful localization. Interestingly, this policy uses the interspersed belief nodes' decisions to help guide the stochastic FSC nodes' decisions to the goal (blue and green in Figure 4.1).

How does this new policy form improve performance over just a pure FSC policy form? Table 4.2 shows results of a pure FSC-based policy versus a BI-FSC-based policy ($\lambda = 0.5$) on standard benchmark domains. The NLP algorithm is used by *both* policy forms to *isolate a direct comparison* of different policy forms, instead of the algorithms used on top of a chosen policy form. Experiments were run with SNOPT on the NEOS Server. BI-FSC policies are suited for domains that have many local optima (e.g., Aloha-10 and Tiger) in which solvers like SNOPT easily get stuck. BI-FSCs obviate the issue by infusing important beliefs to help find the global optima. Thus, BI-FSCs are able to greatly outperform FSC-based policies in ADR (especially Aloha-10 and Tiger) and can even improve time.

4.6 Conclusion

We now revisit the questions from the introduction. How can we *simultaneously* integrate belief point-based and FSC-based techniques? Nodes can be defined to be beliefs *and* FSC nodes, with the successor node selector to be a free variable or argmax. Is there a way to introduce compression into policy forms? Simply select compressed or reconstructed beliefs for some or all nodes; other elements (e.g., successor node selectors) use these reconstructed beliefs. Can we automatically select some of the nodes (i.e., *node iteration*)? Simply use any prior approach appropriate to the type of node—belief nodes use belief exploration (e.g., HSVI2), FSC nodes use node addition/pruning (e.g., BPI), or compressed nodes use compression (e.g., E-PCA). In general, how are iteration techniques related? Each is an operator on a different type of node as described above.

Lastly, what is the mathematical reason for the existence of value and policy iteration? From the perspective offered by the controller family, it is determined by the action selector ψ. When ψ is constrained to a fixed function, we have value iteration. This assignment prevents policy iteration because the policy is determined by the other free elements; the values V determine the mapping from belief to action. Conversely, when ψ is at all unconstrained, V and ψ are both free and can be computed separately (e.g., policy iteration's evaluation/improvement steps) or together (e.g., NLP).

In conclusion, is there an underlying principled framework to design POMDP policies? This chapter defines the controller family as an answer to this question. We show they generalize the policy and value representations used by state-of-the-art solutions. To validate its effectiveness, we construct a novel policy formulation that infuses beliefs into an FSC. We demonstrate this improved policy form's execution on a real robot acting in the world, and show it overcomes some well-known issues with a vanilla FSC. Finally, we will provide our source code with the goal of building new controller family policies to improve POMDP solutions under this unified formal language with the greater research community.

CHAPTER 5
MULTI-OBJECTIVE DECISION-MAKING

In this chapter, we present a general model for reasoning about multiple objectives. Leveraging a graph of constraining relationships among the objective functions, we allow each objective to select policies from a restricted space of policies that ensure all ancestor objectives are within a slack of their optimal value. We provide both optimal and approximate algorithms, and show how it can be used for multi-objective autonomous vehicle route planning.

This model demonstrates a generalization of two previous multi-objective models. Moreover, it is a special case of a policy network in which all edges simply form a directed acyclic graph of constraints in the same state-action space. These relationships are proven here. In general, this multi-objective decision-making model is useful for any domain in which there is an easily identifiable preference structure for higher priority domain objectives to be satisfied before lower priority domain objectives.

5.1 Introduction

Reasoning about multiple objectives is prevalent in many real-world problem domains such as water reservoir control [23], industrial scheduling [1], energy-conserving smart environments [75], anthrax outbreak detection [116] and autonomous vehicles [142, 135]. Multi-objective Markov decision processes (MOMDPs) represent a model of multiple objectives with two main methodologies to structure their typically conflicting nature: scalarization and preference orderings. Scalarization approaches attempt to weigh each objective properly in a complex function, creating a single-objective MDP which can be solved with standard techniques [103]. However, finding this scalarziation function is non-trivial, and suffers from both computational complexity issues and the conflation of the reward function, losing any semantic meaning the objectives might have once had. We instead leverage the latter, using a preference ordering over objectives [81, 114, 43, 2, 142, 135]. We assign a preference structure and only considers other objectives in the case of tie-breaking, combined with the notion of slack or constraints to liberate successive objectives' choice. In our proposed approach, this ordering is defined by the topological order of a directed acyclic graph (DAG) over the constraints. We call this general decision-making model a *topological MDP (TMDP)*, and its partially observable version a *topological POMDP (TPOMDP)*.

Previous work called ordinal dynamic programming, for both finite [81] and infinite [114] horizon MDPs, placed a strict chain ordering over objectives. This structure is also called a lexicographic MDP (LMDP). Ties among equivalently value-optimal policies were broken by evaluating successive objectives. However it is rare that many policies are even available for each successive objective because they must be exact matches to other maximally optimal policies. Ordinal dynamic programming was explored within reinforcement learning [43]. This work included a notion of a minimum criterion value to allow more policies for successive objectives to explore. Later work began to apply these linearly chained orderings for robot path planning [82], but the problem still remains. One model that does overcome this issue is called a constrained MDP (CMDP) [2]. It instead defines a flat equal structure over the objectives, instead of an iterated linear chain, with some slack that allows them to be reduced up to some constraint value. Our work presents a model that generalizes both lexicographic and constrained MDPs, allowing for slack when tie-breaking among objectives in a topologically-ordered graph of constraints.

The partially observable cases of multi-objective POMDPs (MOPOMDP) have been explored to a lesser degree, generalizing the POMDPs with a vector of rewards [115], although they used an evolutionary algorithm approach to solve them. For example, within this context, simpler lexicographic preference orderings over MOPOMDPs have been considered [100]. It becomes immediately apparent in such a lexicographic POMDP (LPOMDP) that ties in value are exceedingly rare in belief space α-vectors. Similar to CMDPs, constrained POMDPs (CPOMDP) have also been explored to a limited degree, though primarily with respect to algorithms that solve them approximately with point-based techniques [57, 65]. Our work here also includes the partially observable case, generalizing the lexicographic and constrained POMDPs.

Foundational questions regarding lexicographic orderings and constraints in MOMDPs with preference orderings still remain. What would allow a larger space of policies to be considered to fix this common problem with lexicographic orderings? How are lexicographic orderings and constraints related? Is there a common algorithm that can solve both problems? Does a model exist that generalizes both? Can such a model describe novel multi-objective sequential optimization problems? We take steps toward providing answers to these questions.

Our work introduces the *topological MDP (TMDP)* which generalizes both lexicographic orderings and constraints in MDPs. This also describes and extends the line of work on recent lexicographic MDP (LMDP) models [142, 135, 131]. The objective is to maximize each objective following a graph describing their preference relations. As part of this, we define a notion of *slack*—allowable deviations from optimal values—for each objective in the graph. Slack allows more policies to be considered in successive objective functions, while simultaneously capturing the notion of a constraint. Overall, the lexicographic orderings in L(PO)MDPs equate to a linearly ordered *chain graph* and the constraints in C(PO)MDPs equate to a flattened two-level *fan graph.* We show that both are representable in a topological (PO)MDP that applies a general topological ordering over objectives defined by a directed acyclic graph. Preference orderings remain a core technique in multi-criteria decision-making but to the best of our knowledge there have not been any general models that encapsulate these previous models into such a versatile network of constraints. Conditional preference networks (CP-nets) define a graph over variables with a conditional preference table for use in determining the final decision given the assigned decision of each variable in the network [15]. Generalized additive independence (GAI) decomposition networks use a graph to construct a scalarization function, and can employ clever pruning techniques [46]. However, both of these approaches do not solve the challenging multi-objective sequential decision-making problems for MDPs or POMDPs, and do not allow for any flexible slack constraints, as in our proposed TMDP and TPOMDP models.

To ground our analysis in an real-world application of multi-objective decision-making, we consider a semi-autonomous [146] driving scenario. The autonomous vehicle (AV) needs to plan a route that allows both the vehicle and driver to be in control, depending on if the driver is attentive to the road or distracted. The AV is able to drive on main roads. It monitors the state of the human driver and decides if it can transfer control to them to traverse side roads, as needed. The objective is to reach the destination as quickly as possible, while allowing some slack to spend more time autonomous.

Our primary contributions are: (1) a formal definition of the TMDP and TPOMDP models (Sections 5.2 and 5.3); (2) a general algorithm for solving TMDP and TPOMDPs as well as two scalable approximations (Section 5.4); (3) an application of TPOMDPs for semi-autonomous vehicles with multiple route planning objectives (Section 5.5); (4) a theoretical analysis of models and algorithms (Section 5.6); (5) a formal representation of TMDPs and TPOMDPs as a policy network (Section 5.6.2; and (6) an evaluation of a TPOMDP on semi-autonomous vehicle domain using real-world road data of 10 cities (Section 5.7).

5.2 Topological MDP

Lexicographic or ordinal MDPs allow for a sequence of constraints to be defined. Each constraint is dependent on the optimal values of the ancestors. Conversely, constrained MDPs allow for a list of constraints to be applied simultaneously, each bounded by a given constant. The key insight behind a TMDP is that these represent two extremes of a rich landscape of constrained problems. We instead consider any arbitrary directed acyclic graph of constraints.

A **topological Markov decision process (TMDP)** is a sequential decision-making model defined by the tuple $\langle S, A, T, \mathbf{R}, E, \delta \rangle$:

- 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 state transition function such that $T(s,a,s') = Pr(s'|s,a)$ is the probability of successor state s' given action a was performed in state s;
- $\mathbf{R} = [R_1, \ldots, R_k]^T$ is a vector of reward functions for $K = \{1, \ldots, k\}$ such that $R_i : S \times A \to \mathbb{R}$ denotes an immediate reward $R_i(s,a)$ for performing action a in state s;
- $E \subseteq K \times K$ is a finite set of edges over the k rewards forming a directed acyclic graph, with one leaf/sink reward vertex which, without loss of generality, is reward vertex k; and
- $\delta : E \to \mathbb{R}^+$ is a function mapping edges $e = \langle i, j \rangle \in E$ to a non-negative slack constraint $\delta(e) \geq 0$, or also overloading notation by the equivalent $\delta(i,j) \geq 0$.

As in MDPs, a TMDP stationary **policy** is either deterministic $\pi : S \to A$ or stochastic $\pi : S \times A \to [0,1]$. Distinct from MDPs, and rather aligned with CMDPs, TMDPs can require stochastic policies to obtain optimal value (Proposition 10) since they generalized both LMDPs (Proposition 8) and CMDPs (Proposition 9). For notational clarity, we will use deterministic policies; however, stochastic policies follow in the natural way, identical to their use in an MDP.

The **infinite horizon** TMDP with initial state s^0 has a discount factor $\gamma \in [0,1)$. For a policy π, it is based on the expected rewards:

$$\mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t \mathbf{R}(s^t, \pi(s^t) \Big| \pi, s^0\Big] \tag{5.1}$$

with s^t denoting the random variable for the state at time t generated following T. Thus, for a policy π, the **value** $V^\pi : S \to \mathbb{R}$ is the expected reward at state s following:

$$\mathbf{V}^\pi(s) = \mathbf{R}(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') \mathbf{V}^\pi(s') \tag{5.2}$$

Given the DAG E we define two helpful sets. Let $\mathcal{P}_i \subset K$ denote the set of all parents of $i \in K$. Let $\mathcal{A}_i \subset K$ denote the all ancestors of $i \in K$.

Additionally, the graph is assumed to have one leaf/sink reward vertex. If there are multiple vertices, then the set of policies in which the optimal policy is selected for the agent to actually execute is ill-defined. Such graphs, however, are entirely mathematically correct in that the policy sets and values can be computed, but would leave this final important step ambiguous. Thus, we do not consider such graphs in this work. From a policy network perspective, the initial controller is the leaf/sink vertex k, always remaining in control. Hence, any non-ancestor vertices do not impact the behavior of the agent.

5.2.1 Optimality Criterion

In an MDP, we can maximize the value of an *initial state* (i.e., assume s^0 is given) or over *all states* (i.e., make no such assumption). They turn out to be equivalent, as shown by SSPs. In TMDPs, we can also assume the initial state is provided or not in the exact same manner for the objective being optimized. Here, we will assume maximization starting from a known initial state s^0 for consistency across definitions—both the TMDP and the TPOMDP—with the other forms following identically in the natural way from MDPs.

Interestingly, there is an orthogonal property regarding the slack constraints: we can enforce the slack bound only at the *initial state* or, more rigidly, universally across *all states*. While the former is much cleaner and easier to use, the latter is very useful to approximate the former, as we will show in the subsequent sections.

The **initial slack** TMDP with initial state s^0 is the recursively defined objective to find a policy π that maximizes the expected value for reward $i \in K$ following:

$$\begin{aligned} &\text{maximize} && V_i^\pi(s^0) \\ &\text{subject to} && V_w^*(s^0) - V_w^\pi(s^0) \leq \delta(w, v), \quad \forall v \in \mathcal{A}_i \cup \{i\}, \forall w \in \mathcal{P}_v \end{aligned} \tag{5.3}$$

with $V_w^*(s^0)$ denoting the optimal value of ancestor w recursively following this same constrained objective. The astute reader might recognize the policy network constraint edge slack formulation from Equation 3.17. We will show how TMDPs are a special case of the much more general policy network in Section 5.6.2. As such, this can be equivalently written in terms of policy sets:

$$\Pi_i = \{\pi \in \Pi | V_w^*(s^0) - V_w^\pi(s^0) \leq \delta(w,v), \forall v \in \mathcal{A}_i \cup \{i\}, \forall w \in \mathcal{P}_v\} \tag{5.4}$$

with optimal policy π^* constrained to policies within the set Π_i.

We observe that this is also equivalent to the recursive restriction of policy sets following the DAG E. Formally, given any parent vertices $j \in \mathcal{P}_i$ and their policy sets Π_j, the policy set for i is:

$$\Pi_i = \bigcap_{j \in \mathcal{P}_i} \Pi_{ji} \tag{5.5}$$

with edge constraints Π_{ji} from parents j to vertex i following:

$$\Pi_{ji} = \{\pi \in \Pi_j | V_j^*(s^0) - V_j^\pi(s^0) \leq \delta(j,i)\}. \tag{5.6}$$

This follows a similar form as the policy network constraint edges in Equation 3.17. Importantly, this assumes one can compute Π_j and do not need to keep all ancestors of i. We will see that the local action restriction (LAR) can do this by approximating the full policy set Π_i using local sets of actions at each state. But generally speaking, we must at least know all the constrained-optimal values V_j^* for all ancestors of i.

The **universal slack** TMDP with initial state s^0 is the recursively defined objective to find a policy π that maximizes the expected value for reward $i \in K$ following:

$$\begin{aligned} &\text{maximize} && V_i^\pi(s^0) \\ &\text{subject to} && V_w^*(s) - V_w^\pi(s) \leq \delta(w,v), \quad \forall v \in \mathcal{A}_i \cup \{i\}, \forall w \in \mathcal{P}_v, \forall s \in S \end{aligned} \tag{5.7}$$

with $V_w^*(s)$ denoting the optimal value of ancestor w recursively following this same constrained objective. Again, this can be equivalently written in terms of sets of policies:

$$\Pi_i = \{\pi \in \Pi | V_w^*(s) - V_w^\pi(s) \leq \delta(w,v), \forall v \in \mathcal{A}_i \cup \{i\}, \forall w \in \mathcal{P}_v, \forall s \in S\} \tag{5.8}$$

with optimal policy π^* constrained to policies within the set Π_i. We may also equivalently compute Π_i here following Equation 5.5 and edge constraints Π_{ji} from parents j to vertex i:

$$\Pi_{ji} = \{\pi \in \Pi_j | V_j^*(s) - V_j^\pi(s) \leq \delta(j,i), \forall s \in S\} \tag{5.9}$$

similar to Equation 5.6. Technically, in these definitions, if we have an initial state we should only consider the *reachable* states from s^0 in the universal slack TMDP objective definition; however, we defined it here in the most general case.

Hereafter, the final policy set defined at the single leaf vertex is referred to as the *optimal set of policies* Π^*, with optimal policy selected to execute denoted $\pi^* \in \Pi^*$. When necessary, to delineate between the two objectives (e.g., in Section 5.6's Proposition 12), for objective i, we mark the set of initial slack policies as Π_i^I and the set of universal slack policies as Π_i^U. Finally, when not specified otherwise, we let optimality be with respect to the initial slack objective.

5.2.2 Lexicographic MDP

A special case of the TMDP is called a lexicographic MDP [142]. It defines a graph as a chain with an allotted slack at each successive objective. We briefly define it here for the sake of completeness and as a simple useful example of the TMDP.

A **lexicographic MDP (LMDP)** is a sequential decision-making model defined by the tuple $\langle S, A, T, \mathbf{R}, \delta \rangle$:

- S is a finite set of states;
- A is a finite set of actions;
- $T: S \times A \times S \rightarrow [0,1]$ is a state transition function such that $T(s,a,s') = Pr(s'|s,a)$ is the probability of successor state s' given action a was performed in state s;
- $\mathbf{R} = [R_1, \ldots, R_k]^T$ is a vector of reward functions for $K = \{1, \ldots, k\}$ such that $R_i: S \times A \rightarrow \mathbb{R}$ denotes an immediate reward $R_i(s,a)$ for performing action a in state s; and
- $\delta = [\delta_1, \ldots, \delta_{k-1}]^T$ is a vector of non-negative slack constraints $\delta_i \geq 0$.

The policy, value, and objectives are the same as in TMDPs, with each objective (vertex) i iteratively constrained by the $1, \ldots, i-1$ before it. The term *lexicographic* comes from the lexicographic preference ordering over objectives. The precursor *ordinal* multi-objective MDPs are essentially LMDPs with each slack value set to zero. Formally, the objective function follows the universal slack TMDP objective in Equation 5.7 over all states s^0, such that for each reward $i < k$, a policy π must preserve the slack constraint $V_i^*(s) - V_i^\pi(s) \leq \delta_i$ for all $s \in S$.

Interestingly, the solution originally proposed called **lexicographic value iteration (LVI)** uses local action restriction (LAR) to ensure that at each action deviation did not deviate more than a

local one-step slack $\eta_i \geq 0$ from optimal. Formally, LVI defined action sets $A_i(s) \subseteq A$ for all $i < k$, with $A_1(s) = A$. Then, using the action set of i, the successor objective $i+1$ had action set:

$$A_{i+1}(s) = \{a \in A_i(s) | V_i^*(s) - Q_i^*(s,a) \leq \eta_i\}. \tag{5.10}$$

These action sets were maintained for each state, running value iteration k times following the objective order $1,\ldots,k$. This has one main drawback: it can over-restrict the policy space since policy is restricted only locally via $A_i(s)$ instead of over the actual policy space [92]. A generalized version of LVI is provided in Section 5.4 as well as theoretical results regarding LVI in Section 5.6.

5.2.3 Stationarity

As we show in Section 5.6.2, TMDPs are a special case of policy networks. Hence, we will use the same terminology here, leaving aside the comprehensive relatively formal analysis to this upcoming section. Thus, for a vertex $i \in K$, we call it **stationary** if the policy and its policy set are fixed and given. For example, while solving a TMDP we must compute each policy set either implicitly through $V_i^*(s^0)$ or explicitly through Π_i; in both cases we call the reward vertex i stationary once it is solved. As another example, it can simply be given, in which case it is also called stationary. This is foundational to the concept of independence within a TMDP.

Any TMDP offline planning algorithm attempts to exploit these independences to solve all remaining relevant vertices, making them also stationary, before the agent's execution in the environment. To clarify, this is essentially the planning that converts a policy network with non-stationary vertices to one with stationary vertices; the result is a stationary policy for the initial controller vertex. For a TMDP, once the planning algorithm has solved the vertices, then it can be converted to its policy network form, which will have many, if not all, vertices marked as stationary. Of course, online planning and reinforcement learning would allow for non-stationary vertices at the point of execution; however, we will leave this interesting and complex domain to future work.

5.2.4 Graphical Representation

The DAG formed by E enables a straight-forward visualization of the TMDP's topological constraint relationships. Formally, the visualized graph is defined by the tuple $\langle K, E \rangle$: vertices to refer to the reward functions $i \in K$, and edges $e \in E$ refer to the constraints, each of which has a corresponding slack δ_e. We can write the slack δ_e, or just the actual number of the slack, along each edge. Additionally, a vertex's stationarity can be easily rendered by shaded vertex. Recall that this stationarity refers to the *initial*, pre-computation, given assignment of a policy set, either implicitly

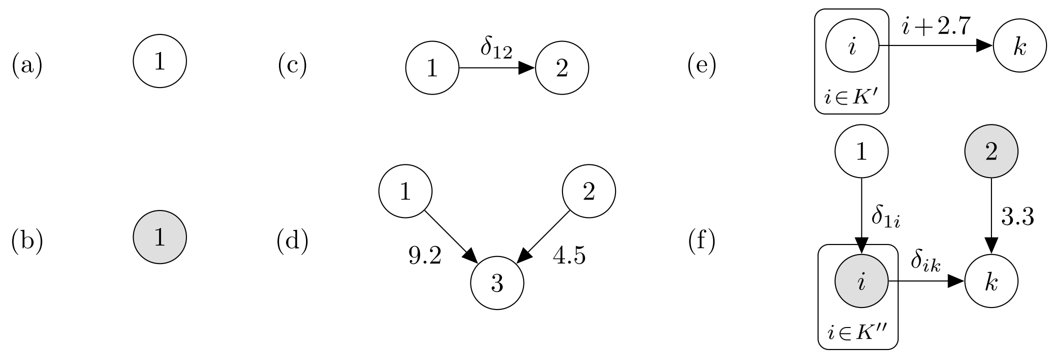


Figure 5.1: Basic examples of the graphical notation that is used to represent a TMDP's topological constraints, for each reward vertex $i \in K$. The examples shown are: (a) an MDP before solving it; (b) an MDP after solving it; (c) a simple two-reward LMDP; (d) a simple two-constraint CMDP; (e) plate notation for a set of $K' = K - \{k\}$ constraints; and (f) mixture of the concepts for a set of $K'' = K - \{1, 2, k\}$ constraints.

by $V_i^*(s^0)$ or explicitly by Π_i. This directly aligns with the notation used by Bayesian networks, as well as policy networks which will be shown generalize TMDPs in the subsequent sections.

Figure 5.1 shows six example TMDPs. We observe in (b) how stationarity is denoted by a filled in vertex. Also, we suggest here how a linear chain of constraints can be constructed in (c) as defined in LMDPs, or even a fan of constraints in (d) as defined in CMDPs. For convenience, edges can denote their slack values directly as shown in (d) or simply state the slack constant name as in (c). As in Bayesian networks and policy networks, in cases when objectives can be visually grouped due to the DAG structure, we may use the plate notation shown in (e) and (f).

When considering the process of solving a TMDP, it quickly becomes apparent that a large input of k objectives might be computationally onerous. Is some way to leverage knowledge of stationary policy sets in order to rapidly compute the final policy, such as in (f)? In Bayesian networks, for example, we can compute the full joint probability quickly if independences among random variables are found, as well as efficiently store the Bayesian network. Can TMDPs benefit from a similar analysis? It turns out that TMDPs have a very similar notion of independence. Since a TMDP follows from a policy network, it shares the same definition of independence as in Section 3.3.6.

5.3 Topological POMDP

As the POMDP is an extension of the MDP to include partial observability, so to can we derive a TMDP with partial observability. In fact, as in POMDPs, a topological POMDP is merely a continuous TMDP with a special structure resulting in PWLC a value function. As such, we will briefly state the TPOMDP for completeness, referring to Section 5.2 and highlighting any key distinctions.

A **topological partially observable Markov decision process (TPOMDP)** is a sequential decision-making model defined by the tuple $\langle S, A, \Omega, T, O, \mathbf{R}, E, \delta \rangle$:

- S is a finite set of states;
- A is a finite set of actions;
- Ω is a finite set of observations;
- $T : S \times A \times S \rightarrow [0,1]$ is a state transition function such that $T(s,a,s') = Pr(s'|s,a)$ is the probability of successor state s' given action a was performed in state s;
- $O : A \times S \times \Omega \rightarrow [0,1]$ is an observation function such that $O(a,s',\omega) = Pr(\omega|a,s')$ is the probability of observing ω given action a was performed resulting in successor s';
- $\mathbf{R} = [R_1, \ldots, R_k]^T$ is a vector of reward functions for $K = \{1, \ldots, k\}$ such that $R_i : S \times A \rightarrow \mathbb{R}$ denotes an immediate reward $R_i(s,a)$ for performing action a in state s;
- $E \subseteq K \times K$ is a finite set of edges over the k rewards forming a directed acyclic graph, with one leaf/sink reward vertex which, without loss of generality, is reward vertex k; and
- $\delta : E \rightarrow \mathbb{R}^+$ is a function mapping edges $e = \langle i,j \rangle \in E$ to a non-negative slack constraint $\delta(e) \geq 0$, or also overloading notation by the equivalent $\delta(i,j) \geq 0$.

The definition of policy is identical to TMDPs, with the same properties such as generally requiring stochastic policies to obtain optimal value. As in a POMDP, the TPOMDP operates over a *belief* $b \in \triangle^{|S|}$ of the world following the POMDP description in Chapter 2. Consequently, they follow the same belief update in Equation 2.13. For notational brevity, we will write equations with deterministic policies $\pi : \triangle^{|S|} \rightarrow A$. They also generalize the LPOMDP and CPOMDP models in the same manner as TMDPs do for the LMDP and CMDP models (Section 5.6).

The **infinite horizon** TPOMDP with initial state s^0 has a discount factor $\gamma \in [0,1)$. For a policy π, it is based on the expected rewards:

$$\mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t \mathbf{R}(b^t, \pi(b^t) \Big| \pi, b^0\Big] \tag{5.11}$$

with b^t denoting the random variable for the belief at time t generated following T and O. For a policy π, the **value** $V^\pi : \triangle^{|S|} \to \mathbb{R}$ is the expected reward at belief b following:

$$\mathbf{V}^\pi(b) = \mathbf{R}(b, \pi(b)) + \gamma \sum_{\omega \in \Omega} Pr(\omega | b, \pi(b)) \mathbf{V}^\pi(b'_{\pi(b)\omega}) \tag{5.12}$$

and $\mathbf{R}(b,a) = \sum_s b(s)\mathbf{R}(s,a)$ and $b'_{\pi(b)\omega}$ following the belief update equation.

As in POMDPs, the infinite horizon TPOMDP is undecidable. However, it shares the same value function for each objective, just in vector form. Thus, we may leverage the PWLC property of a similar **finite horizon** TPOMDP objective to approximate the infinite horizon TPOMDP. Following the same logic as POMDPs and leveraging Equation 2.19, we may use a set of α-vectors Γ_i for each objective $i \in K$, with their collection $\mathbf{\Gamma} = \{[\alpha_1, \ldots, \alpha_k]^T \in \mathbb{R}^k | \forall i \in K, \alpha_i \in \Gamma_i\}$, to represent the value function. The finite horizon Bellman equation for policy π at belief b:

$$\mathbf{V}^\pi(b) = \mathbf{R}(b, \pi(b)) + \gamma \sum_{\omega \in \Omega} \max_{\alpha' \in \mathbf{\Gamma}} \sum_{s \in S} b(s) \sum_{s' \in S} T(s, \pi(b), s') O(\pi(b), s', \omega) \alpha'(s'). \tag{5.13}$$

The graphical representation of a TPOMDP is identical to that of a TMDP, as is the notion of stationarity and independence.

5.3.1 Optimality Criterion

Both notions of optimality follow directly from a TMDP with state space of beliefs using TPOMDP value Equation 5.12. Thus, an **initial slack** TPOMDP objective for initial belief b^0 is the recursively defined objective to find a policy π that maximizes the expected value for reward $i \in K$ following:

$$\begin{aligned} &\text{maximize} \quad V_i^\pi(b^0) \\ &\text{subject to} \quad V_w^*(b^0) - V_w^\pi(b^0) \leq \delta(w,v), \quad \forall v \in \mathcal{A}_i \cup \{i\}, \forall w \in \mathcal{P}_v \end{aligned} \tag{5.14}$$

with $V_w^*(b^0)$ denoting the optimal value of ancestor w recursively following this same constrained objective. Again, this can be equivalently written in terms of policy sets:

$$\Pi_i = \{\pi \in \Pi | V_w^*(b^0) - V_w^\pi(b^0) \leq \delta(w,v), \forall v \in \mathcal{A}_i \cup \{i\}, \forall w \in \mathcal{P}_v\} \tag{5.15}$$

with optimal policy π^* constrained to policies within the set Π_i. The **universal slack** TPOMDP objective follows naturally from the TMDP in the same manner.

5.3.2 Lexicographic POMDP

As the TMDP can define a chain of slack constraints to produce the LMDP, the LPOMDP with a chain of constraints produces a lexicographic POMDP. Formally, a **lexicographic MDP (LMDP)** is a sequential decision-making model defined by the tuple $\langle S, A, \Omega, T, O, \mathbf{R}, \delta \rangle$:

- S is a finite set of states;
- A is a finite set of actions;
- Ω is a finite set of observations;
- $T: S \times A \times S \rightarrow [0,1]$ is a state transition function such that $T(s,a,s') = Pr(s'|s,a)$ is the probability of successor state s' given action a was performed in state s;
- $O: A \times S \times \Omega \rightarrow [0,1]$ is an observation function such that $O(a,s',\omega) = Pr(\omega|a,s')$ is the probability of observing ω given action a was performed resulting in successor s';
- $\mathbf{R} = [R_1, \ldots, R_k]^T$ is a vector of reward functions for $K = \{1, \ldots, k\}$ such that $R_i: S \times A \rightarrow \mathbb{R}$ denotes an immediate reward $R_i(s,a)$ for performing action a in state s; and
- $\delta = [\delta_1, \ldots, \delta_{k-1}]^T$ is a vector of non-negative slack constraints $\delta_i \geq 0$.

The policy, value, and objectives are the same as in TPOMDPs, with each objective (vertex) i iteratively constrained by the $1, \ldots, i-1$ before it. Formally, the objective function follows a universal slack TPOMDP objective for initial belief b^0, such that for each reward $i < k$, a policy π must preserve the slack constraint $V_i^*(b) - V_i^\pi(b) \leq \delta_i$ for all $b \in \triangle^{|S|}$.

The **lexicographic value iteration (LVI)** for LPOMDPs also used local action restriction (LAR) at each belief explored. Thus, each action deviation did not deviate more than a **local one-step slack** $\eta_i \geq 0$ from optimal. Formally, LVI defined action sets $A_i(b) \subseteq A$ for all $i < k$, with $A_1(b) = A$. Again, using the action set of i, the successor objective $i+1$ had action set:

$$A_{i+1}(b) = \{a \in A_i(b) | V_i^*(b) - Q_i^*(b,a) \leq \eta_i\}. \tag{5.16}$$

These action sets were maintained for each belief explored $b \in B \subseteq \mathcal{R}(b^0)$, running value iteration k times following the objective order $1, \ldots, k$. With this equation, we can also consider any explored subset of beliefs B such as those explored by PBVI. This variant is called **lexicographic point-based value iteration (LPBVI)**. A generalized version of LVI, and its approximation LPBVI, is provided in Section 5.4 as well as theoretical results regarding LVI and LPBVI in Section 5.6.

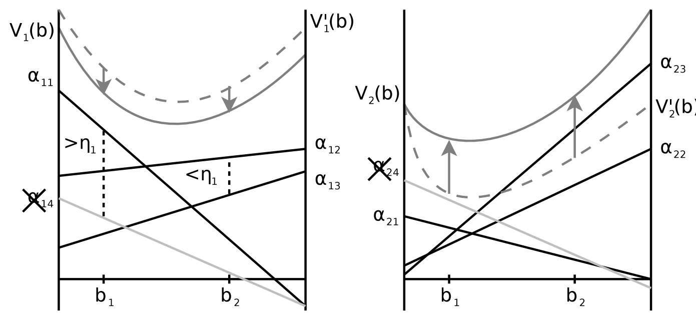


Figure 5.2: Examples of action restriction and slack affordance. The true value functions (gray dotted lines) are approximated by the α-vectors from below (black lines). The first value function V_1 (left) is reduced up to the slack δ_1 to afford the second value function V_2 (right) to greatly increase its value. This example illustrates LVI's operation.

Example A two objective LPOMDP, or more generally TPOMDP, is shown in Figure 5.2. It depicts two belief points b_1 and b_2; two value functions V_1 and V_2; and four α-vectors for each value function. The unknown true (infinite horizon) value of $V_1(b)$ and $V_2(b)$ are shown, as well as the original $V_1'(b)$ and $V_2'(b)$ which would have been selected if slack was not introduced. First we examine V_1. For belief b_1, we observe two α-vectors: α_{11} and α_{14}. The difference between $b_1 \cdot \alpha_{11}$ and $b_1 \cdot \alpha_{14}$ is greater than the allowed slack η_1. Therefore, α_{14} is thrown out of Γ_1^t. Conversely, for belief b_2, the $b_2 \cdot \alpha_{12}$ and $b_2 \cdot \alpha_{13}$ are within η_i so Γ_1^t contains both. These two sets of α-vectors define the actions available to V_2. We make three observations about V_2. First, α_{24} corresponds to the *potential* α-vector that *would* have been available had we not removed α_{14}'s action. Its actual value was higher, but the first value function restricted the set of available actions for the second value function. Second, α_{13}'s action inclusion for the actions available at b_2 in V_2 enabled it to obtain a *higher* value (with α_{23}) than it would have if we had only allowed the maximal action to be taken (with α_{22}). Third, the infinite horizon values of $V_1'(b)$ decreased slightly to $V_1(b)$ because we allowed for slack in order to *greatly improve* the second value function from $V_2'(b)$ to $V_2(b)$.

5.4 Algorithm

We begin by describing the general approach to solve a TMDP or TPOMDP. Essentially, we simply follow the DAG's topological order, solving that vertex until we arrive at the leaf. As a consequence of the duality of the optimality criteria, there are two types of algorithm. Then we discuss infeasibility in TMDP and TPOMDP resulting from infeasible constraints. Once these concepts are established, we propose the general algorithm for TMDPs and TPOMDPs as well as two approximations following the two types of algorithm.

5.4.1 Duality in Algorithms

The general algorithm is simple given TMDP constraints form a directed acyclic graph (DAG). For each objective, we can follow the topological order of the DAG and solve the optimization problem corresponding to Equation 5.3. This optimization problem is a quadratically constrained linear program (QCLP) and can be solved using non-linear programming (NLP) [3]. However, this is challenging for TPOMDPs who must explore an exponential number of beliefs. In this case, a controller family policy form such as a simple FSC can be used to solve it approximately. Alternatively, we can approximate the policy set restrictions using a more general form of LVI with LAR at states or beliefs.

Interestingly, these two approaches correspond to the equivalent definitions of the optimality criteria in Equations 5.3 and 5.4 for the initial slack TMDP, or Equations 5.7 and 5.8 for the universal slack TMDP. In the first case, we compute the values $V_i^*(s^0)$ to *implicitly* represent the policy sets. In the second case, we compute the policy sets Π_i—or approximations of them—*explicitly*. Both implicit and explicit algorithms are described in this section. The experiments on autonomous vehicles focus on an explicit approximate algorithm based on LVI.

Implicit algorithms for TMDPs require solving for the value $V_i^*(s^0)$. Mathematical programming using NLP-based forms are the most natural way to do this because the slack constraints use the values $V_i^*(s^0)$ to ensure they are satisfied. The benefits are: (1) the optimal values are actually computed so it can return the correct optimal policy; and (2) the NLP form can be solved by very efficient off-the-shelf solvers that are easy to use. The drawbacks are: (1) for any vertex i, we must have constraints for *all* its ancestors $\mathcal{A}_i$; and (2) for large state spaces such as in TPOMDPs this is impossible to write as a solvable NLP.

Explicit algorithms for TMDPs require solving for the policy sets Π_i. LVI-based approaches are the most natural way to do this because storing the action sets $A_i(s)$ are surrogates for part of the policy set Π_i. The benefits are: (1) it can benefit from the most popular scalable Bellman optimality equation algorithms such as PBVI; and (2) it is typically much faster to use VI-based algorithms custom-made for MDPs rather than solving an NLP using generic NLP algorithms. The drawbacks are: (1) it typically over-restricts the space of policies at each step, yielding TMDP policies that are not optimal; and (2) more memory is required to store the set of actions $A_i(s)$.

5.4.2 Infeasibility

As with CMDPs, sometimes the constraints are impossible to satisfy. This creates *infeasible solutions* in the mathematical programming problem. Formally, we call a TMDP or TPOMDP

Algorithm 2 An algorithm to compute the optimal policy of a TMDP.

```
Require: $\langle S, A, T, \mathbf{R}, E, \delta \rangle$: The TMDP to solve.
1: procedure SolveTMDP($K$, $E$, $\mathbf{x}$, $k$)
2:     $\langle \pi^*, \mathbf{V}^*(s^0), \mathbf{\Pi} \rangle \leftarrow \langle \pi^0, \mathbf{0}^k, \{\} \rangle$
3:     for $i \leftarrow x_1, \ldots, x_k$ do
4:         $\mathcal{A}_i \leftarrow \{ j \in K | \exists p \in$ Path($K$, $E$, $j$, $i$) $\}$
5:         $\langle \pi_i^*, V_i^*(s^0), \Pi_i \rangle \leftarrow$ SolveRelativeCMDP($S$, $A$, $T$, $\mathbf{R}$, $i$, $\mathcal{A}_i$, $\mathbf{V}^*(s^0)$, $\mathbf{\Pi}^*$)
6:         $\langle \pi^*, \mathbf{V}^*(s^0), \mathbf{\Pi} \rangle \leftarrow \langle \pi_i^*, \mathbf{V}^*(s^0) + V_i^*(s^0) e_i, \mathbf{\Pi} \cup \{\Pi_i\} \rangle$
7:     return $\pi^*$
8: $\mathbf{x} \leftarrow$ ComputeReversePostOrderDFS($K$, $E$, $k$)
9: return SolveTMDP($K$, $E$, $\mathbf{x}$, $k$)
```

infeasible if there exists an $i \in K$ and a slack constraint $\delta(j,i)$ such that the optimization problem Equation 5.3 (or Equation 5.7) is infeasible.

From the perspective of the policy sets, this would create an empty intersection of the objectives' parents sets. Infeasibility can be checked, as shown in Algorithm 2. It can be checked in both the implicit form, which computes values $V_i^*(s^0)$—approximately if it uses an NLP with a controller family policy like an FSC—and the explicit form, which maintains the policy sets Π_i—approximately if it uses LAR to store only the action sets for each state.

Technically our definition of slack being satisfied, $V_j^*(s^0) - V_j^\pi(s^0) \leq \delta(j,i)$ for $\langle j,i \rangle \in E$, is not as general as a CMDP's $-V_i^\pi(s^0) \leq c_i$. There may not exist *any* policy that satisfies the constraint c_i if the assignment is too harsh. Therefore in a TMDP, only infeasibility with respect to the intersection of constraints is possible, omitting infeasibility with respect to a single constraint in isolate. We determined that this form of infeasibility was unnecessary to include because: (1) the use of slack is much more intuitive universally, especially to a user [142, 135], whereas the CMDP constraints are not as universal; (2) a system designer would not consider an impossible agent constraint; and (3) the defined the slack constraint cleanly unifies many other models such as LMDPs and MODIA. That said, it is a trivial change to use the CMDP constraint form instead; the TMDP properties hold in the same manner.

5.4.3 Optimal Algorithm

Algorithm 2 defines the general algorithm to solve a TMDP. The TPOMDP can be solved in exactly the same manner. On Line 2, both $\pi^0 \in \Pi$ and $\mathbf{0}^k = [0, \ldots, 0]^T \in \mathbb{R}^k$ are initial assignments for the policy and value, respectively. On Line 6, we simply overwrite the optimal policy, store the optimal value in the correct vector location using standard basis e_i, and store the policy set used to constrain any children. Path(K, E, j, i) returns the set of paths between j and i on $\langle K, E \rangle$. SolveRelativeCMDP(S, A, T, $\mathbf{R}$, i, $\mathcal{A}_i$, $\mathbf{V}^*$, $\mathbf{\Pi}^*$) solves the relative CMDP (e.g., via Equation 5.3

Function: $\textsc{ApproximatelySolveRelativeCMDP}(\cdot)$
Given: S, A, T, R, $i \in K$, $\mathcal{A}_i$, $x^0 \in X$, $s^0 \in S$, $V_v^*(s^0)$, $\forall v \in \mathcal{A}_i$
Variables: $\psi(x,a)$, $\forall x,a$ $\quad \eta(x,a,x')$, $\forall x,a,x'$ $\quad \hat{V}_j(x,s)$, $\forall x,s$, $\forall j \in \mathcal{A}_i \cup \{i\}$
Maximize: $\hat{V}_i(x^0,s^0)$
Bellman Constraints:
$\hat{V}_v(x,s) = \sum_{a \in A} \psi(x,a) \Big[R_v(s,a) + \gamma \sum_{s' \in S} T(s,a,s') \sum_{x' \in X} \eta(x,a,x') \hat{V}_v(x',s') \Big], \quad \forall v \in \mathcal{A}_i \cup \{i\}$
Slack Constraints:
$\hat{V}_w^*(s^0) - \hat{V}_w^\pi(x^0,s^0) \leq \delta(w,v), \quad \forall v \in \mathcal{A}_i \cup \{i\}, \forall w \in \mathcal{P}_v$
Probability Constraints:
$\psi(x,a) \geq 0$, $\forall x,a$ $\quad \sum_{a \in A} \psi(x,a) = 1$, $\forall x$
$\eta(x,a,x') \geq 0$, $\forall x,a,x'$ $\quad \sum_{x' \in X} \eta(x,a,x') = 1$, $\forall x,a$

Table 5.1: An NLP using an FSC to approximate $\textsc{SolveRelativeCMDP}(\cdot)$ in Algorithm 2.

or 5.7). It returns an optimal policy π_i^*, the value of this optimal policy V_i^*, and the set of policies Π_i for this vertex. The term *relative* refers to the relative position in the DAG's topological order. An approximate NLP algorithm for SolveRelativeCMDP(i)s provided in the proceeding subsection. Moreover, an approximate Bellman equation-based algorithm for SolveTMDP(i)tself is provided as well. The astute reader might recognize that this algorithm is a special case of the policy network's Algorithm 1 in Chapter 3 as the TMDP is a special case of a policy network.

5.4.4 Finite State Controller Algorithm

Ideally for an implicit algorithm, we employ an NLP solver such as SNOPT [44] for the mathematical optimization problem in Equation 5.3 or 5.7 for Algorithm 2's SolveRelativeCMDP(.) The problem is that this is intractable for these generic solvers, especially for a TPOMDP, in comparison to the specialized algorithms designed to solve MDPs and POMDPs. Thankfully, as we saw in Chapter 4, we can leverage a finite state controller (FSC) as a policy instead to more compactly represent a policy for use in NLP solvers.

The general NLP solution that uses an FSC is presented in Table 5.1 for a TMDP. Specifically, we use an FSC $\langle X, \psi, \eta \rangle$ shown, with action selector $\psi : X \times A \rightarrow [0,1]$ and successor selector $\eta : X \times A \times X \rightarrow [0,1]$; however, we could easily use any policy in the controller family as described in Chapter 4. In the solution, notice that the constraints of ancestors $\mathcal{A}_i$ are satisfied, provided we iteratively store all ancestor values in $\mathbf{V}^*$ as part of Algorithm 2's inner loop over the topological order $\mathbf{x}$ of DAG E. Also notice that the. This shows the two drawbacks to implicit algorithms for TMDPs.

Importantly, a third drawback emerges with *approximate* implicit algorithms such as this: the optimal $V_i^*(s^0)$ is not actually computed, instead we compute an approximate $\hat{V}_i^*(s^0)$ (e.g., using

Algorithm 3 Local action restriction to approximate SolveTMDP($\cdot$) in Algorithm 2.

```
Require: $\langle S, A, T, \mathbf{R}, E, \delta \rangle$: The TMDP to approximately solve.
Require: $\eta$: The local slack, used at each state, specific for each $e \in E$.
 1: procedure ApproximatelySolveTMDP($K$, $E$, $\mathbf{x}$, $k$, $\eta$)
 2:     $\langle \pi^*, \hat{\mathbf{V}}^*, \mathbf{A} \rangle \leftarrow \langle \pi^0, \{\}, \{\} \rangle$
 3:     for $i \leftarrow x_1, \ldots, x_k$ do
 4:         $\mathcal{P}_i \leftarrow \{ j \in K | \exists \langle j, i \rangle \in E \}$
 5:         $A_i(s) \leftarrow A \cap \Big( \bigcap_{j \in \mathcal{P}_i} \Big\{ a \in A_j(s) \Big| \hat{V}_j^*(s) - \hat{Q}_j^*(s,a) \leq \eta(j,i) \Big\} \Big)$, $\forall s \in S$
 6:         $\langle \pi_i^*, \hat{V}_i^* \rangle \leftarrow$ SolveMDP($S$, $A_i$, $T$, $R_i$)
 7:         $\langle \pi^*, \hat{\mathbf{V}}^*, \mathbf{A} \rangle \leftarrow \langle \pi_i^*, \hat{\mathbf{V}}^* \cup \{\hat{V}_i^*\}, \mathbf{A} \cup \{A_i\} \rangle$
 8:     return $\pi^*$
 9: $\mathbf{x} \leftarrow$ ComputeReversePostOrderDFS($K$, $E$, $k$, $\eta$)
10: return ApproximatelySolveTMDP($K$, $E$, $\mathbf{x}$, $k$, $\eta$)
```

an FSC policy form). Consequently the constraints do not actually represent slack from the optimal policy's value $V_j^*(s^0) - V_j^\pi(s^0) \leq \delta(j,i)$, only deviation from an approximate policy's value $\hat{V}_j^*(s^0) - \hat{V}_j^\pi(s^0) \leq \delta(j,i)$. Hence, even if it is merely an ϵ error from optimal, with $|V_i^*(s^0) - \hat{V}_i^*(s^0)| = \epsilon > 0$, then this error can still result in arbitrarily poor policies for descendent objectives. Moreover, even just the computation of the value of a policy could also be off by some ϵ with $|V_i^\pi(s^0) - \hat{V}_i^\pi(s^0)| = \epsilon > 0$, producing similar erroneous results. This can even propagate over each objective i, leading to unbounded violations of the slack when compared with the *true* optimal value. In summary, implicit algorithms compute $V_i^*(s^0)$ and approximations cannot exactly compute this, instead computing $\hat{V}_i^*(s^0)$. This results in a slack guarantee only with respect to the approximate value $\hat{V}_i^*(s^0)$, not the true value $V_i^*(s^0)$.

5.4.5 Local Action Restriction Algorithm

Ideally for an explicit algorithm, we store the entire Π_i in Equations 5.4 or 5.8, each time computing the intersection of the constraint edges Π_{ji} for each $j \in \mathcal{P}_i$ using Equation 5.5 with either Equation 5.6 or 5.9. The problem is that this is intractable to store, again this is essentially impossible for a TPOMDP. As in the case of implicit algorithms, we can also leverage approximate algorithms based on the Bellman optimality equations by iteratively restricting actions at states.

Specifically, we generalize LVI for the TMDP and TPOMDP by recording a set of action sets $\mathbf{A}$, with each $A_i \in \mathbf{A}$ denoting a set of available actions $A_i(s) \subseteq A$ at each state $s \in S$ for objective $i \in K$. Formally, let $A_i : S \rightarrow \mathcal{P}(A)$ denote this mapping from state to action set. All nodes begin with $A_i(s) = A$ for all i and s. At each step of the original Algorithm 2, we can simplify Lines 4-6 by only considering the intersection of parent action sets and instead solving a much simpler MDP rather than a much more complex CMDP. As in LVI, we only keep the actions within a *local one-step slack*

$\eta : E \rightarrow \mathbb{R}^+$ of the value. This allows us to leverage a standard Bellman optimality equation-based algorithm for Algorithm 2's Line 5. Therefore, instead of storing all combinations of actions, we are able to: (1) store a quadratic number of policies in the form of actions $O(|K||A||S|)$, rather than an exponential number of policies $O(|K||A|^{|S|})$, one for each $i \in K$; (2) leverage the formal definition of MDPs that allows for action sets at each state; (3) perform an intersection operation that produces exactly one unique successor set of actions at each iteration; and (4) ignore all unnecessary ancestors to focus on intersecting parent actions within slack. Algorithm 3 describes this modified approximate process formally.

Importantly, a third drawback emerges with *approximate* explicit algorithms too, similar to the approximate implicit algorithm described above: the optimal set of policies within slack Π_i is not actually computed, instead we compute an approximate $\hat{\Pi}_i$. For example, using local action sets $A_i(s)$ for all states s, $\hat{\Pi}_i = \{\pi \in \Pi | \pi(s) \in A_i(s), \forall s \in S\}$. Consequently, the constraints do not actually represent *all* the policies that are within slack from optimal. Instead, as we will show in the next section, with the proper assignment of local one-step slack $\eta(j,i) = (1-\gamma)\delta(j,i)$ the approximate set of policies is a subset of the true set $\hat{\Pi}_i \subseteq \Pi_i$. Thus for a child vertex, this can produce arbitrarily poor policies and values when compared with the true optimal values obtainable if the full policy set had been used. Furthermore, this error can also propagate to descendants the DAG, leading to unbounded violations of the slack when compared with the *true* optimal value; this is a similar effect to the approximate method above. In summary, explicit algorithms compute Π_i and approximations cannot exactly compute this, instead computing $\hat{\Pi}_i$. This results in a slack guarantee only with respect to the approximate value $\hat{V}_i^*(s^0)$, not the true value $V_i^*(s^0)$.

5.5 Application to Autonomous Vehicles

In many real-world applications, we must more than one concern, such as cost, quality, or time. Within the context of autonomous vehicles (AVs), we consider the problem of route planning. The primary objective is to reaching a goal address as quickly as possible, weighing the speed limits and stochasticity of traffic lights and any traffic. Additionally, since this is an AV, we are willing to take longer routes provided more time is spent autonomous on autonomy-capable roads. We formally define this focused problem as an LMDP, allowing for additional AV-related objectives to be easily added as a TMDP.

5.5.1 Problem Definition

The **multi-objective route planning problem** is defined by a strongly connected weighted directed graph $\langle V, E, w, \delta, \sigma, \psi \rangle$. V is a set of vertices forming intersections. $E \subseteq V \times V$ is a set edges between intersections, thus defining roads. However, only some roads $E_c \subseteq E$ are deemed officially **autonomy-capable**—main roads with a speed limit greater than or equal to 30—with non-autonomy-capable roads discouraged. $w : E \rightarrow \mathbb{R}^+$ is the cost in time (in seconds) for traversing each road. The probability of traversing a road is assumed to be given as part of this. Generally, it should be proportional to any historical traffic data. For simplicity of discussion here, we assume this is a known $\epsilon > 0$ additional cost in seconds applied to each $w(e)$; this can be made vertex and edge dependent, if desired, for use with historical traffic data. We can assume initial and goal vertices are provided as part of V, denoted $v^0 \in V$ and $v^g \in V$, respectively.

There are two objectives: *time* and *autonomy*. Primarily, we seek to minimize the cost to reach the goal location. However, we are willing to sacrifice the faster routes up to a slack of $\delta \in \mathbb{R}^+$ seconds in order to improve the amount of time spent autonomously along the route.

This is a simple application of semi-autonomy [26, 146, 132] in which control of the system is transferred between a human λ and an agent ν, denoted $I = \{\lambda, \nu\}$. In our semi-autonomous driving scenario, the main focus will be on transfer of control decisions based on the driver's level of *fatigue*; formally, we denote attentive α and tired τ as $X = \{\alpha, \tau\}$. We assume the human becomes more fatigued over time, modelled by a probability $\psi = 0.1$ of becoming fatigued at each time step. The multi-objective model we use is motivated by an extensive body of engineering and psychological research on monitoring driver fatigue and risk perception [58, 97]. We will assume both a perfect monitoring of the human—as an LMDP—as well as imperfect monitoring via noisy sensors—as an LPOMDP—instead assuming a known success rate of $\sigma = 0.75$. Chapter 6 provides a comprehensive study of semi-autonomous systems.

5.5.2 Multi-Objective AV Route Planning Formulation

This route planning problem is an LMDP $\langle S, A, T, \mathbf{R}, \delta \rangle$ with $k = 2$ rewards. $S = V \times V \times X \times I$ describes the decisions made at the intersection of roads, the level of fatigue of the driver, and which agent is controlling the vehicle. $A = D \times I$ describe the possible directions (roads) to take at a vertex (intersection) (e.g., $D = \{\leftarrow, \uparrow, \rightarrow\}$) and which agent should be in control next. Let $\theta : V \times V \times D \rightarrow V$ map each previous intersection, current intersection, and a direction taken to a successor vertex in the graph following E, including self-loops for invalid directions.

The state transition T must capture the route planning graph as well as the driver's fatigue changing over time. Formally, T is defined by Equation 5.20 with three distinct state transition functions: T_v, T_x, and T_i, one for each state factor. Each considers a state $s = \langle p, v, x, i \rangle$, an action $a = \langle d, j \rangle$, and a successor state $s' = \langle p', v', x', i' \rangle$. First, T_v describes the validity of an action following the route planning graph:

$$T_v(p, v, d, p', v') = \begin{cases} 1, & \text{if } v' = \theta(p, v, d) \wedge p' = v \\ 0, & \text{otherwise} \end{cases}. \tag{5.17}$$

Next, T_x describes the driver's fatigue level potentially increasing:

$$T_x(x, x') = \begin{cases} 1, & \text{if } x = \tau \wedge x' = \tau \\ \psi, & \text{if } x = \alpha \wedge x' = \tau \\ (1 - \psi), & \text{if } x = \alpha \wedge x' = \alpha \\ 0, & \text{otherwise} \end{cases}. \tag{5.18}$$

Lastly, T_i describes the effect to transferring control between agents:

$$T_i(i, j, i') = \begin{cases} 1, & \text{if } i' = j \\ 0, & \text{otherwise} \end{cases}. \tag{5.19}$$

We combine Equations 5.17, 5.18, and 5.19 to get our state transition:

$$T(s, a, s') = T_v(p, v, d, p', v') T_x(x, x') T_i(i, j, i'). \tag{5.20}$$

Chapter 6 formalizes these notions into a general framework, allowing for probabilistically rich interactions among agents and with the environment.

The first reward function R_1 measures the time spent during travel, including the expected time to wait an each intersection:

$$R_1(s, a) = \begin{cases} -w(\langle v, \theta(p, v, d) \rangle), & \text{if } v \neq v^g \\ 0, & \text{otherwise} \end{cases}. \tag{5.21}$$

The second reward function R_2 must also consider if the driver is fatigued and if the road is autonomy-capable:

$$R_2(s,a)=\begin{cases} -w(\langle v,\theta(p,v,d)\rangle), & \text{if } v\neq v^g \wedge i=\lambda \wedge (x=\tau \vee \langle v,\theta(p,v,d)\rangle \in E_c) \\ -\epsilon, & \text{if } v\neq v^g \wedge \neg(i=\lambda \wedge (x=\tau \vee \langle v,\theta(p,v,d)\rangle \in E_c)) \\ 0, & \text{otherwise} \end{cases}, \tag{5.22}$$

which essentially measures a cost whenever autonomy is not being used properly.

Additionally, we can consider the LPOMDP version of the problem $\langle S,A,\Omega,T,O,\mathbf{R},\delta\rangle$ with all set and function assignments as above. $\Omega=X$ denotes the noisy observations of the driver's level of fatigue. Thus, O follows the model of the sensor noisy from an eye tracker or other monitoring device [58, 97]:

$$O(a,s',\omega)=\begin{cases} \sigma, & \text{if } \omega=x' \\ (1-\sigma), & \text{if } \omega\neq x' \end{cases}. \tag{5.23}$$

5.6 Theoretical Analysis

The TMDP has a very flexible structure for relating constrained objectives to one another. Consequently, we begin by formally generalizing the LMDP and CMDP with the TMDP in Propositions 8 and 9—in fact, they represent two extreme cases of a TMDP: chain- and fan-structured graphs, respectively.

Proposition 8. *Topological MDPs (TDPs) generalize lexicographic MDPs (LMDPs).*

Proof. We must construct an LMDP using a TMDP. Let the TMDP's S, A, T, and $\mathbf{R}$ be equivalent to the LMDP. Given k objectives, let $E=\{\langle i,i+1\rangle,\forall 1\leq i\leq k-1\}$. Let $\delta(i,i+1)$ be any slack assignment from the LMDP (e.g., 0 or some $\delta_i\geq 0$). We observe that we have reconstructed the LMDP's objective (Equation 2.47) with the initial slack TMDP objective (Equation 5.3). □

Proposition 9. *Topological MDPs (TMDPs) generalize constrained MDPs (CMDPs).*

Proof. We must construct a CMDP using a TMDP. Let the TMDP's S, A, and T be equivalent to the CMDP. Let $\mathbf{R}$ be equivalent to the CMDP, except let R_1, …, R_{k-1} be the cost functions in CMDPs instead of rewards, with $R_i(s,a)=-C_i(s,a)$ for each such i. Since expectation is a linear operator, the CMDP's constraint objective values denoted $\bar{V}_i^\pi(s^0)$ are exact negations of the values computed by the TMDP denoted $V_i^\pi(s^0)$ such that $V_i^\pi(s^0)=-\bar{V}_i^\pi(s^0)$. Let the TMDP's R_k be identical to the CMDP's maximizing objective reward. Let $E=\{\langle i,k\rangle,\forall 1\leq i\leq k-1\}$. Given the CMDP's constraints c_i, let $\delta(i,k)=c_i-V_i^*(s^0)$. This slack definition results in the TMDP parental constraints of the form: $V_i^*(s^0)-V_i^\pi(s^0)\leq c_i-V_i^*(s^0)$. This yields $-V_i^\pi(s^0)\leq c_i$, which equates to the original CMDP constraints $\bar{V}_i^\pi(s^0)\leq c_i$. □

An important corollary of the CMDP mathematical optimization problem being representable by the TMDP, as shown in Proposition 10, is that stochastic policies may be required for some TMDP. This also holds for TPOMDPs for the same reason.

Proposition 10. *There exist TMDPs that require stochastic policies to obtain their maximal value.*

Proof. By Proposition 9, a CMDP mathematical optimization problem can be represented as a TMDP; the value equation is the same in both, whether deterministic or stochastic. (In fact, each iteration of objectives is just a CMDP.) In some cases, CMDPs can require stochastic policies to obtain their maximally value [2]. Thus, there exist TMDPs which require stochastic policies to obtain their maximal value. □

Similarly, we also can ensure that an optimal stationary policy exists by leveraging the properties of CMDP. This is shown in Proposition 11.

Proposition 11. *There exists an optimal stationary policy for any TMDP.*

Proof. For each objective $i \in K$, following a sequence of the topological order $\mathbf{x} = \langle x_1, \ldots, x_k \rangle$ of DAG E, we have a CMDP defined by Equation 5.3 (or Equation 5.7). By induction on i, we show that TMDPs admit an optimal stationary policy.

Base Case: The root nodes of E are first in the topological order. They are MDPs which have optimal stationary policies [8].

Induction Step: Assume true for all ancestors $\mathcal{A}_i$ of i, which is equivalent to assuming true following the sequence $\mathbf{x}$ of the topological ordering. An optimal stationary policy produces an optimal value for all ancestors. These form constraints in the objectives Equation 5.3 (or Equation 5.7). To solve for objective i, we have a CMDP, which have an optimal stationary policy [2]. This policy may need to be stochastic, as in Proposition 10.

Thus by induction, we have shown that the k-th objective admits an optimal stationary policy. Therefore, there exists an optimal stationary policy for any TMDP. □

Next, we focus on the relation of policies. Due to the definition of initial and universal slack, a straight-forward property emerges: comparing the optimization of any objective i, the set of universal slack policies is more restricted than the set of initial slack policies. Importantly, this does not mean that the final policy sets Π_k^I and Π_k^U necessarily have any relation for $k > 2$ because they can each iteratively restrict in divergent ways. Consequently, Proposition 12 proves that given the *same ancestor policy sets*, the application of Equation 5.7 constraints the policy space more than

Equation 5.3. However, this relation is a useful property to understand how Algorithm 3, and thus LVI for L(PO)MDPs, operates for the universal and initial slack objectives.

Proposition 12. *For any TMDP and an objective $i \in K$, given the stationary policy sets Π_v of all i's ancestors $v \in \mathcal{A}_i$, the set of policies for the initial slack objective Π_i^I and the universal slack objective Π_i^U are related by $\Pi_i^U \subseteq \Pi_i^I$.*

Proof. By definition of initial slack in Equation 5.3, there are fewer constraints in the optimization problem than the definition of universal slack in Equation 5.7. Therefore, the set of policies satisfying universal slack Π_i^U must be a subset of (or equal to) the policies of initial slack Π_i^I. Equivalently, we can simply observe the policy sets in Equation 5.4 versus Equation 5.8. At each objective i, universal slack contains a weak subset of the policies that satisfy the initial slack. Therefore, $\Pi_i^U \subseteq \Pi_i^I$. □

We can convert any TPOMDP to an equivalent belief TMDP using the same mapping as for POMDPs and belief MDPs [101, 63]. Consequently, these properties hold for TPOMDPs as well, following the same logic.

Corollary 1. *Topological POMDPs (TPOMDPs) generalize lexicographic POMDPs (LPOMDPs).*

Corollary 2. *Topological POMDPs (TPOMDPs) generalize constrained POMDPs (CPOMDPs).*

Corollary 3. *There exist TPOMDPs that require stochastic policies to obtain their maximal value.*

Corollary 4. *For any TPOMDP and an objective $i \in K$, given the stationary policy sets Π_v of all i's ancestors $v \in \mathcal{A}_i$, the set of policies for the initial slack objective Π_i^I and the universal slack objective Π_i^U are related by $\Pi_i^U \subseteq \Pi_i^I$.*

Corollary 5. *There exists an optimal stationary policy for any TPOMDP.*

5.6.1 Analysis of Algorithms

Now we turn our attention to Algorithms 2 and 3. Recall that the latter algorithm also encompasses the more specific lexicographic value iteration (LVI) for L(PO)MDPs. First, however, we consider the optimal algorithm in Proposition 13, proving that it indeed returns the optimal policy following either of the optimality criteria.

Proposition 13. *Algorithm 2 returns the optimal policy for the initial (or universal) slack objective.*

Proof. On Line 8, we produce a sequence $\mathbf{x}$ that preserves the topological order of DAG E. On Line 9, and subsequently Lines 3-6, this sequence is followed for each objective i. Line 4 defines the ancestors of i by definition. Line 5 solves the relative CMDP following the optimality criterion, recursively defined for objective i, in Equation 5.3 (or Equation 5.7). Line 6 simply stores the results. Finally, the last updated policy $\pi^* = \pi_k^*$, which is the definition of an optimal policy for a TMDP. Therefore, Algorithm 2 returns the optimal policy for the initial (or universal) slack objective. □

With optimality of Algorithm 2 established, we can evaluate how the approximate local action restriction (LAR) from Algorithm 3 performs.

In practice, the TMDP and TPOMDP operate differently for LAR because they are based on value iteration (VI) and point-based VI (PBVI), respectively. As such, we will provide separate proofs for each, first focusing here on LAR for TMDPs. Proposition 14 then proves that there is a unique fixed point in the spaces of value functions $\hat{\mathbf{V}}^*$ when following Algorithm 3. This is important to ensure LAR algorithms actually converge to a collection of values.

Proposition 14. *Algorithm 3's local action restriction (LAR) converges to a unique fixed point* $\hat{V}_i^*$ *for all* $i \in K$*, if* $|A_i(s)| > 1$ *for all* $i \in K$ *and* $s \in S$*.*

Proof. For each objective $i \in K$, following a sequence of the topological order $\mathbf{x} = \langle x_1, \ldots, x_k \rangle$ of DAG E (Algorithm 3, Line 9), we have a CMDP defined by Equation 5.3 (or Equation 5.7). By induction on i, we show that TMDPs admit an optimal stationary policy.

Base Case: The root nodes $i \in K$ of DAG E are first in the topological order. They are MDPs, so the Bellman optimality equation operator used in Line 6 is a contraction map in the Banach space of value functions $\hat{V}_i^\pi$ [8]. As in all MDPs, Banach's fixed point theorem proves this converges to a unique fixed point $\hat{V}_i^*$.

Induction Step: Assume true for all ancestors $\mathcal{A}_i$ of i, which is equivalent to assuming true following the sequence $\mathbf{x}$ of the topological ordering; all ancestors converged to a unique fixed point, resulting in an optimal value. By Line 5, each one was restricted by their parents in sequence. Line 6 results in action sets for each state such that each parent's local one-step slack $\eta(j,i)$ is satisfied. If there exists a state s such that $|A_i(s)| = 0$, then we can simply return infeasibility, as would occur in any CMDP whose constraints are not satisfiable. If $|A_i(s)| > 0$ for all states s, then we have an MDP with action sets at each state. Such MDPs admit the same Bellman optimality equation operator, as used in Line 6, that is again a contraction map in the Banach space of value functions $\hat{V}_i^\pi$ [8, 10]. Again, Banach's fixed point theorem proves this converges to a unique fixed point $\hat{V}_i^*$.

Thus by induction, we have shown that the k-th objective converges to a unique fixed point in value space. Therefore, local action restriction (Algorithm 3) converges to a unique fixed point $\hat{\mathbf{V}}^*$ for all values in any TMDP. □

Now that we have established that convergent values exist, we must ensure that the slack constraints are satisfied at each step of LAR in Algorithm 3. Proposition 15 formally proves that a local one-step slack of $\eta(j,i)=(1-\gamma)\delta(j,i)$ guarantees that slack is satisfied for each parent for the universal slack objective criterion.

Proposition 15. *For a TMDP and an objective $i\in K$, given the values $\hat{V}_j^*$ of i's parents $j\in\mathcal{P}_i$, if Algorithm 3's local action restriction (LAR) assigns local one-step slack $\eta(j,i)=(1-\gamma)\delta(j,i)$, then the universal slack constraint is satisfied $\hat{V}_j^*(s)-\hat{V}_j^\pi(s)\leq\delta(j,i)$ for all $s\in S$ and any policy $\pi\in\hat{\Pi}_i$.*

Proof. Let $\hat{V}_j^{\pi,t}(s)$ be t applications of Equation 5.2 following any policy $\pi\in\hat{\Pi}_i$, and let $\hat{V}_j^{*,t}(s)$ be t applications of Equation 5.2 following any optimal policy $\pi^*\in\hat{\Pi}_i$, both starting at a state $s\in S$. Additionally, $\hat{Q}_j^{*,t}(s,\pi(s))$ is a one-step action deviation following any policy π, after which an optimal policy π^* is followed for the remaining $t-1$ iterations.

First, we will show for any parent $j\in\mathcal{P}_i$ and any state $s\in S$ that:

$$\hat{V}_j^{\pi,t}(s)\geq\hat{V}_j^{*,t}(s)-\sum_{\tau=0}^{t}\gamma^\tau\eta(j,i) \tag{5.24}$$

by induction on t iterations of the Bellman equation operator.

<u>Base Case</u>: At $t=0$, by the definition of $\pi(s)\in A_i(s)$, we have local slack equation:

$$\hat{V}_j^{*,0}(s)-\hat{Q}_j^{*,0}(s,\pi(s))\leq\eta(j,i) \quad\Rightarrow\quad \hat{V}_j^{*,0}(s)-R_j(s,\pi(s))\leq\eta(j,i)$$
$$\Rightarrow\quad \hat{V}_j^{*,0}(s)-\hat{V}_j^{\pi,0}(s)\leq\eta(j,i) \quad\Rightarrow\quad \hat{V}_j^{\pi,0}(s)\geq\hat{V}_j^{*,0}(s)-\eta(j,i)$$

<u>Induction Step</u>: Assume for $t-1$ the induction hypothesis:

$$\hat{V}_j^{\pi,t-1}(s)\geq\hat{V}_j^{*,t-1}(s)-\sum_{\tau=0}^{t-1}\gamma^\tau\eta(j,i)$$

is true. We must show that Equation 5.24 true for t now:

$$\begin{aligned}
\hat{V}_j^{\pi,t}(s) &= R_j(s,\pi(s))+\gamma\sum_{s'\in S}T(s,\pi(s),s')\hat{V}_j^{\pi,t-1}(s') && \text{by Equation 5.2}\\
&\geq R_j(s,\pi(s))+\gamma\sum_{s'\in S}T(s,\pi(s),s')\Big(\hat{V}_j^{*,t-1}(s')-\sum_{\tau=0}^{t-1}\gamma^\tau\eta(j,i)\Big) && \text{by induction hypothesis}\\
&\geq \Big(R_j(s,\pi(s))+\gamma\sum_{s'\in S}T(s,\pi(s),s')\hat{V}_j^{*,t-1}(s')\Big)-\sum_{\tau=0}^{t-1}\gamma^{\tau+1}\eta(j,i) && \text{by rewrite and normalize}\\
&\geq \hat{Q}_j^{*,t}(s,\pi(s))-\sum_{\tau=0}^{t-1}\gamma^{\tau+1}\eta(j,i) \;=\; \hat{Q}_j^{*,t}(s,\pi(s))-\sum_{\tau=1}^{t}\gamma^\tau\eta(j,i) && \text{by definition of } \hat{Q}_j^{*,t}
\end{aligned}$$

By the definition of LAR's slack in Algorithm 3, Line 5, we have $\hat{V}_j^{*,t}(s) - \hat{Q}_j^{*,t}(s,\pi(s)) \leq \eta(j,i)$. Rearrange this, apply to $\hat{Q}_j^{*,t}$, and group η in the sum:

$$\hat{V}_j^{\pi,t}(s) \geq \hat{V}_j^{*,t}(s) - \eta(j,i) - \sum_{\tau=1}^{t} \gamma^\tau \eta(j,i) = \hat{V}_j^{*,t}(s) - \sum_{\tau=0}^{t} \gamma^\tau \eta(j,i).$$

Thus, by induction on t, we have shown that Equation 5.24 is true for all t.

Now let $t \to \infty$ and evaluate Equation 5.24:

$$\begin{aligned}
\hat{V}_j^{\pi,t}(s) &\geq \hat{V}_j^{*,t}(s) - \sum_{\tau=0}^{t} \gamma^\tau \eta(j,i) && \text{by Equation 5.24} \\
&\geq \hat{V}_j^{*,t}(s) - \sum_{\tau=0}^{\infty} \gamma^\tau \eta(j,i) && \text{by } \gamma \in [0,1) \text{ and } \eta(j,i) \geq 0 \\
&\geq \hat{V}_j^{*,t}(s) - \frac{\eta(j,i)}{1-\gamma} && \text{by geometric series} \\
\hat{V}_j^{*,t}(s) - \hat{V}_j^{\pi,t}(s) &\leq \frac{\eta(j,i)}{1-\gamma} && \text{by rearranging}
\end{aligned}$$

Finally, if we assign $\eta(j,i) = (1-\gamma)\delta(j,i)$, then for all $t \to \infty$ we obtain $\hat{V}_j^*(s) - \hat{V}_j^\pi(s) \leq \delta(j,i)$. □

Provided parent slack is satisfied, we can finally relate local action restriction to the two optimality criteria. Proposition 16 states the conservative over-restriction that this approximate algorithm computes at each step, as compared with the initial and universal objectives.

To this end, let the policy set $\hat{\Pi}_i = \{\pi \in \Pi | \pi(s) \in A_i(s), \forall s \in S\}$ denote the set of policies resulting from Algorithm 3's local action restriction, given the approximation of ancestor policy set restrictions via $A_i(s)$. Let $\hat{\Pi}_i^I$ and $\hat{\Pi}_i^U$ denote the initial and universal objectives applied to i, respectively, given the approximation of ancestor policy set restrictions via $\hat{\Pi}_v = \{\pi \in \Pi | \pi(s) \in A_v(s), \forall s \in S\}$ for all $v \in \mathcal{A}_i$. In other words, $\hat{\Pi}_i$ applies one optimization step of LAR at i via Algorithm 3's Lines 4-7, $\hat{\Pi}_i^I$ applies one optimization step via Equation 5.3, and $\hat{\Pi}_i^U$ applies one optimization step via Equation 5.7. In all cases, the same stationary ancestor policy sets $\hat{\Pi}_v$ are used to isolate a direct comparison of the effects.

Proposition 16. *For a TMDP and an objective $i \in K$, given the stationary policy sets $\hat{\Pi}_v$ of all i's ancestors $v \in \mathcal{A}_i$, if Algorithm 3's local action restriction (LAR) assigns local one-step slack $\eta(j,i) = (1-\gamma)\delta(j,i)$ for all $j \in \mathcal{P}_i$, then $\hat{\Pi}_i \subseteq \hat{\Pi}_i^U \subseteq \hat{\Pi}_i^I$.*

Proof. By Proposition 15, for all parents $j \in \mathcal{P}_i$, we have the universal slack constraint satisfied $\hat{V}_j^*(s) - \hat{V}_j^\pi(s) \leq \delta(j,i)$ for any policy $\pi \in \hat{\Pi}_i$. By the definition of universal slack optimality criterion in Equation 5.7, we have satisfied all constraints. This results in the policy sets $\hat{\Pi}_i^U$ following the equivalent intersection of parent policy sets in Equation 5.5:

$$\hat{\Pi}_i^U = \bigcap_{j \in \mathcal{P}_i} \hat{\Pi}_{ji}^U$$

with universal slack constraint edges in Equation 5.9:

$$\hat{\Pi}_{ji}^U = \{\pi \in \hat{\Pi}_j | \hat{V}_j^*(s) - \hat{V}_j^\pi(s) \leq \delta(j,i), \forall s \in S\}.$$

Thus, $\hat{\Pi}_i \subseteq \hat{\Pi}_i^U$. By Proposition 12, we obtain $\hat{\Pi}_i \subseteq \hat{\Pi}_i^U \subseteq \hat{\Pi}_i^I$. □

Again, since every TPOMDP can be converted to an equivalent belief TMDP, many of the proofs are identical. For example, Propositions 13 and 14 apply to TPOMDPs as well. However, while this continuous (belief) TMDP is theoretically sound and valid, in practice it is intractable for the same reasons as a POMDP, CPOMDP, or LPOMDP: it operates over $\mathcal{R}(b^0) \subseteq \triangle^{|S|}$. Instead we leverage PBVI-based methods for Algorithm 3's Line 6, which operate over a fixed set of beliefs $B \subseteq \triangle^{|S|}$. Consequently, our choice of local one-step slack $\eta(j,i)$ must change in order to preserve the slack constraints. The new equation uses PBVI-related variables, specifically the density of beliefs $\delta_B \geq 0$ and the maximal (worst-case) error after t iterations denoted as $\epsilon_i^t \geq 0$ [90]. Due to δ_B requiring a linear program to solve, we can instead use an approximation $\hat{\delta_B} \geq 0$. They are:

$$\delta_B = \max_{b' \in \triangle^{|S|}} \min_{b \in B} \|b - b'\|_1 \qquad \text{and} \qquad \hat{\delta}_B = \max_{b' \in B} \min_{b \in B} \|b - b'\|_1.$$

Also, we will assume the PBVI-based algorithm is initialized by $R_i^{min}/(1-\gamma)$ so that $\hat{V}_i^* \leq V_i^*$ is a lower bound on the true values $\hat{V}_i^* \leq V_i^*$ [79, 90]. We formally state the definition of $\eta(j,i)$ in Proposition 17 and prove it satisfies universal slack.

Proposition 17. *For a TMDP and an objective $i \in K$, given the values $\hat{V}_j^*$ of i's parents $j \in \mathcal{P}_i$, if Algorithm 3's local action restriction (LAR) assigns local one-step slack:*

$$\eta(j,i) = (1-\gamma)\delta(j,i) - \frac{R_j^{max} - R_j^{min}}{1-\gamma}\delta_B \tag{5.25}$$

then the universal slack constraint is satisfied $\hat{V}_j^*(b) - \hat{V}_j^\pi(b) \leq \delta(j,i)$ for all beliefs $b \in \triangle^{|S|}$ and any policy $\pi \in \hat{\Pi}_i$.

Proof. For any policy $\pi \in \hat{\Pi}_i$, any parent $j \in \mathcal{P}_i$, and any belief $b \in \triangle^{|S|}$:

$$\begin{aligned}
\hat{V}_j^*(b) - \hat{V}_j^\pi(b) &\leq V_j^*(b) - \hat{V}_j^\pi(b) && \text{by } R_i^{min}/(1-\gamma) \\
&\leq V_j^*(b) - (V_j^\pi(b) - \epsilon_j^t) && \text{by worst-case upper bound} \\
&\leq (V_j^*(b) - V_j^\pi(b)) + \epsilon_j^t && \text{by rewrite} \\
&\leq \frac{\eta(j,i)}{1-\gamma} + \epsilon_j^t && \text{by Proposition 15} \\
&\leq \frac{\eta(j,i)}{1-\gamma} + \frac{R_j^{max} - R_j^{min}}{(1-\gamma)^2}\delta_B && \text{by Pineau et al. [90]}
\end{aligned}$$

The worst-case upper bound comes from the fact that $\hat{V}_j^\pi(b) \in [V_j^\pi(b) - \epsilon_j^t, V_j^\pi(b)]$ after any t iterations of PBVI [90].

As in Proposition 15, we can select $\eta(j,i)$ to ensure slack $\delta(j,i)$ is satisfied. We also select a maximum of 0 to ensure a valid slack of $\delta(j,i) \geq 0$, equivalently satisfying the constraint that $\hat{V}_j^*(b) - \hat{V}_j^\pi(b) \geq 0$. Thus, if we want to show that:

$$\hat{V}_j^*(b) - \hat{V}_j^\pi(b) \leq \delta(j,i) = \frac{\eta(j,i)}{1-\gamma} + \frac{R_j^{max} - R_j^{min}}{(1-\gamma)^2}\delta_B$$

then we solve for $\eta(j,i)$ and ensure $\delta(j,i) \geq 0$ as described above:

$$\eta(j,i) = (1-\gamma)\delta(j,i) - \frac{R_j^{max} - R_j^{min}}{1-\gamma}\delta_B.$$

We return infeasible if there exist any $\eta(j,i) < 0$; or if desired, simply take the $\max\{0, \eta(j,i)\}$. $\square$

Interestingly, this assignment has the desired property that as we improve the density of our belief points (i.e., $B \rightarrow \triangle^{|S|}$ and $\delta_B \rightarrow 0$), the acceptable value of each $\eta(j,i)$ converges to the result from Proposition 15. Additionally, the bound makes intuitive sense: η describes the acceptable slack for one iteration's action's deviation from optimal. It turns out that the adjustment of $((R_i^{max} - R_i^{min})/(1-\gamma))\delta_B$ is *exactly* the definition of PBVI's one-step error [90], which obviously must be accounted for in the tolerable amount of one-step slack η.

Finally, the same logic as Proposition 16, instead leveraging Proposition 17 and Corollary 4, allows us to arrive at the same relation using PBVI-based local action restriction in Corollary 6.

Corollary 6. *For a TPOMDP and an objective $i \in K$, given the stationary policy sets $\hat{\Pi}_v$ of all i's ancestors $v \in \mathcal{A}_i$, if Algorithm 3's local action restriction (LAR) assigns local one-step slack in Equation 5.25 for all $j \in \mathcal{P}_i$, then $\hat{\Pi}_i \subseteq \hat{\Pi}_i^U \subseteq \hat{\Pi}_i^I$.*

5.6.2 Policy Network Representation of TMDPs

We now prove in Proposition 18 and Figure 5.3 how any TMDP (or TPOMDP) is representable as a policy network. In the proof, we refer directly to the formal definition established in Section 5.2, and assume the initial slack objective criterion. However, we will refer to the TMDP's edges as E^T to differentiate it from the policy network's edges E. TMDPs serve as an example of how policy networks can describe iteratively constraining each other's policy spaces with the same state and action spaces across all vertices. It demonstrates how policy networks can allow for rich sets of constraints, which are organized hierarchically, to properly model the kinds of complex multi-objective problems found in real-world domains.

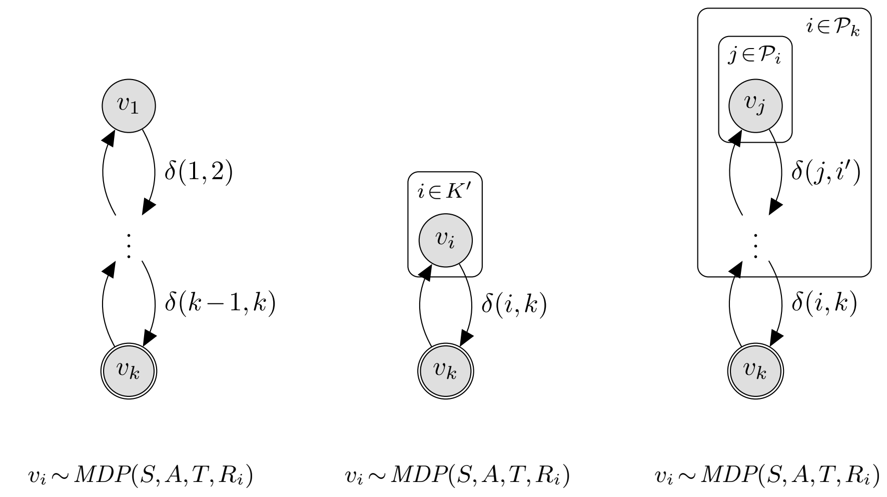


Figure 5.3: LMDP (left), CMDP (center), and an example tree graph TMDP (right) represented as a policy network, with $K' = K - \{k\}$ and $i' \in \mathcal{P}_k$.

Proposition 18. *Policy networks generalize TMDPs.*

Proof. For any TMDP, we must construct an equivalent policy network. See Figure 5.3. Let $V = \{v_i, \forall i \in K\}$ with $v \sim MDP(\cdot)$ as in the figure with shared state and action spaces. Let $E = \{\langle v_i, v_j \rangle, \langle v_j, v_i \rangle, \forall \langle j, i \rangle \in E^T\}$ (bi-directional) with edges $e = \langle v_j, v_i \rangle$ having policy constraint:

$$\Pi_e = \{\pi \in \Pi_{v_j} | V^*_{v_j}(s^0) - V^{\pi}_{v_j}(s^0) \leq \delta(j,i)\}.$$

Thus, this policy network produces the same Bellman equations and initial slack optimality criterion of a TMDP. $\square$

5.7 Evaluation

Semi-autonomous systems require collaboration between a human and an agent in order to achieve a goal [26, 146]. We experiment within the previously described autonomous vehicle domain. In this domain, an AV may only drive autonomously on some subset of autonomy-capable roads, requiring manual driving on the remaining roads. The driver may be *attentive* or *tired* [142, 135]. The AV must reason about the driver capability to minimize travel time; however, we allow for a slack in travel time in order to improve the amount of time spent autonomous.

We use a real-world dataset and allow for autonomy whenever the speed limit is greater than 30 miles per hour—that is, on main roads. State transitions capture the likelihood that the human driver will drift from attentive to tired. In the LMDP case, the AV's location, who is in control, and driver's state of attentiveness are perfectly observable by the interior sensors. In the LPOMDP case, we relax the assumption to allow for noisy observations over the driver's state.

Tables 5.2 and 5.3 show the problem sizes and run times over 10 cities, as well as the values of both objective functions at the initial state or belief. The LPOMDP case used a uniform belief over the attentiveness of the driver. To improve the scalability of local action restriction for both the LMDP's value iteration and the LPOMDP's PBVI, we developed a GPU-based parallel variant for both LMDPs and LPOMDPs.

Experiments were conducted with an Intel(R) Core(TM) i7-4702HQ CPU at 2.20GHz, 8GB of RAM, and an Nvidia(R) GeForce GTX 870M graphics card using C++ and CUDA(C) 6.5. The results demonstrate that the GPU implementation can produce a speedup of more than two orders of magnitude over the CPU implementation. Note, however, that our CPU implementation here used STL objects, whereas the GPU used arrays. In improved version of this code is provided in *nova* [136] with continued development in the library resulting in a speedup closer to an order of magnitude.

Figures 5.4 and 5.5 demonstrates an policies for a sections of Denver, Austin, San Francisco, and Boston. The arrows mark the policy at each intersection; green arrows mean the driver drivers, and purple arrows mean the AV drives. The light blue roads denote autonomy-capable roads, and the white roads denote non-autonomy-capable roads. The green path denotes the optimal policy given an initial starting point. For the LPOMDP, values and actions exist within an impossible-to-visualize high-dimensional belief space. As such, we simply took two "slices" over this space: assuming the physical state and autonomy were observable, as in the model, we assigned a probability of being tired to 0.2 (i.e., likely attentive) and 0.8 (i.e., likely tired).

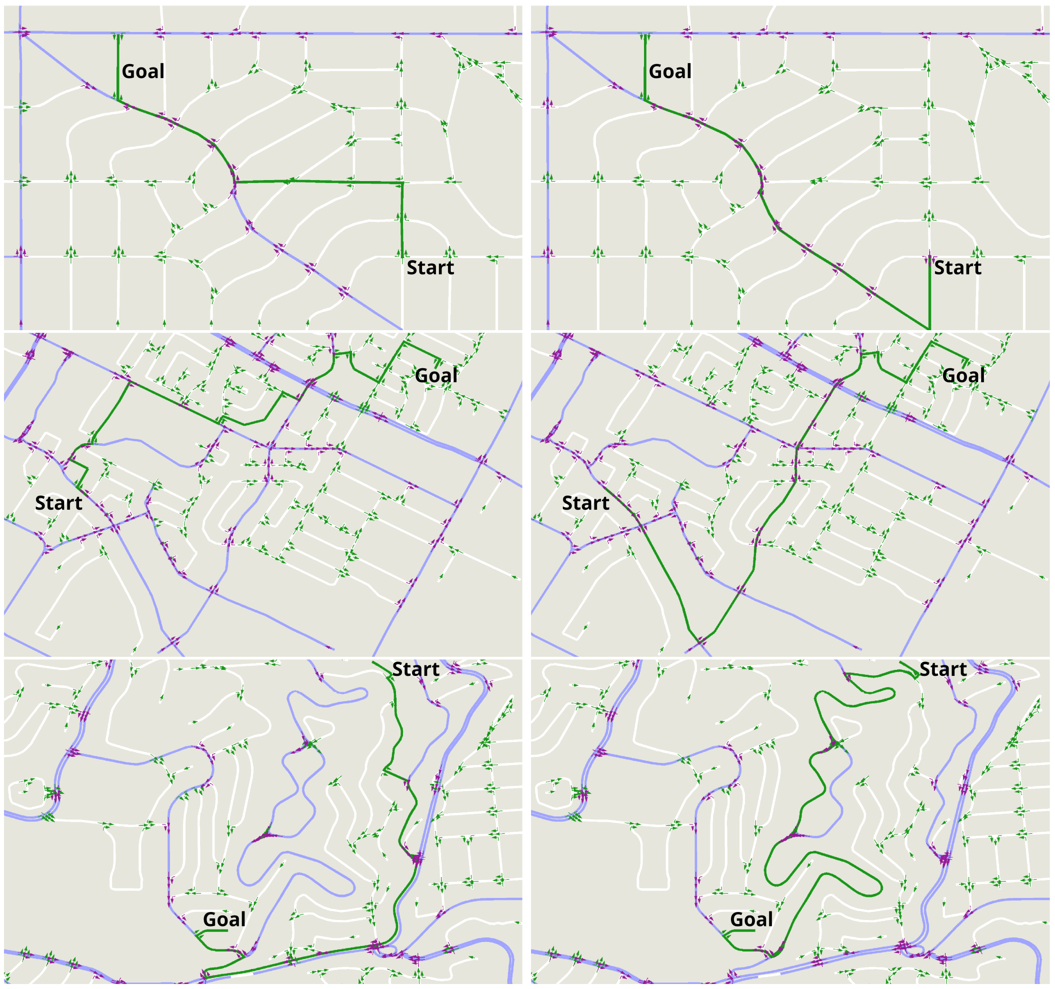


Figure 5.4: Example LMDP policy for Denver (top), Austin (center), and San Francisco (bottom) with driver attentive (left) and tired (right).

City	$\|S\|$	$\|A\|$	CPU	GPU
Chicago	400	10	3.3	3.9
Denver	616	10	12.9	6.4
Baltimore	676	8	14.1	5.7
Pittsburgh	864	8	15.4	7.9
Seattle	1168	10	63.5	14.2
Austin	1880	10	433.3	29.8
San Francisco	2016	10	4685.7	159.4
Los Angeles	2188	10	273.5	37.8
Boston	2764	14	11480.9	393.2
New York City	3608	10	16218.5	525.7

Table 5.2: LMDP problem domain sizes ($|S|$ and $|A|$) and run times (CPU and GPU) for autonomous vehicle LMDP experiments over 10 cities using real-world data from OpenStreetMap.

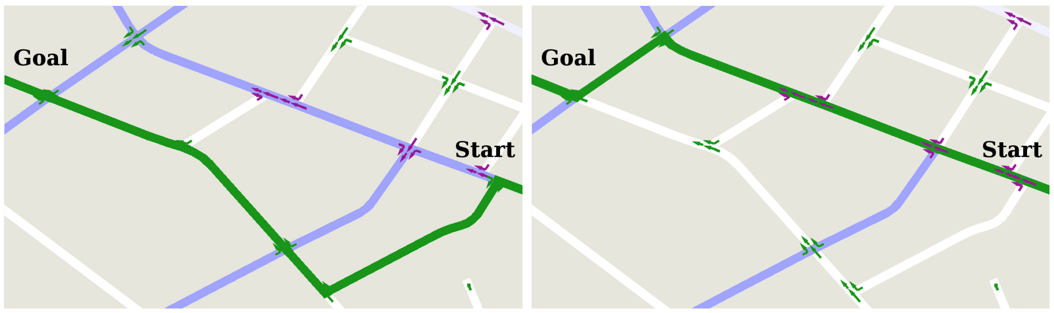


Figure 5.5: Example LPOMDP policy for Boston with driver belief 0.2 (left) and 0.8 (right).

City Name	$\|S\|$	$\|A\|$	$\|\Omega\|$	$\|B\|$	$\hat{V}_1^\pi(b^0)$	$\hat{V}_2^\pi(b^0)$	CPU	GPU
Austin	92	8	2	230	57.4	35.9	14.796	3.798
San Francisco	172	8	2	430	97.8	53.8	51.641	8.056
Denver	176	8	2	440	123.7	77.3	60.217	8.299
Baltimore	220	8	2	550	56.2	43.9	104.031	11.782
Pittsburgh	268	10	2	670	148.0	142.2	169.041	19.455
Los Angeles	380	8	2	950	167.9	114.4	298.794	25.535
Chicago	404	10	2	1010	67.4	31.6	399.395	36.843
Seattle	432	10	2	1080	111.2	66.9	497.061	48.204
New York City	1064	12	2	2660	108.1	73.7	—	351.288
Boston	2228	12	2	5570	109.3	79.2	—	2424.961

Table 5.3: LPOMDP problem domain sizes ($|S|$, $|A|$, and $|\Omega|$), the value following the policy π at initial beliefs ($\hat{V}_i^\pi(b^0)$), and run times (CPU and GPU) for autonomous vehicle LPOMDP experiments over 10 cities using real-world data from OpenStreetMap.

We observe in all examples that the policy produces the desired result of first minimizing the distance to get from the start to the goal, but also considering the time spent autonomous, namely when the driver is tired. For example, in Figure 5.4's Austin example—the center two subfigures—we observe that the primary "minimize time" objective (Equation 5.21) is clearly chosen over the secondary "minimize time spent not autonomus" objective (Equation 5.22). Note that Equation 5.21 does not condition its reward on the driver's level of fatigue, meaning if the slack $\delta = 0$ then both attentive and tired state factor assignments (i.e., $x = \alpha$ or $x = \tau$) would produce the same path that minimizes time. However, since our slack is positive $\delta > 0$, policies that favor the secondary objective— drive on autonomy-capable roads (in blue)—is chosen when the state factor is tired $x = \tau$. If we were to allow $\delta \to \infty$ then the agent would ignore the primary objective in complete favor of the secondary objective. Slack allows for a natural tuning parameter between any pair of objective functions, with a clear mathematical formalism, and a useful practical description understablable by an end-user, say of an autonomous vehicle.

5.8 Conclusion

We present a model for a multi-objective MDP and POMDP that admits a hierarchical preference structure over the collection of objectives. The hierarchical preference structure is a directed acyclic graph (DAG) that induces a topological order over the objectives. As such, we call the model a topological MDP (TMDP) or topological POMDP (TPOMDP) depending if it has full or partial observability, respectively. For each pair of objectives directly connected by the DAG, we define a slack constraint edge—that is, allowable deviation from optimal of the parent objective in favor of the child objective's optimization. This model's characterization of slack constraints and the generality in preferences allows for a natural description of many real-world problem domains with multiple, prioritized objectives.

We rigorously prove that the T(PO)MDP generalizes a number of previous models, such as the lexicographic MDP (LMDP) and constrained MDP (CMDP). We present two forms of objective criteria, initial and universal slack, proving their relation. We also provide a number of algorithms for T(PO)MDPs: (1) an optimal algorithm that solves the T(PO)MDP; two approximate algorithms for TMDPs using: (1) non-linear programming, and (2) local action restriction (LAR); and an approximate PBVI-based TPOMDP algorithm for TPOMDPs that uses LAR as well. We formally prove the relationship of these algorithms with the true and approximate values returned by the objective criteria.

To illustrate our model's applicability to real-world problems, we examine its performance within the recently proposed semi-autonomous driving domain. Our experiments show that our algorithms can solve practical TMDPs and TPOMDPs with large state-spaces–in line with the capabilities of state-of-the-art POMDP solvers.

In future work, we will expand our investigation of TMDPs and TPOMDPs and their applications. For example, the area of proactive learning—query multiple cost-varying oracles for noisy labels to build an accurate dataset and classifier—must trade-off a budget, belief over dataset accuracy, and the number of queries [137]. We hope to explore larger hierarchical preference structures over the objectives. Finally, we will provide our source code to facilitate the creation and use of TMDPs and TPOMDPs in practical real-world systems.

CHAPTER 6
SEMI-AUTONOMOUS SYSTEMS

In this chapter, we present a formal model for a semi-autonomous system—an agent who can be controlled by many distinct actors, including a human. Importantly, we discuss the critical transfer of control process between each of the actors. We illustrate how this can be used for autonomous vehicle route planning that proactively reasons about the second-to-second control transfer between a human driver and the vehicle. More broadly, the semi-autonomous system model shows how humans can be integrated into an agent system to leverage the human's capabilities at key points. This enables it to overcome the rare but inevitable situations in which the agent cannot properly act, given its known limitations. A small bit of human aid can make an otherwise undeployable AI or robot actually deployable, as is the case with autonomous vehicles.

This model has a mathematical mechanism that facilitates a two-level hierarchy. It is a demonstration of the transfer of control process between models in a policy network. This is formally proven and discussed. In general, semi-autonomous systems are best used for domains in which there are multiple controllers of a single robotic agent, namely any number of humans or other agents, each of which can approve actions or even gain control of the agent to greatly expand the ability for the system to accomplish its goals.

6.1 Introduction

Autonomous systems have been deployed in a wide variety of applications such as space exploration [147], reservoir control [23], energy conservation [75], and autonomous driving [134]. These systems, however, almost universally require human intervention or interaction at some point in order to achieve their objectives (e.g., the Mars rovers), or recover from failure (e.g., the Roomba vacuum cleaner). Within the proposed automated planning solutions to these problems, few if any approaches take full advantage of this collaboration. Instead, they commonly resort to default hard-coded behaviors instead of integrating human capabilities into the planning process [11]. Semi-autonomous systems (SAS) capture explicitly this collaborative process in which a human and an agent—or any number of actors—work together to achieve a goal, smoothly transferring control over the system back and forth, while proactively considering each actor's capabilities and the human's preferences [146, 132].

New challenges arise in semi-autonomous systems because the overall plan must factor the inherent uncertainty and unpredictability associated with human behavior. We consider a semi-autonomous driving domain where the vehicle can operate autonomously only on well-mapped roads under ideal conditions. To reach a distant destination, the vehicle may require the human to occasionally take control. This transfer of control process requires second-to-second monitoring as various messages are conveyed to the driver. It is also not always successful given an allotted time window and the driver's state (e.g., distracted). These factors must be taken into consideration as the system is planning its long-term route. Additionally, this process of transfer of control must incorporate these factors to provide a measure of safety for the system. Car companies are already developing nascent semi-autonomous capabilities and user interfaces to support transfer of control [85], but research has been sparse on generalized planning models.

Previous work on semi-autonomous systems has focused on preventing or reacting to human error [4], for example, automatically correcting an undesired lane change [61] or human-reactive implementations of adaptive cruise control [99]. While a long line of research exists on collaboratively controlling a system [27], planning with the explicit consideration of the human in the plan execution cycle has been lacking [42]. No existing algorithm explicitly tackles the transfer of control problem.

Our proposed collaborative multiagent framework is quite distinct from existing approaches for collaboration such as SharedPlans [50], Teamwork [123], and Dec-POMDP [9]. First, a SAS requires exactly one actor to be in control of plan execution at any given time. Second, this fact requires explicit mechanisms for transferring control among the actors. Finally, a SAS must proactively plan to leverage each actor's capabilities (or lack thereof) as it efficiently moves in the state space.

Our primary contributions are: (1) a formal definition of a SAS and its key properties (Section 6.2); (2) a general transfer of control model as a hierarchical approach for integrating domain action planning with transfer of control (Section 6.3); (3) an application of SAS for semi-autonomous vehicles (Section 6.4); (4) an analysis showing the hierarchical model is a *strong SAS* (Section 6.5); (5) a formal representation of SAS as a policy network (Section 6.5.1; and (6) an evaluation of the approach for 10 cities using real road data that show the benefits of the method (Section 6.6).

6.2 Semi-Autonomous Systems

Semi-autonomous systems (SAS) rely on collaboration between a human and an agent in order to achieve some goals while maintaining a measure of safety [146]. We consider semi-autonomy within the context of automated planning, extending an SSP to support semi-autonomy.

Formally, a **semi-autonomous system (SAS)** is a sequential decision-making model defined by the tuple $\langle I, \mathbf{S}, \mathbf{A}, T, C, G, L \rangle$:

- I is a set of actors (i.e., the controlling entities);
- $\mathbf{S} = S \times I$ is a set of factored states: a standard states S and the current controlling actor I;
- $\mathbf{A} = A \times I$ is a set of factored actions: a standard actions A and the next desired actor I;
- $T : \mathbf{S} \times \mathbf{A} \times \mathbf{S} \rightarrow [0,1]$ is a transition function, comprised of a state transition $T_i : S \times A \times S \rightarrow [0,1]$ for each actor $i \in I$, and control transfer function $\rho : \mathbf{S} \times I \times I \rightarrow [0,1]$;
- $C : \mathbf{S} \times \mathbf{A} \rightarrow \mathbb{R}^+$ is a cost function;
- $G \subseteq \mathbf{S}$ is a set of goal states; and
- $L \subseteq \mathbf{S}$ is a set of live states with $L = \{\langle s, i \rangle \in \mathbf{S} | i \in \psi(s)\}$ for actor capability function $\psi : S \rightarrow 2^I$.

The actors I of the system describe controlling entities, which include at a minimum an autonomous agent ν and possibly a human λ; we focus in this work on situations involving these specific two actors. Note the distinction made between a generic *agent* versus an *actor*. Only one entity is acting in the system, though multiple exist, as opposed to the more general agent which might act simultaneously as others such as in a Dec-POMDP.

The states must record who is in control at any given time, and the actions must record intentions to switch control to new actors. In SAS, we cannot always assume that transfer of the control has a flawless execution. Hence our T is factored into two components: T_i and ρ. Formally, the **actor state transition function**, denoted $T_i : S \times A \times S \rightarrow [0,1]$, describes how an actor $i \in I$ can operate in the world when in control. The **control transfer function**, denoted $\rho : \mathbf{S} \times I \times I \rightarrow [0,1]$, describes the result of attempting to transfer control from the current actor in a given state. The **SAS state transition function** for $\mathbf{s} = \langle s, i \rangle$, $\mathbf{a} = \langle a, \hat{i} \rangle$, and $\mathbf{s}' = \langle s', i' \rangle$ is:

$$T(\mathbf{s}, \mathbf{a}, \mathbf{s}') = \begin{cases} T_i(s, a, s'), & \text{if } i = \hat{i} = i' \\ T_i(s, a, s') \rho(\mathbf{s}, \hat{i}, i'), & \text{if } i \neq \hat{i} \\ 0, & \text{otherwise} \end{cases} . \tag{6.1}$$

In Equation 6.1, the first component corresponds to keeping the current actor, which simply follows the actor's state transition. The second component describes the actor still in control but seeking to switch to a different actor at the next state. The third component indicates that it is impossible to take control from an actor without the desire to transfer.

We develop a hierarchical approach to transfer control that treats each decision of the high-level planning process as a macro-action or an *option* [122], which involves micro-actions to support the successful transfer of control. This hierarchical design seems particularly suited for our target domain of semi-autonomous driving. Here, we perform path planning on large, world-scale roads in time scales of minutes or hours. Transfer of control, however, requires much more care, and is done in time scales of seconds. If we were to path plan with transfer of control at full detail everywhere along the route, the state spaces would be astronomically large. For example, for the *smallest* problem instance in our experiments (Pittsburgh), this would blow up the state space by a factor of 387, resulting in a POMDP with approximately 7.6×10^4 states. Instead, we take advantage of the fact that transfer of control is a generic process that depends on a handful of context variables such as the time remaining to complete the transfer and some general driving conditions (e.g., transferring control on a straight road, turns, low-speed, and high-speed). Apart from that, the way in which the transfer of control is performed is largely independent of the remaining route and destination. This enables us to generalize the transfer process and model it as a compact state transition at the higher level following ρ.

Following Zilberstein [146], a **SAS of type I (SAS-I)** does *not* explicitly model the human in the execution loop, whereas a **SAS of type II (SAS-II)** does. Thus, we have presented a SAS-II as we explicitly model the human within our set of actors ($\lambda \in I$).

Within a SAS, we define two types of histories. First we define the meaning of any trajectory over states and actions, given the limits of the stochastic state transition. This is called a **realizable history** is a sequence of the form $\bar{h} = \langle \mathbf{s}^0, \mathbf{a}^0, \ldots, \mathbf{s}^\ell \rangle$ such that for all $\mathbf{s}^t$, $\mathbf{a}^t$, and $\mathbf{s}^{t+1}$, $T(\mathbf{s}^t, \mathbf{a}^t, \mathbf{s}^{t+1}) > 0$. The set of all realizable histories starting at $\mathbf{s}^0 \in \mathbf{S}$ with horizon $\ell \in \mathbb{N}$ is denoted $\bar{H}(\mathbf{s}^0, \ell)$. Next, we define a more constrained history with respect to a specific *policy*. Here a policy $\pi : \mathbf{S} \rightarrow \mathbf{A}$ is a mapping from factored states to factored actions. Formally, given policy π, a **policy realizable history**, is a realizable history $\bar{h}$ such that $\forall t, \mathbf{a}^t = \pi(\mathbf{s}^t)$. We denote the set of all policy realizable histories starting at state $\mathbf{s}^0 \in \mathbf{S}$ with horizon $\ell \in \mathbb{N}$ as $\bar{H}_\pi(\mathbf{s}^0, \ell)$.

Given a policy π, the agent incurs a cost per time step given by $C : \mathbf{S} \times \mathbf{A} \rightarrow \mathbb{R}^+$ as it tries to reach a *goal state* from $G \subseteq \mathbf{S}$. For initial state $\mathbf{s}^0 \in \mathbf{S}$, the objective is to find a policy π that minimizes the expected cost:

$$\mathbb{E}\Big[\sum_{t=0}^{\infty} C(\mathbf{s}^t, \pi^*(\mathbf{s}^t)) | \pi, \mathbf{s}^0\Big]. \tag{6.2}$$

A policy is *optimal* if it minimizes the expected cost over time, also called the *value* of a state $V : \mathbf{S} \to \mathbb{R}$. This defines an SSP. The Bellman optimality equation in SAS for state s:

$$V(\mathbf{s}) = \min_{\mathbf{a} \in \mathbf{A}} \Big(C(\mathbf{s}, \mathbf{a}) + \sum_{\mathbf{s}' \in \mathbf{S}} T(\mathbf{s}, \mathbf{a}, \mathbf{s}') V(\mathbf{s}') \Big). \tag{6.3}$$

Following Bertsekas and Tsitsiklis [10], this equation produces an optimal policy π^* under two assumptions. First, a *proper policy* must exist that can reach a goal with probability 1 from s. Second, all *improper policies* must incur infinite cost at states that cannot reach a goal with probability 1. Such SSPs can be solved using search methods such as LAO* [52].

So far we have described a concrete model that explicitly represents the current actor (controlling entity) and the dynamics for attempting to transfer control among actors. We now introduce **actor capability constraints** for SAS, which specify limits on the abilities of actors to control the system under certain conditions, through the function ψ. Formally, an **actor capability function (ACF)** $\psi : S \to 2^I$ maps states to the actors capable of acting in that state. For example, in the semi-autonomous driving domain, the autonomous agent may not be able to drive on every road, but only on well-mapped roads or under certain weather conditions. Thus, the planner must incorporate the limited capabilities of the autonomous agent, as well as the uncertainty regarding transfer of control between the human and agent, in order to construct a route from a starting location to a destination.

The actor capability function defines a formal notion of *live state*, that is a state in which an actor can control the system. Thus, **live states** $L = \{\langle s, i\rangle \in \mathbf{S} | i \in \psi(s)\}$ are states which satisfy the ACF ψ. Live states are states in which the system is considered *active* or *safe* because the controlling entity can act there. We require these constraints to be satisfied for all SAS. Formally, **live state constraints** are: (1) $G \subseteq L$ and (2) $\forall \mathbf{s} \notin L$, $\forall \mathbf{a} \in \mathbf{A}$, $\forall \mathbf{s}' \in L$, $T(\mathbf{s}, \mathbf{a}, \mathbf{s}') = 0$.

Unlike general *dead ends* [68], which are states from which the goal becomes unreachable, our live state constraints form a particular *structured* type of dead end that is easier to analyze (largely because these conditions are explicitly captured by L). In fact, we will show that our semi-autonomous vehicle formulation produces policies that guarantee avoidance of non-live states. In general, this requires us to prove that the given transfer of control model produces a ρ that never enters a non-live state. This key mechanism is described in the following section. We now formalize this.

Our objective is to characterize *policies* and *systems* in terms of their ability to *maintain live state*. We present three key properties of policies and SAS themselves. A policy π is strong if for all $\mathbf{s}^0 \in L$ and $\ell \in \mathbb{N}$, and for all $\bar{h} \in \bar{H}_\pi(\mathbf{s}^0, \ell)$ and $t \in \{0, \dots, \ell\}$, $\mathbf{s}^t \in L$. A SAS is **strong** if there exists a optimal strong policy. A policy π is conditionally strong if there exists an $\mathbf{s}^0 \in L$ and $\ell \in \mathbb{N}$, such that for all $\bar{h} \in \bar{H}_\pi(\mathbf{s}^0, \ell)$ and $t \in \{0, \dots, \ell\}$, $\mathbf{s}^t \in L$. A SAS is **conditionally strong** if an optimal conditionally strong policy exists. A policy π is weak if it is not strong or conditionally strong. A SAS is **weak** if its optimal policies are weak. Finally, we extend these terms from policies to an entire SAS. A SAS is said to be **strong** (**conditionally strong**) if there exists a strong (conditionally strong) policy π^* that is optimal. Otherwise, the SAS is said to be **weak**.

6.3 Transfer of Control

Transfer of control (TOC) is the critical method that enables effective and safe transference of the controlling entity within the stochastic decision-making process. TOC both to and from a human requires the optimal selection of various messages (e.g., visual or auditory) in order to prompt the human to reengage and ensure smooth transference. Each message type presents a trade-off between the efficacy of alerting the human to the agent's intention and the human's amiable perception of the agent (e.g., aggregated annoyance). For example, a continuous alarm is effective but undesirable, and a blinking light is not as effective but more favorable. Additionally, the system receives noisy observations of the human's state of engagement due to the limited sensing capabilities available. Thus, it instead must make decisions based on a belief regarding the engagement level. Finally, this is a time-sensitive sequential optimization problem due to the limited time window in which control may be transferred. For example, in semi-autonomous driving, the vehicle may not be able to operate on insufficiently mapped roads and must seamlessly relinquish control before reaching these roads. We model this process using a POMDP. First, we formally define the TOC problem, then POMDPs, and finally construct the POMDP model of TOC.

6.3.1 Problem Definition

The **transfer of control (TOC) problem** is a tuple $\langle \mathcal{H}, \mathcal{M}, \mathcal{O}, \mathcal{T}, \mathcal{P}_h, \mathcal{P}_c, \mathcal{P}_o, \mathcal{C} \rangle$. $\mathcal{H}$ is a set of human states. $\mathcal{M}$ is a set of available messages to inform the user of the desire to transfer control. The absence of a message is indicated by $\emptyset \in \mathcal{M}$ (i.e., no operation or 'NOP') and is always available. $\mathcal{O}$ is the set of observations made by sensors, which provide partial information about the human's state. $\mathcal{T} = \{1, \dots, \tau\}$ is a set of limited time steps for the transfer of control to complete (e.g., τ seconds). $\mathcal{P}_h : \mathcal{H} \times \mathcal{M} \times \mathcal{T} \times \mathcal{H} \rightarrow [0,1]$ is the probability of the human state transitioning from h

to h' given message m was sent t time steps ago, such that we have $\mathcal{P}_h(h,m,t,h') \equiv Pr(h' \mid h,m,t)$. $\mathcal{P}_c : \mathcal{H} \times \mathcal{M} \times \mathcal{T} \to [0,1]$ is the probability that control will be transferred given the human state h and that message m was sent t time steps ago. If control is transferred, then the process terminates; the agent knows when this occurs. $\mathcal{P}_o : \mathcal{H} \times \mathcal{O} \to [0,1]$ is the probability of making a sensor observation o given the human state is h, such that $\mathcal{P}_o(h,o) \equiv Pr(o|h)$. $\mathcal{C} : \mathcal{H} \times \mathcal{M} \times \mathcal{T} \to \mathbb{R}^+$ is the cost of sending message m given human state h and t time steps since sending the last message (i.e., $\mathcal{C}(h,m,t)$). The agent can always **abort**, ending the transfer attempt.

The human has a true hidden state $h \in \mathcal{H}$. This changes over time as the agent selects a message $m_t \in \mathcal{M}$ for each $t \in \mathcal{T}$, forming sequence $m = \langle m_1, \ldots, m_\tau \rangle$. The objective is to *minimize the total sum of message costs*; however, failing to transfer control without safely aborting should strictly be avoided. Thus, the agent must also decide when to abort.

Importantly, control can be transferred either way in this model. That is, it captures requesting control to be *both* taken from and given to the agent. Furthermore, different TOC problems may be defined, each encoding a *different* environment or scenario in which control must be transferred. For example, a vehicle taking control on a highway turn, or a human taking control on a quiet suburban road, are both different transfer of control instances.

6.3.2 Transfer of Control POMDP Formulation

We model the control transfer problem as a POMDP called a **TOC POMDP** $\langle \bar{S}, \bar{A}, \bar{\Omega}, \bar{T}, \bar{O}, \bar{R} \rangle$. The state space is $\bar{S} = \mathcal{T} \times \mathcal{H} \times \mathcal{M} \times \mathcal{T} \cup \mathcal{E}$ with a set of 'end result' states $\mathcal{E} = \{\oplus, \ominus, \oslash\}$ denoting 'success,' 'failure,' and 'aborted,' respectively. Each state captures the time remaining, current human state, the previous message sent, and how long it has been since that message was sent, as well as the outcome of the transfer of control. The action space $\bar{A} = \mathcal{M} \cup \{\oslash\}$ is the messages to send and the 'abort' action (denoted $\oslash$). The observation space $\bar{\Omega} = \mathcal{O} \cup \mathcal{E}$ represents the observations, and lets the model know the end result, as per the problem definition. The state transition function needs to encapsulate the notions of human state, as well as the success or failure of transferring control. We break this into two scenarios.

The first scenario examines transitions only among non-end result states such that $s, s' \notin \mathcal{E}$, with state factors denoted $s = \langle t, h, m, t_m \rangle$ and $s' = \langle t', h', m', t'_m \rangle$. This scenario has two non-zero cases each with the same probability. In both, control has not successfully transferred yet, so we always update the human state and count down the timer (via constraint $t' = t - 1 \geq 0$). The first case encodes the effect of sending a message from $\mathcal{M} \setminus \{\emptyset\}$. The second case encodes the effect of a 'NOP' message $\emptyset$. Formally, for any s, a, and s':

$$\bar{T}(s,a,s') = \begin{cases} \bar{\mathcal{P}} & \text{if } a \notin \{\oslash, \emptyset\} \wedge m' = a \wedge t'_m = 0 \\ \bar{\mathcal{P}} & \text{if } a = \emptyset \wedge m' = m \wedge t'_m = \min\{t_m + 1, \tau\} \\ 0 & \text{otherwise} \end{cases} \tag{6.4}$$

with $\bar{\mathcal{P}} = (1 - \mathcal{P}_c(h,m,t_m))\mathcal{P}_h(h,m,t_m,h')$ above.

The second scenario examines transitions to an end result state: successor $s' \in \mathcal{E}$. This scenario has four non-zero cases. The first case is simply the absorbing states $\mathcal{E}$. The second case is immediate termination via the abort action $\oslash$. The third case captures the ever-possible chance of successful control transfer ($\oplus$). The fourth case handles a failure ($\ominus$) transition by running out of time. Thus, for a state s, action a, and successor s' we have:

$$\bar{T}(s,a,s') = \begin{cases} 1 & \text{if } s = s' \in \mathcal{E} \\ 1 & \text{if } s \notin \mathcal{E} \wedge a = s' = \oslash \\ \mathcal{P}_c(h,m,t_m) & \text{if } s \notin \mathcal{E} \wedge a \neq \oslash \wedge s' = \oplus \\ 1 - \mathcal{P}_c(h,m,t_m) & \text{if } s \notin \mathcal{E} \wedge t = 0 \wedge a \neq \oslash \wedge s' = \ominus \\ 0 & \text{otherwise} \end{cases} \tag{6.5}$$

with $s = \langle t,h,m,t_m \rangle$ above provided $s \notin \mathcal{E}$.

The observation transition function only needs to model two components. First, the agent always has perfect knowledge of the final outcome state. Second, the agent makes noisy observations from sensors which hint at the true human state (e.g., a face or eye tracker in a vehicle). Formally, for action a, successor state s', and observation ω:

$$\bar{O}(a,s',\omega) = \begin{cases} 1 & \text{if } \omega = s' \in \mathcal{E} \\ \mathcal{P}_o(h',\omega) & \text{if } s' \notin \mathcal{E} \wedge \omega \in \mathcal{O} \\ 0 & \text{otherwise} \end{cases} \tag{6.6}$$

with states $s' = \langle t',h',m',t'_m \rangle$ above provided $s' \notin \mathcal{E}$.

The reward function has four components. First, there are costs associated with all normal messages, as defined by the TOC problem. Second, an arbitrarily small $\epsilon > 0$ cost is given for a NOP $\emptyset$. Third, there is a large penalty for unnecessary aborting. Fourth, failure repeatedly incurs the maximal cost. Thus, for state s and action a:

$$\bar{R}(s,a)=\begin{cases}-\mathcal{C}(h,m,t_m) & \text{if } s\notin\mathcal{E}\wedge a\notin\{\emptyset,\oslash\}\\ -\epsilon & \text{if } s\notin\mathcal{E}\wedge a=\emptyset\\ -\mathcal{C}^* & \text{if } s\notin\mathcal{E}\wedge a=\oslash\wedge t>0\\ -\mathcal{C}^* & \text{if } s=\ominus\\ 0 & \text{otherwise}\end{cases} \tag{6.7}$$

with $s=\langle t,h,m,t_m\rangle$ and a non-success penalty $\mathcal{C}^*$ (e.g., $\mathcal{C}^*=\ell\mathcal{C}_{max}$ for $\mathcal{C}_{max}=\max_{h,m,t_m}\mathcal{C}(h,m,t_m)$).

Within a TOC POMDP, the end result terminal states are fully observable, as well as each state factor *except* for the true human state $\mathcal{H}$. So all beliefs, including initial belief b^0, only contain uncertainty regarding these human states. Thus, for true initial state s^0, the initial state b^0 is defined as:

$$b^0(s)=\begin{cases}1 & \text{if } s^0\in\mathcal{E}\wedge s=s^0\\ 1/|\mathcal{H}| & \text{if } s^0\notin\mathcal{E}\wedge t=t^0\wedge m=m^0\wedge t_m=t_m^0\\ 0 & \text{otherwise}\end{cases} \tag{6.8}$$

with states s=$\langle t,h,m,t_m\rangle$ above provided $s\notin\mathcal{E}$.

6.4 Application to Autonomous Vehicles

The TOC formulation as a POMDP enables us to incorporate semi-autonomy into stochastic path planning problems. Again, our main motivation is the semi-autonomous driving domain. Route decisions are made at intersections of roads; however, only well-mapped main roads are capable of autonomy. While the driver can drive on any road, the longer, uninteresting, boring highways are assumed to be roads in which the human prefers autonomy, meaning that control should be transferred to the vehicle. All costs are proportional to the time spent on the road. The uncertainty stems from the transfer of control, also decided at road intersections. We first formally define the problem, then describe the full model of a specific SAS called a semi-autonomous vehicle (SAVE).

6.4.1 Problem Definition

The **semi-autonomous vehicle (SAVE) problem** begins with a strongly connected weighted directed graph $\langle V,E,w\rangle$. V is a set of vertices forming intersections. $E\subseteq V\times V$ is a set of pairs of vertices (intersections) defining edges which form roads. $w:E\rightarrow\mathbb{R}^+$ defines a positive weight for each edge (road) which captures the time spent on the road. There are initial and goal vertices denoted $v^0\in V$ and $v^g\in V$, respectively.

Additionally, the system may be driven by the human or the agent (vehicle) itself; however, given the allotted time between each vertex (intersection), there is uncertainty if control transfer will be successful. This micro-level behavior is modeled using TOC POMDPs. (These may be solved offline for different scenarios, as described in the previous section.) We label edges (roads) $E_c \subseteq E$ as **autonomy-capable**, meaning the set of roads in which the vehicle is capable of driving autonomously. Similarly, $E_p \subseteq E_c$ are **autonomy-preferred** roads in which the human prefers autonomous driving. Thus, at each intersection the agent must decide if control should be maintained or transferred as well as which road to take next, in order to *minimize the sum of traversed weights (travel time)*. If control transfer is required, fails to succeed, but the agent aborted, then the system is assumed to safely pause at the vertex, i.e., the vehicle safely pulls over to the side of the road. Otherwise, the SAS will enter an unsafe state in which the vehicle is in control but cannot drive on the road; this should be avoided at all costs.

6.4.2 Semi-Autonomous Vehicle Formulation

A **semi-autonomous vehicle (SAVE)** is a SAS with $\langle I, \mathbf{S}, \mathbf{A}, T, C, G, L \rangle$. Actors $I = \{\lambda, \nu, \sigma\}$ encode the current controlling agent of the vehicle: either the *human* λ, the *vehicle* itself ν, or no active actor as it *safely waits* on the side of the road σ. $\mathbf{S} = V \times I$ have standard states V corresponding to the vertices (intersections) of the map. $\mathbf{A} = D \times I$ is the action set with D denoting the possible directions (roads) to take at a vertex (intersection) (e.g., $D = \{\leftarrow, \uparrow, \rightarrow\}$). This notation is commonly overloaded such that $\mathbf{A}(s)$ returns the set of actions available at state s. Let θ: $V \times D \rightarrow V$ map a vertex (intersection) and an action (direction) to the subsequent vertex following E. Additionally, we assume: (1) the map is expanded to include a 'failure' absorbing vertex $v^f \in V$, with $\theta(v^f, d) = v^f$ for all $d \in D$, and (2) the goal is also absorbing with $\theta(v^g, d) = v^g$ for all $d \in D$.

The state transition function T follows Equation 6.1, and introduces the uncertainty from our TOC POMDP's transfer of control process given by ρ. First, for the human actor λ, the actor transition function T_λ simply follows the map; for state v, action d, and successor v' is:

$$T_\lambda(v, d, v') = \begin{cases} 1, & \text{if } v' = \theta(v, d) \\ 0, & \text{otherwise} \end{cases} . \tag{6.9}$$

Next, for vehicle actor ν, the actor transition function T_ν ensures that: (1) autonomy-capable states follow the path, and (2) non-autonomy-capable states enter the absorbing vertex v^f; for state v, action d, and successor v' is:

$$T_\nu(v,d,v') = \begin{cases} 1, & \text{if } \langle v,v' \rangle \in E_c \wedge v' = \theta(v,d) \\ 1, & \text{if } \langle v,\theta(v,d) \rangle \notin E_c \wedge v' = v^f \\ 0, & \text{otherwise} \end{cases} . \tag{6.10}$$

Finally, for the safely parked vehicle σ, the actor transition function T_σ always self-loop since it is on the side of the road; for state v, action d, and successor v' is:

$$T_\sigma(v,d,v') = \begin{cases} 1, & \text{if } v' = v \\ 0, & \text{otherwise} \end{cases} . \tag{6.11}$$

We now define the control transfer function ρ. Formally, for $\mathbf{s} = \langle v,i \rangle \in \mathbf{S}$ with action $\langle d,\hat{i} \rangle \in \mathbf{A}(s)$, the allotted travel time is $\tau = \lfloor w(\langle v,\theta(v,d) \rangle) \rfloor$. We have a particular TOC POMDP given the intersection v, current controlling actor i, and desired successor actor $\hat{i}$ from optimal SAVE policy $\pi^*(s) = \langle d,\hat{i} \rangle$. We assume the TOC POMDP's $\mathcal{T} = \{1,\ldots,\tau\}$ has units in seconds without loss of generality. Given the solved TOC POMDP, we would like to compute the expected result from transfer of control. Formally, we sample trajectories over the unobserved true state, observations, and resultant action following the TOC POMDP's optimal policy. Due to the structure of the TOC POMDP, this always results in a collapsed belief with a known end result from $\mathcal{E}$. Formally, let $J = \{s_1,\ldots,s_k\}$ be a set of k final 'end result' states from the TOC POMDP's $\mathcal{E}$ which are determined by random state-action-observation trajectories following the TOC POMDP.

In the case with $i = \sigma$, initially the car is safely on the side of the road. Either (1) the car remains on the side of the road, (2) the human TOC succeeds, or (3) the human TOC fails:

$$\rho(s,\hat{i},i') = \begin{cases} 1, & \text{if } \hat{i} \neq \lambda \wedge i' = \sigma \\ \frac{1}{k}\sum_{i=1}^{k}[s_i = \oplus], & \text{if } \hat{i} = \lambda \wedge i' = \lambda \\ \frac{1}{k}\sum_{i=1}^{k}[s_i \neq \oplus], & \text{if } \hat{i} = \lambda \wedge i' = \sigma \\ 0, & \text{otherwise} \end{cases} \tag{6.12}$$

with $[\cdot]$ denoting Iverson brackets.

In the case with $i \neq \sigma$, either the human λ or vehicle ν is the controlling entity. Either (1) no transfer is requested and the actor remains the same, (2) the TOC succeeds to switch actors, (3) the TOC fails to switch actors, or (4) the TOC is aborted and safely pulls over to the side of the road:

$$\rho(\mathbf{s},\hat{i},i')=\begin{cases}1, & \text{if } \hat{i}=i\wedge i'=\hat{i}\\ \frac{1}{k}\sum_{i=1}^{k}[s_i=\oplus], & \text{if } \hat{i}\neq i\wedge i'=\hat{i}\\ \frac{1}{k}\sum_{i=1}^{k}[s_i=\ominus], & \text{if } \hat{i}\neq i\wedge i'=i\\ \frac{1}{k}\sum_{i=1}^{k}[s_i=\oslash], & \text{if } \hat{i}\neq i\wedge i'=\sigma\\ 0, & \text{otherwise}\end{cases}. \tag{6.13}$$

Trivially, in the limit as the number of sampled trajectories grows $k\to\infty$, this converges to the exact probabilities of obtaining each resulting actor, our desired result. This formulation of the expected value with number of samples k enables us to sample a finite number of times and obtain an approximation of ρ in practice.

The cost function C simply measures the time traveling on a road, given its length and speed limit. We assume ties between actions are broken following the autonomy-preferred roads in E_p. In other words, if autonomy is preferred and the current actor is the human λ, then the next action will attempt to transfer to the vehicle ν. For $\mathbf{s}=\langle v,i\rangle$ and $\mathbf{a}=\langle d,\hat{i}\rangle$:

$$C(\mathbf{s},\mathbf{a})=\begin{cases}w(\langle v,\theta(v,d)\rangle), & \text{if } v\neq v^g\\ 0, & \text{otherwise}\end{cases}. \tag{6.14}$$

The goal is $G=\{\langle v^g,\lambda\rangle\}$ and the initial state is $s^0=\langle v^0,\lambda\rangle$. The live states are defined by the actor capability function ψ. Following the problem definition, the human is capable of acting in all states, and the vehicle can be safely on the side of the road in any state. Thus, for a vertex v, $\psi(v)$ is $\{\lambda,\nu,\sigma\}$ if the road v is autonomy-capable, and $\{\lambda,\sigma\}$ otherwise. Trivially, the failure state has no actors ($\psi(v^f)=\emptyset$). This defines L following the live state definition. Formally, we primarily must consider the set E_c which defines the set of edges (roads) on which the vehicle is capable of driving. Formally, for any state $v\in V\setminus\{v^f\}$:

$$\psi(v)=\begin{cases}\{\lambda,\nu,\sigma\}, & \text{if } \exists d\in D \text{ s.t. } \langle v,\theta(v,d)\rangle\in E_c\\ \{\lambda,\sigma\}, & \text{otherwise}\end{cases} \tag{6.15}$$

and $\psi(v^f)=\emptyset$. Therefore, $L=\{\langle v,i\rangle\in\mathbf{S}|i\in\psi(v)\}$.

6.5 Theoretical Analysis

We now integrate all three concepts we have introduced thus far: SAS, TOC POMDP, and SAVE, in order to show that the transfer of control within our stochastic path planning model provably maintains live state guarantees.

Proposition 19. *SAVE satisfies the live state constraints.*

Proof. By definition of the two SAS live state constraints, we must have (1) $G \subseteq L$, and (2) for all $\mathbf{s} \notin L$, $\mathbf{a} \in \mathbf{A}$, and $\mathbf{s}' \in L$ we need $T(\mathbf{s}, \mathbf{a}, \mathbf{s}') = 0$. The first constraint is trivially true by definition of G, since for $\langle v^g, \lambda \rangle$ we have $\lambda \in \psi(v^g)$. The second constraint requires us to examine any $\mathbf{s} \notin L$, which by Equation 6.15 must be a state $\mathbf{s} = \langle v, \nu \rangle$ such that $\forall d \in D$, $\langle v, \theta(v, d) \rangle \notin E_c$. Therefore, we examine the actor transition function T_ν in Equation 6.10. Case 2 captures this exact scenario, and states that $v' = v^f$. By definition $\theta(v^f, d) = v^f$ for all $d \in D$, they are absorbing. Additionally, regardless of any attempt to transfer control, all states $\langle v^f, i \rangle$ are non-live because $\psi(v^f) = \emptyset$ (Equation 6.15). Thus, the second constraint holds as well. $\square$

Next, we establish Lemma 3 which states that belief within the TOC POMDP only remains over the human state; the other state factors are known (Corollary 7). This enables us to ensure that the optimal policy always defines its actions at the true underlying state. To prove this, let $S^H_{tmt_m} = \{s \in S | s = \langle t, h, m, t_m \rangle, h \in \mathcal{H}\}$ be the set of the states which only differ over the human state, given the rest of the factors t, m, and t_m are fixed. Let the true initial state be defined as any $s^0 = \langle t^0, h^0, m^0, t^0_m \rangle$. Let initial belief be $b^0 \in \triangle^n$ such that for all $s \notin S^H_{t^0 m^0 t^0_m}$, $b^0(s) = 0$ (Equation 6.8). Lemma 3 and Corollary 7 are defined for all action-observation histories $\langle a^0, \omega^1, \ldots, a^{\ell-1}, \omega^\ell \rangle$, and all realizable histories $\bar{h} \in \bar{H}(s^0, \ell)$. Here, we only consider $s^\ell \notin \mathcal{E}$, since trivially if $s^\ell \in \mathcal{E}$, then $b^\ell(s^\ell) = 1$.

Lemma 3. *Belief uncertainty within the TOC POMDP is only over the human state factor $\mathcal{H}$; all other factors are known.*

Proof. We must must show for resulting belief $b^h \in \triangle^n$ that $b^h(s) = 0$ for all $s \notin H(t^h, m^h, t^h_m)$ with $s^h = \langle t^h, h^h, m^h, t^h_m \rangle$. To prove that this holds at some horizon ℓ, we must show that this holds for the iterative application of Equation 2.13. Proof by induction on $i \in \{0, \ldots, \ell\}$.

Base Case: Trivially true by Equation 6.8 for b^0.

Induction Step: Assume true for $i-1$, must show for i. First, if $s \notin S^H_{t^{i-1} m^{i-1} t^{i-1}_m}$, then $b^{i-1}(s) = 0$. Next, by Equation 2.13, if $\bar{T}(s, a, s') b^{i-1}(s) = 0$ for all $s \in \bar{S}$, then $b^i(s') = 0$. We know the constraint on s, so by Equations 6.5 and 6.4, we simply check the three valid cases; recall $s' \notin \mathcal{E}$. All cases strictly define the next state's t^i, m^i, and t^i_m, and all are mutually exclusive in this regard. Thus, as each case defines, the non-zero probability (w.r.t. $\mathcal{P}_c$ and $\mathcal{P}_h$) is defined over possible h^i. Finally, $\bar{O}(a,s',\omega)$ follows $\mathcal{P}_o$. Note that for $s' \in \mathcal{E}$, $b(s')=0$ since $\omega = s \in \mathcal{E}$ is the only scenario this may occur. This implies that $s' \in S^H_{t^i m^i t^i_m}$ (following cases in Equations 6.5 and 6.4). Therefore, we have shown that $b^i(s')=0$ for all $s \notin S^H_{t^i m^i t^i_m}$.

By induction, this is true for b^ℓ. $\square$

Corollary 7. *For resulting belief $b^\ell \in \triangle^n$, true resulting state $s^\ell \in \{s \in S | b^\ell(s) > 0\}$.*

Establishing this property allows us to prove that the TOC POMDP never enters the failure state $\ominus$ in Proposition 20. This is a critical requirement in order to prove that SAVE is a strong SAS in Proposition 21.

Proposition 20. *Following a TOC POMDP's optimal policy π^*, for any horizon ℓ, the underlying true state $s^\ell \neq \ominus$.*

Proof. For a TOC POMDP with optimal policy π^*, for all $s^0 \in \bar{S} \setminus \mathcal{E}$, for all horizons h, we must show that the true state $s^h \in \bar{S}$ must be $s^h \neq \ominus$. Let π^* be the optimal policy for the TOC POMDP following value iteration on the full policy tree's beliefs for any horizon ℓ. Let $s^0 \in \bar{S} \setminus \mathcal{E}$ be any valid initial state. Of course, if $s^0 \in \{\oplus, \oslash\}$ then we would immediately have $b^0(s^0)=1$ anyway by Equation 6.8. This case would trivially have $s^\ell = s^0 \neq \ominus$ by Equation 6.5. Next, if $s^0 \notin \mathcal{E}$, then by Equations 6.5 and 6.4 there is only one collection of states which transition to state $\ominus$ (scenario 1, case 4). This case requires $t=0$ and $a \neq \oslash$. By Lemma 3 and Corollary 7, we only must consider the i in which $s^i = \langle 0, h, m, t_m \rangle$. Therefore, it is sufficient to show that $\pi^*(b^i) = \oslash$.

By Equation 2.20, we rewrite $\bar{V}^t(b) = \max_{a \in \bar{A}} x^a_1 + x^a_2$ with:

$$x^a_1 = \sum_{s \in \bar{S}} b^i(s) \bar{R}(s,a) \qquad x^a_2 = \sum_{\omega \in \Omega} \max_{\alpha \in \Gamma^{t-1}} \sum_{s \in \bar{S}} b^i(s) \bar{V}^t_{sa\omega\alpha}$$

For $a = \oslash$, by Lemma 3 and Equation 6.7 we have $\bar{R}(s,\oslash)=0$ since $s \in S^H_{t^i m^i t^i_m}$ yielding $x^{a=\oslash}_1 = 0$ for each of the t belief updates. Given this constraint on s, by Equation 6.5 $\bar{T}(s,\oslash,s')=1$ only if $s' = \oslash$ (case 2), and subsequently by Equation 6.6 $\bar{O}(\oslash,\oslash,\omega)=1$ only if $\omega = \oslash$. Thus, $\bar{V}^t_{sa\omega\alpha}$ equals $\gamma 1 \cdot 1 \cdot \alpha(\oslash)$ for all $\alpha \in \Gamma^{t-1}$. We now have:

$$x^{a=\oslash}_2 = \max_{\alpha \in \Gamma^{t-1}} \sum_{s \in \bar{S}} b^i(s) \gamma \alpha(\oslash) = \gamma \max_{\alpha \in \Gamma^{t-1}} \alpha(\oslash)$$

For $a \neq \oslash$, by Lemma 3 and Equation 6.7 we have $\bar{R}(s,\emptyset) = -\epsilon$ or $\bar{R}(s, a \notin \{\emptyset, \oslash\}) = -\mathcal{C}(h,m,t)$. Both cases are less than 0 so $x^{a=\oslash}_1 > x^{a \neq \oslash}_1$. Lastly, we examine $x^{a \neq \oslash}_2$. Again, since $s \in S^H_{t^i m^i t^i_m}$ there are two possible successor states with $a \neq \oslash$ following Equation 6.5; either $s' = \oplus$ (case 3) with state transition $\bar{T}(s, a \neq \oslash, \oplus) = \mathcal{P}_c(h^i, m^i, t_m^i)$, or $s' = \ominus$ (case 4) with $\bar{T}(s, a \neq \oslash, \ominus) = 1 - \mathcal{P}_c(h^i, m^i, t_m^i)$. Subsequently Equation 6.6 states that $\bar{O}(a \neq \oslash, s', \omega) = 1$ only if $\omega = s' \in \{\oplus, \ominus\}$. Thus, $\bar{V}_{sa\omega\alpha}^t = \gamma(1 \cdot \mathcal{P}_c(h^i, m^i, t_m^i) \cdot \alpha(\oplus) + 1 \cdot (1 - \mathcal{P}_c(h^i, m^i, t_m^i)) \cdot \alpha(\ominus))$ which in turn equals $\gamma(\alpha(\oplus) + \alpha(\ominus))$. Similarly:

$$x_2^{a \neq \oslash} = \gamma \max_{\alpha \in \Gamma^{t-1}} (\alpha(\oplus) + \alpha(\ominus))$$

To compare $x_2^{a=\oslash}$ and $x_2^{a \neq \oslash}$, we recognize that for $s \in \mathcal{E}$, the respective $\alpha(s)$'s are the result of iteratively computing the maximal action in Γ^{t-1} over each of the t Bellman updates. Similar to above, $\bar{T}(s, a, s') = 1$ and $\bar{O}(a, s', \omega) = 1$ only for $s = s' = \omega \in \mathcal{E}$. Upon each Bellman optimality update, the values for states in $\mathcal{E}$ within the optimal α-vector remain the same for any belief point because the reward is unaffected by the action, and each time it compounds the discount factor γ. These absorbing states have rewards 0 for $s \in \{\oplus, \oslash\}$ and $-\mathcal{C}^*$ for $s = \ominus$. With t discounts, we have: $\alpha(\oplus) = \alpha(\oslash) = 0(1 + \gamma + \cdots + \gamma^{t-1}) + \gamma^t \frac{\bar{R}_{min}}{1-\gamma}$ as compared with $\alpha(\ominus) = -\mathcal{C}^*(1 + \gamma + \cdots + \gamma^{t-1}) + \gamma^t \frac{\bar{R}_{min}}{1-\gamma}$.

Since $-\mathcal{C}^* < 0$, for any number of t Bellman optimality equation updates, we have $x_2^{a=\oslash} \geq x_2^{a \neq \oslash}$, implying that $\pi^*(b^i) = \oslash$ for all b^i. Thus, given any horizon ℓ and initial state $s^0 \notin \mathcal{E}$, the resultant state $s^\ell \neq \ominus$ following optimal policy π^*. $\square$

Proposition 21. *SAVE is a strong SAS.*

Proof. Let π^* be the optimal policy for the SAVE. By definition of a strong SAS, we must show that there exists a strong optimal policy π^*. By the definition of a strong policy, we must show that for all $s^0 \in L$ and $\ell \in \mathbb{N}$, for all $\bar{h} \in \bar{H}_{\pi^*}(s^0, \ell)$ and for all $i \in \{0, \ldots, \ell\}$, $s^i \in L$. Additionally, for π^* to be a optimal policy, we must also show that all live states are proper, i.e., the goal is reachable with probability 1. If these states are not proper, then they could not yield an optimal policy π^* in the first place by live state constraint 2, and non-zero cost in Equation 6.14. First we prove that for all $s^0 \in L$, the there exists a proper policy by showing that there exists at least one state-action history which reaches the goal with probability 1. This guarantees there exists an optimal policy π^* by SSP assumption (Section 2) because we know at least one such policy exists; it is impossible for an SSP to have an optimal improper policy when there exists a proper policy. By Equation 6.1 of T, we need to examine T_i and ρ.

First, we consider any $\langle v, \lambda \rangle \in L$. The graph is strongly connected, therefore for any v there exists a directed path to v^g. T_λ deterministically follows the graph via θ; ρ need not be considered if $\hat{i} = i$ at every state (Equation 6.1). So for any $\langle v, \lambda \rangle \in L$ there exists a proper policy.

Second, we consider any $\langle v,\sigma\rangle \in L$. This self-loops, so we examine ρ (Equation 6.12) which transitions to λ with non-zero probability, otherwise it remains with σ in control. As shown above, there exists a proper policy from any $\langle v,\lambda\rangle \in L$, so for all $\langle v,\sigma\rangle \in L$ there exists a proper policy.

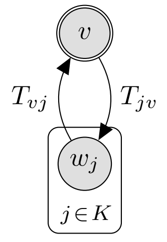
Third, we consider any $\langle v,\nu\rangle \in L$. Since all non-live states (except those with v^f) have controlling entity ν, we only must consider the live states $\langle v,\nu\rangle$ which are (directed) neighbors to these non-live states. We must show the goal can still be reached with probability 1. By definition, a state is live if $\exists d \in D$ such that $\langle v,\theta(v,d)\rangle \in E_c$. Thus, there must exist a $d \in D$ for our live state $\langle v,\nu\rangle$ such that $\langle v,\theta(v,d)\rangle \in E_c$. By Equation 6.10, case 2 will not trigger for this d. Now examine ρ. By Equation 6.13, only case 3 would keep ν as the controlling entity. By Proposition 20, this has probability 0 using our TOC POMDP. Therefore, the action $\langle d,\lambda\rangle$, for example, will always lead to either $\langle \theta(v,d),\lambda\rangle$ or $\langle \theta(v,d),\sigma\rangle$. In both cases, we have shown above that these states have a proper policy. Therefore, for any $\langle v,\nu\rangle \in L$, there exists a proper policy.

All lives states have been shown to have a proper policy, thus the optimal policy π^* exists (i.e., they are not dead ends). By Proposition 19, live state constraints hold. By live state constraint 1, $G \subseteq L$, and by live state constraint 2, for all $s \notin L$, for all $a \in \mathbf{A}$, and $s' \in L$, $T(s,a,s')=0$. Thus, for any policy-realizable history $\bar{h} \in \bar{H}_{\pi^*}(s^0,\ell)$ with $s^0 \in L$, we have for any $i \in \{1,\ldots,\ell\}$, $s^i \in L$ because π^* is proper and all non-live states are dead ends. Therefore, π^* is a strong optimal policy, and so SAVE is a strong SAS. $\square$

6.5.1 Policy Network Representation of SAS

We now prove in Proposition 22 and Figure 6.1 how SAS is representable as a policy network. First, we summarize the relevant components of SAS and then prove this fact. SAS is an example of how policy networks can describe transferring control between models that have differing state and action spaces. It demonstrates how policy networks can be used to effectively break down a large problem with completely different model representations of sub-tasks (i.e., POMDP and SSPs) and maintain a mathematically sound meaning.

Semi-autonomous systems model the transfer of control of a single agent among a group of *actors* I that control it, such as transferring control between an AV and a human driver. It is built on a two-level hierarchy with a SSP [10] reasons about transfer of control success and failure by executing a POMDP. The SAS state space $\mathbf{S} = S \times I$ includes the current actor, and the action space $\mathbf{A} = A \times I$ includes the desired next actor. The state transition $T : \mathbf{S} \times \mathbf{A} \times \mathbf{S} \to [0,1]$ follows the current actor's state transition $T_i : S \times A \times S \to [0,1]$. However, if a transfer is attempted at $\mathbf{s} = \langle s,i\rangle$, we multiply by $\rho : \mathbf{S} \times \mathcal{I} \times \mathcal{I} \to [0,1]$ as $\rho(\mathbf{s},\hat{i},i') = Pr(i'|\mathbf{s},\hat{i})$ denotes the probability that the next actor is i' given an attempt to transfer from i to $\hat{\imath}$. For each state-action pair, ρ is computed by a POMDP in a completely different state and action space that considers communication messages and belief about the state of the actor's preparedness to take control. Its execution ends in a collapsed absorbing belief state: success b^s, failure b^f, or abort b^a. A mapping $f:\{b^s,b^f,b^a\}\rightarrow I$ must be provided from these POMDP result states to the next SSP actor. Given its policy, the probability of reaching these three absorbing states is computed as ρ for use by the SSP.



$$v \sim \mathit{SSP}(\mathbf{S},\mathbf{A},T,C,\mathbf{s}^0,\mathbf{s}^g)$$
$$w_j \sim \mathit{POMDP}(S_j,A_j,\Omega_j,T_j,O_j,R_j)$$

Figure 6.1: SAS represented as a policy network.

Proposition 22. *Policy networks generalize SAS.*

Proof. For any SAS, we must construct an equivalent policy network. See Figure 6.1. Let $V = \{v\} \cup \{w_j\}$ with $v \sim \mathit{SSP}(\cdot)$ and $w_j \sim \mathit{POMDP}(\cdot)$ as in the figure with distinct state and action. Without loss of generality and to remain inline with SAS, an SSP is used, which is akin to an MDP with discount $\gamma = 1$, initial state $\mathbf{s}^0$, and goal state $\mathbf{s}^g$. For each SSP state-action pair there is a distinct POMDP w_j, with $K = \{j \in \mathbf{S} \times \mathbf{A} | j = \langle \mathbf{s},\mathbf{a}\rangle \wedge i \neq \hat{\imath}\}$. Let $\{\langle v,w_j\rangle\} \cup \{\langle w_j,v\rangle\} \subset E$. For T_{vj} and T_{jv}, we have the notation $j = \langle \mathbf{s},\mathbf{a}\rangle$, $\mathbf{s} = \langle s,i\rangle$, $\mathbf{a} = \langle a,\hat{\imath}\rangle$, and $\mathbf{s}' = \langle s',i'\rangle$. First, let $T_{vj}(\mathbf{s},\mathbf{a},b_j^0) = 1$ be defined for any $i \neq \hat{\imath}$, always executing the corresponding POMDP starting at its b_j^0. Second, T_{jv} has three cases: for each $b \in \{b_j^s, b_j^f, b_j^a\}$ we have $T_{jv}(b,a_j,\mathbf{s}') = T_i(s,a,s')[f(b) = i']$. Since the action spaces are different ($\mathbf{A} \neq A_j$), v's CSMDP summarizes the state transitions of each w_j within $\underline{\lambda}$. This is identical to using ρ in Equation 6.1's state transition $T_i(s,a,s')\rho(\mathbf{s},\hat{\imath},i')$ which summarizes the success and failure, and resulting state, for each w_j. Thus, this policy network produces the same Bellman equations for the SAS's SSP and POMDPs. □

6.6 Evaluation

We present a series of trials with subsets of 10 cities' road data from OpenStreetMap (OSM). Distant start and goal addresses are selected as one would do using a global positioning system (GPS)

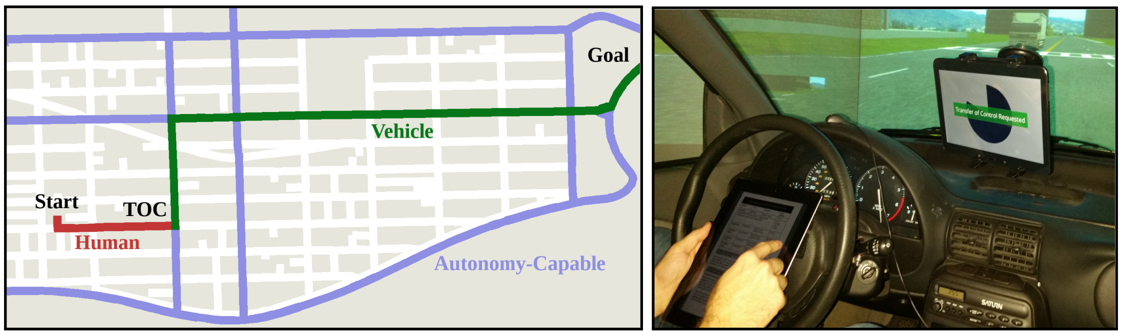


Figure 6.2: Example SAVE policy with TOC in Boston (left) and a TOC POMDP in SAVE simulator (right).

device. All main roads with a speed limit of 30 or greater are marked as autonomy-preferred. We compare our approach with a human driver following the GPS (λ), only the autonomous car (ν), and the collaboration between human and vehicle (λ & ν). We use three metrics for comparison. First, we check if the goal was reachable from the initial road (G). Second, we determine the percentage of time the vehicle drives autonomously, provided it could do so, for roads along its route (%). Third, we record the average travel time to compare efficiency along each of the routes (T).

Our experiments solve the TOC POMDP with PBVI [90] and the SAVE SAS using LAO* [52]. Table 6.1 shows our results for 100 trials for each city. Figure 6.2 depicts a sample collaborative policy in which the human and vehicle gracefully transfer control along the route. The driver is always able to reach the goal, but never drives autonomously, even when the car is capable of doing so. The autonomous vehicle always succeeds in autonomously driving, but is only able to reach the goal in 3 of the 10 scenarios. When it does drive autonomously, it has to take long main roads, causing travel time to be greatly increased. Interestingly, the human and autonomous vehicle collaboration always reaches the goal, and drives autonomously for large portions of the route. Also, the average travel times are relatively similar between the human and collaborative scenarios. This collaborative approach selects routes that properly balance main road autonomous driving and back road human driving.

We also implement our TOC POMDP as depicted in Figure 6.2. It shows a person receiving a request to transfer control while they are engaged in a reading task. Interaction with the POMDP occurs in real-time. Control is transferred from autonomous vehicle to human in the simulated environment in the manner shown. The simulator consists of three large projectors and a real car that is wired into the three computers controlling the experiments.

Domain			Human (λ)			Vehicle (ν)			Team (λ & ν)		
City Name	$\|S\|$	$\|A\|$	G	%	T	G	%	T	G	%	T
Austin	303	12	Y	0	128	N	100	—	Y	13	128
Baltimore	315	12	Y	0	146	Y	100	232	Y	46	154
Boston	912	18	Y	0	136	N	100	—	Y	95	140
Chicago	258	12	Y	0	99	N	100	—	Y	85	142
Denver	348	15	Y	0	128	N	100	—	Y	81	132
Los Angeles	291	12	Y	0	120	N	100	—	Y	42	120
New York City	960	15	Y	0	294	N	100	—	Y	54	313
Pittsburgh	198	12	Y	0	81	N	100	—	Y	8	89
San Francisco	504	18	Y	0	151	Y	100	183	Y	80	174
Seattle	366	12	Y	0	111	Y	100	138	Y	0	111

Table 6.1: Results for SAVE experiments on 10 cities for human driver λ, autonomous vehicle ν, and the human semi-autonomous vehicle collaboration λ & ν drivers. Metrics: goal reachability 'G', autonomous driving percentage '%', and travel time 'T' (seconds).

6.7 Conclusion

We present a hierarchical approach to the transfer of control in semi-autonomous systems, which facilitates efficient planning for a human-agent collaboration. The hierarchical model captures explicitly and optimizes the critical transfer of control process using a POMDP. We show how to apply the general framework to SAS for semi-autonomous vehicles and demonstrate its benefits. Furthermore, we analyze the SAS with TOC model, showing that it maintains live state and thus is a strong SAS. The experiments show that the hierarchical approach is able to leverage the capabilities of the human and agent as it optimizes the desired objective.

Future work will include experiments with humans in a full-scale driving simulator. We will also explore other SAS domains such as assistive technologies (e.g., physical therapy) and disaster response (e.g., search-and-rescue). Finally, we will provide our source code to facilitate the creation of a wide variety of strong semi-autonomous systems.

CHAPTER 7
SCALABLE ONLINE DECISION-MAKING

In this chapter, we present a scalable solution for decision-making that embraces multiple models acting simultaneously as part of an agent. We show how multiple models can each recommend an action and a executive decision-making function decides on the final action to perform. To demonstrate this model, we provide a complete solution for the second-to-second decision-making system within autonomous vehicles, namely for any form of intersections and pedestrians. We implement this solution on a fully-operational autonomous vehicle prototype acting in the real world on public roads and discuss the results.

This model describes how multiple models can simultaneously interact through constraining the available actions. This is a demonstration of policy constraints in a policy network, with each model differing in their state spaces but sharing an action space. This mapping to a policy network is formally defined and discussed in this chapter as well. In general, the proposed scalable online decision-making model is most useful for domains in which multiple problems can be encountered simultaneously and producing a tractable solution requires managing these multiple models in a clean scalable manner.

7.1 Introduction

There has been substantial progress with planning under uncertainty in partially observable, but fully modeled worlds. However, few effective formalisms have been proposed for planning in open worlds with an unspecified, large number of objects. This remains a key challenge for autonomous systems, particularly for *autonomous vehicles* (AVs). AV research has advanced rapidly since the DARPA Grand Challenge [127], which acted as a catalyst for subsequent work on low-level sensing [120, 111] and control [74, 35], as well as high-level route planning [132].

A critical missing component to enable autonomy in *long-term urban deployments* is the *mid-level intersection decision-making* (e.g., the second-to-second stop, yield, edge, or go decisions). As in many robotic domains, the primary challenges include the sheer complexity of real-world problems, wide variety of possible scenarios that can arise, and unbounded number of multi-step problems that will be actually encountered, perhaps simultaneously. These factors have limited the deployment of existing methods for mid-level decision-making [128, 20, 5, 60]. We present a scalable, realistic solution, with strong mathematical foundations, via decomposition into problem-specific decision-components.

Our primary motivation is to provide a *general* solution for AV decision-making at any intersection, including n-way stops, yields, left turns at green traffic lights, right turns at red traffic lights, etc. In this domain, the AV approaches the intersection knowing only the static features from the map, such as road, crosswalk, and traffic controller information. Any number of vehicles and pedestrians can arrive and interact around the intersection, all potentially relevant to decision-making and unknown a priori. The AV must make mid-level decisions, using *very limited hardware* resources, including when to stop, yield, edge forward, or go, based on all possible interactions among all vehicles including the AV itself. Vehicles can be occluded, requiring the use of information gathering actions based on belief over partial observability. Pedestrians can jaywalk, necessitating that motion forward is taken only under strong confidence they will not cross. Uncertainty regarding priority and right-of-way exists, and must be handled under stochastic changes. Vehicles and pedestrians can block one another's motion, and AV-related blocking conflicts must be discovered and resolved via motion-based negotiation.

We provide a general solution for domains concerning *multiple online decision-components with interacting actions* (MODIA). For the particularly difficult AV intersection decision domain, MODIA considers all vehicles and pedestrians as separate individual decision-components. Each component is a partially observable Markov decision process (POMDP) that maintains its own belief for that particular component problem and proposes an action to take at each time step. MODIA then employs an executor function to act as an action aggregator to determine the actual action taken by the AV. This decomposition enables a tractable POMDP solution, benefiting from powerful belief-based reasoning while only growing linearly in the number of encountered problems.

Previous work on an general models related to MODIA include architectures for mobile robots [22, 105] or other systems [33], and contain decision-components that produce actions, aggregated to a system action. They do not, however, naturally model uncertainty or have a general theoretical grounding. Forms of hierarchicies include action-based execution of child problems with multiple options [7] and abstract machines [88]. Action-space partitioning that execute smaller MDPs [53] and POMDPs [89] also exists. These do not model the online execution of an unknown number of decision-components for use in robotics. More application-focused work on action voting for simple POMDPs to solve intractable POMDPs have been used successfully [143]. Robotic applications of hierarchical POMDPs for an intelligent wheelchair decompose the problem into components [124] or with two POMDP levels for vision-based robots—selecting regions of interest and performing sequential vision operations to predict contents [119]. These practical methods work well but lack generalized mathematical foundations. Also, none of these present AV-specific solutions.

Previous work specific to AV decision-making includes simple rule-based or finite-state controller systems [60], which are simple to implement but are brittle, difficult to maintain, and were unable to handle the abundant uncertainty in AV decision-making. Initial attempts using deep neural networks map raw images to control [25] are slow to train and tend to fail rapidly when presented with novel situations. Mixed-observability MDPs for pedestrian avoidance also successfully use a decision-component approach (AV-pedestrian pairs) but provide limited theoretical work and do not extend to intersections [6]. Using a single POMDP for all decision-making has been explored, including continuous POMDPs using raw spacial coordinates for mid-level decision-making [20], online intention-aware POMDPs for pedestrian navigation [5], and POMDPs for lane changes that use online approximate lookahead algorithms [128]. These approaches do not address the exponential complexity concerns (scalability), provide generalizable theoretical foundations, or enable simultaneous seamless integration of multiple different decision-making scenarios on a real AV, all of which are provided by MODIA.

Our primary contributions are: (1) a formal definition of MODIA (Section 7.2); (2) an application of MODIA to AV intersection decision-making (Section 7.3); (3) a rigorous analysis of the complexity and regret-minimization properties (Section 7.4); (4) a formal representation of MODIA as a policy network (Section 7.4.1); and (5) an evaluation of the approach in both simulation and experiments that integrate MODIA within a real AV (Section 7.5).

7.2 Multiple Online Decision-Components with Interacting Actions

We begin with a general problem description that considers a single autonomous agent that encounters any number of decision problems online during execution. This chapter focuses on collections of POMDPs primarily for their general form, self-consistency, and space limitations. It can be generalized to other decision-making models in the natural way. Finally, Figure 7.1 depicts a complete MODIA example for AVs, and is referenced throughout this section for each concept.

7.2.1 Decision-Making with MODIA

The **multiple online decision-components with interacting actions (MODIA)** model describes a realistic single-agent *online* decision-making scenario defined by the tuple $\langle \mathcal{P}, \mathcal{A} \rangle$. $\mathcal{P} =$

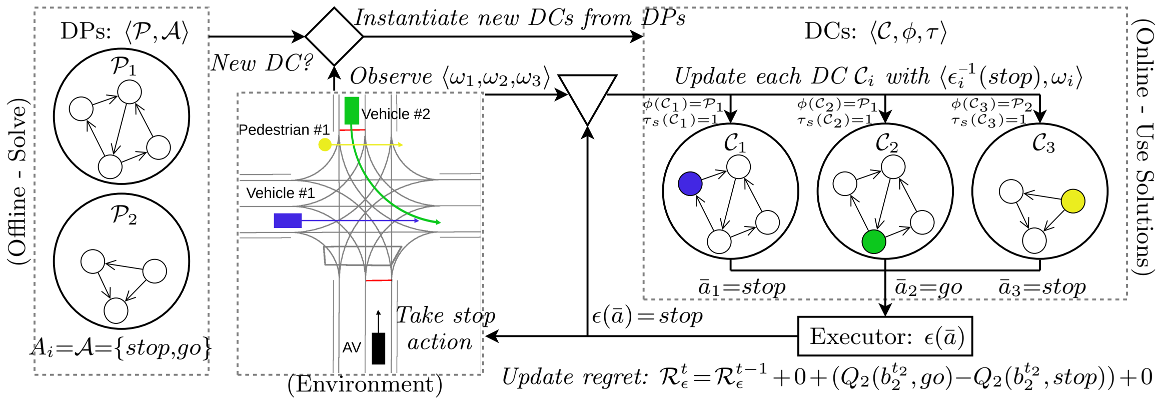

DPs: $\langle\mathcal{P},\mathcal{A}\rangle$
$\mathcal{P}_1$
$\mathcal{P}_2$
(Offline - Solve)
$A_i=\mathcal{A}=\{stop,go\}$
New DC?
Instantiate new DCs from DPs
Observe $\langle\omega_1,\omega_2,\omega_3\rangle$
Pedestrian #1
Vehicle #2
Vehicle #1
AV
Take stop action
(Environment)
DCs: $\langle\mathcal{C},\phi,\tau\rangle$
Update each DC $\mathcal{C}_i$ with $\langle\epsilon_i^{-1}(stop),\omega_i\rangle$
$\phi(\mathcal{C}_1)=\mathcal{P}_1$ $\tau_s(\mathcal{C}_1)=1$
$\phi(\mathcal{C}_2)=\mathcal{P}_1$ $\tau_s(\mathcal{C}_2)=1$
$\phi(\mathcal{C}_3)=\mathcal{P}_2$ $\tau_s(\mathcal{C}_3)=1$
$\mathcal{C}_1$
$\mathcal{C}_2$
$\mathcal{C}_3$
$\bar{a}_1=stop$
$\bar{a}_2=go$
$\bar{a}_3=stop$
(Online - Use Solutions)
$\epsilon(\bar{a})=stop$
Executor: $\epsilon(\bar{a})$
Update regret: $\mathcal{R}_\epsilon^t=\mathcal{R}_\epsilon^{t-1}+0+(Q_2(b_2^{t_2},go)-Q_2(b_2^{t_2},stop))+0$

Figure 7.1: Example visualization of MODIA for AVs. Offline, the DPs (left) are solved: vehicles ($\mathcal{P}_1$) and pedestrians ($\mathcal{P}_2$). Online, the AV approaches an intersection in the environment (center). DCs (right) are instantiated from DPs based on 3 new observations: 2 vehicles ($\mathcal{C}_1$ and $\mathcal{C}_2$) and 1 pedestrian ($\mathcal{C}_3$). Each DC recommends an action ($\bar{a}$): 2 stops and 1 go. The executor decides: stop. The agent takes the action, resulting in regret for $\mathcal{C}_2$'s action in $\mathcal{R}_\epsilon^t$. New observations induce DC updates.

$\{\mathcal{P}_1,\ldots,\mathcal{P}_k\}$ are **decision-problems (DPs)** that could be encountered during execution. For this chapter, each $\mathcal{P}_i \in \mathcal{P}$ is a POMDP with $\mathcal{P}_i = \langle S_i, A_i, \Omega_i, T_i, O_i, R_i\rangle$ starting from an initial belief $b_i^0 \in \triangle^{|S_i|}$. We consider discrete *time steps* $t \in \mathbb{N}$ over the agent's entire lifetime. $\mathcal{A}=\{\mathbf{a}_1,\ldots,\mathbf{a}_z\}$ are z **primary actions** that are the true actions taken by the agent that affect the state of the external system environment. *Importantly, only $\mathcal{P}$ and $\mathcal{A}$ are known offline a priori.*

AV Example Figure 7.1 has two pre-solved intersection decision-components: single vehicle ($\mathcal{P}_1$) or pedestrian ($\mathcal{P}_2$). Each are POMDPs with actions (recommendations) 'stop' or 'go'. Primary actions $\mathcal{A}$ for the AV are also 'stop' or 'go'.

Online, the DPs are *instantiated* based on what the agent experiences in the external system environment. Due to the nature of actually executing multiple decision-making models (e.g., POMDPs) in real applications, there is no complete model for which, when, or how many DPs are instantiated, or even how long they are relevant.

Formally, the online **instantiations** in MODIA are defined by the tuple $\langle\mathcal{C},\phi,\tau\rangle$. Over the agent's lifetime, there are n DP instantiations called **decision-components (DCs)** denoted as $\mathcal{C} = \{\mathcal{C}_1,\ldots,\mathcal{C}_n\}$, with both $\mathcal{C}$ and n *unknown* a priori. Let $\phi : \mathcal{C} \rightarrow \mathcal{P}$ denote the DP for each instantiation. Let $\tau : \mathcal{C} \rightarrow \mathbb{N}\times\mathbb{N}$ be the two time steps that each DC is instantiated and terminated. For notational convenience, for all $\mathcal{C}_i \in \mathcal{C}$, let $\tau_s(\mathcal{C}_i)$ and $\tau_e(\mathcal{C}_i)$ be the start and end times; we have $\tau_s(\mathcal{C}_i) < \tau_e(\mathcal{C}_i)$. Without loss of generality, we also assume for $i<j$, $\tau_s(\mathcal{C}_i) \leq \tau_s(\mathcal{C}_j)$. We call a DC $\mathcal{C}_i \in \mathcal{C}$ **instantiated**

at time step $t \in \mathbb{N}$ if $t \in [\tau_s(\mathcal{C}_i), \tau_e(\mathcal{C}_i)]$. Any instantiated $\mathcal{C}_i \in \mathcal{C}$ includes POMDP $\phi(\mathcal{C}_i)$, its policy $\pi_i : \triangle^{|S_i|} \rightarrow A_i$, and its current belief state $b_i^{t_i} \in \triangle^{|S_i|}$ with *local* POMDP time step $t_i = t - \tau_s(\mathcal{C}_i)$.

AV Example (Continued) Online, the AV encounters an intersection and immediately (at time step 1) observes two vehicles and one pedestrian. Three DCs are instantiated; $\mathcal{C}_1$ and $\mathcal{C}_2$ are for each vehicle ($\phi(\mathcal{C}_1) = \phi(\mathcal{C}_2) = \mathcal{P}_1$), and $\mathcal{C}_3$ is for the pedestrian ($\phi(\mathcal{C}_3) = \mathcal{P}_2$). The start times for all $\mathcal{C}_i$ are $\tau_s(\mathcal{C}_i) = 1$; the end times $\tau_e(\mathcal{C}_i)$ are still unknown. Each POMDP $\mathcal{C}_i$, with $\phi(\mathcal{C}_i) = \mathcal{P}_j$: $b_i^0 = b_j^0$, $t_i = 1$, and $\pi_i = \pi_j$.

7.2.2 The MODIA Executor

With DPs and primary actions $\langle \mathcal{P}, \mathcal{A} \rangle$ (known a priori), and online execution of DCs $\langle \mathcal{C}, \phi, \tau \rangle$ (unknown a priori), the primary actions taken from $\mathcal{A}$ are determined by an action **executor** function $\epsilon : \bar{A} \rightarrow \mathcal{A}$ with $\bar{A} = (\bigcup_i A_i)^*$. (Note: X^* is a Kleene operator on a set X, and A_i is the set of actions for the POMDP from DP $\mathcal{P}_i$.) The executor takes DC action recommendations and converts them to a primary action taken by the agent in the external system environment. It also converts a primary action back to what that decision meant to individual DCs via their action sets. In this chapter, we use the notation $\epsilon^{-1} : \mathcal{A} \rightarrow \bar{A}$ with $\epsilon_i^{-1}(\mathbf{a})$ referring to an individual $\mathcal{C}_i$'s action from POMDP $\phi(\mathcal{C}_i)$ for some $\mathbf{a} \in \mathcal{A}$.

It is important to note the requirement that the executor function ϵ must be able to map any tuple of actions taken from any combination of DPs, with any number of possible duplicates, to a primary action. MODIA is a class of problems that operates without any knowledge about which (or how many) DPs will be instantiated online.

AV Example (Continued) In Figure 7.1, all three DCs produce an action $\langle \bar{a}_1, \bar{a}_2, \bar{a}_3 \rangle = \bar{a} \in \bar{A}$ at each time step. The example states $\bar{a}_1 = \bar{a}_3 = \mathit{stop}$ and $\bar{a}_2 = \mathit{go}$. The executor ϵ decides from $\bar{a}$ that $\mathit{stop} \in \mathcal{A}$ will be the primary action. It informs each DC $\mathcal{C}_i$ what the primary action means to $\mathcal{C}_i$ individually, simply $\epsilon_i^{-1}(\mathit{stop}) = \mathit{stop}$, for belief updates.

7.2.3 The MODIA Objective

The goal of the class of problems captured by MODIA is to design the DPs, primary action set, and executor so that it solves the online real-world problem (e.g., AVs). Prior work on *single*-POMDP online algorithms experimentally analyze their performance with simpler metrics such as average discounted reward (ADR) or run time [73, 144], and richer metrics such as error bound reduction (EBR) or lower bound improvement (LBI) [106]. MODIA is an online *multi*-POMDP model that differs from these previous online single-POMDP solvers. We instead provide a concrete objective function to enable the analysis of this complex online problem within a theoretical context. Our problem domain does not contain a model for how DPs are instantiated as DCs, nor how long DCs remain active. Thus, the *objective* is to *minimize regret* experienced at each step for any given DC instantiations.

Formally, for $\langle \mathcal{P}, \mathcal{A}, \mathcal{C}, \phi, \tau, \epsilon \rangle$, let $h \leq \tau_e(C_n)$ be a horizon, let $I^t = \{i \in \{1, \ldots, n\} | \tau_s(\mathcal{C}_i) \leq t \leq \tau_e(\mathcal{C}_i)\}$ denote the set of indexes for instantiated DCs, and let executor decision $\epsilon(\bar{a}) = \mathbf{a}^t$ at time $t \in \{1, \ldots, h\}$ with primary action $\mathbf{a}^t \in \mathcal{A}$ and the tuple of all instantiated DC's actions $\bar{a} \in \bar{A}$, so for all $i \in I^t$, $\bar{a}_i = \pi_i(b_i^{t_i})$ with π_i, t_i, and $b_i^{t_i}$ from instantiated DC $\mathcal{C}_i \in \mathcal{C}$. The **one-step regret** at time t for all instantiated DCs in I^t is:

$$r_\epsilon^t = \sum_{i \in I^t} Q_i(b_i^{t_i}, \pi_i(b_i^{t_i})) - Q_i(b_i^{t_i}, \epsilon_i^{-1}(\mathbf{a}^t)). \tag{7.1}$$

Informally, a DC's regret in MODIA is the expected reward following the DC's desired policy's action, minus the realized expected reward following the executor's action.

Given this one-step regret, we can consider the total accumulated regret the agent experiences. Ideally, we would minimize this, however, it is not possible to know in most real-world scenarios. The **total regret** over all time is:

$$\mathcal{R}_\epsilon^h = \sum_{t=1}^{h} r_\epsilon^t. \tag{7.2}$$

AV Example (Continued) Executor ϵ selected *stop* $\in \mathcal{A}$, which has $\epsilon_i^{-1}(\mathit{stop}) = \mathit{stop}$ for all $\mathcal{C}_i \in \mathcal{C}$. Following each DC's desired action, only $\mathcal{C}_2$ chose *go* instead. This induces regret equal to $Q_2(b_2^{t_2}, \mathit{go}) - Q_2(b_2^{t_2}, \mathit{stop}) \geq 0$; $\mathcal{C}_1$ and $\mathcal{C}_3$ have 0 regret. $\mathcal{R}_\epsilon^t$ is updated accordingly.

7.2.4 LEAF for MODIA

So far we have described the general form of MODIA using a general executor. Now we examine a particular kind of executor with desirable regret-minimizing properties (shown in Section 7.4). Specifically, we can define a lexicographic preference over the individual actions suggested by each DC. Thus, each DC suggests an action, stored collectively as a tuple of action recommendations, and the executor only executes the best (in terms of preference) action from this set.

A **lexicographic executor action function (LEAF)** has two requirements regarding MODIA's structure in $\langle \mathcal{P}, \mathcal{A} \rangle$. First, let the primary actions $\mathcal{A}$ be factored with the *unique* action sets from among the DPs; formally, $\mathcal{A} = \times_i \Lambda_i$ with $\Lambda = \bigcup_j \{A_j\}$. Second, let $\succ_i$ be a lexicographic ordering over actions in these unique action sets $\Lambda_i \in \Lambda$. If a MODIA satisfies these two requirements, then for all $\bar{a} = \langle \bar{a}_1, \ldots, \bar{a}_x \rangle \in \bar{A}$ and $\mathbf{a} = \langle \mathbf{a}_1, \ldots, \mathbf{a}_y \rangle \in \mathcal{A}$, LEAF $\epsilon(\bar{a}) = \mathbf{a}$ is defined by:

$$\mathbf{a}_i \succ_i \mathbf{a}, \quad \forall \mathbf{a} \in \{\mathbf{a}' \in \Lambda_i | \exists j \text{ s.t. } \bar{a}_j = \mathbf{a}'\} \tag{7.3}$$

for all $\Lambda_i \in \Lambda$, and $\epsilon(\emptyset) = \mathbf{a}$ for some fixed $\mathbf{a} \in \mathcal{A}$. Informally, $\bar{a}$ are the current desired actions from DCs, Λ_i is the unique action set, $\mathbf{a}$ are the resulting actions, and each $\mathbf{a}_i$ (from matching unique action set Λ_i) has the highest preference following $\succ_i$ from the available voted-upon actions. Similarly, the inverse executor extracts the relevant action factor taken by the system and distributes it to all DCs who have that action set; formally, for all $\mathcal{C}_i \in \mathcal{C}$, with $\phi(\mathcal{C}_i) = \mathcal{P}_\ell$, there exists an action $\bar{a}_j \in \Lambda_j = A_\ell$ such that for the primary action taken $\mathbf{a} \in \mathcal{A}$, $\epsilon_i^{-1}(\mathbf{a}) = \bar{a}_j$. In summary, LEAF simply takes the most preferred action among those available.

AV Example (Continued) In the AV example, we have action sets $\{stop, go\} = A_1 = A_2 = \mathcal{A} = \Lambda_1$. Thus, it satisfies the first requirement: primary actions are composed of DP actions. For the second, we define a lexicographic preference $\succ_1$ (encouraging safety) over Λ_1 with $stop \succ go$. Now ϵ in Figure 7.1 is actually LEAF. Namely, the action *stop* is the most preferred action desired among only the actions selected by the DCs. Thus, *stop* is the result of the executor.

7.2.5 Risk-Sensitive MODIA

Now we also consider a specific kind of MODIA, with a form of monotonicity in an ordered relationship over actions and Q-values. Informally, we require DP's Q-values to be monotonic over actions with a penalty for selecting policy-violating high-risk actions. Formally, a MODIA is **risk-sensitive** with respect to a preference $\succ_i$, if for all j, b, a, and a': (1) if $a \succ_i a' \succeq_i \pi_j(b)$ then $Q_j(b,a) \leq Q_j(b,a')$, (2) if $\pi_j(b) \succ_i a$ then $Q_j(b,a) \leq \underline{Q}$ for sufficient penalty $\underline{Q}$.

AV Example (Continued) Action *stop* makes no progress towards the goal while *go* does, so long as *go* is optimal, resulting in (1). Conversely, performing *go* when *stop* is optimal produces a severe expected cost, resulting in (2).

7.3 Application to Autonomous Vehicles

We apply MODIA and LEAF to this concrete problem of AV decision-making at intersections. The formulation expands on the numerous AV examples described in Section 7.2. Due to space considerations, we focus our attention strictly on defining vehicle-related DP (POMDP); however, pedestrian and other DPs follow in a similar manner. Overall, this AV robotic application serves to both ground our theoretical work and simultaneously present an actual solution to intersection decision-making in the real world.

The MODIA AV $\langle \mathcal{P}, \mathcal{A} \rangle$ defines $\mathcal{P}$ by converting *intersection types* (and *pedestrian types*) into POMDP DP. These *types* capture the static abstracted information. For example, intersection types contain features such as the number of road segments, lane information (incoming and outgoing), crosswalk locations, and traffic controller information. A DP is created for all lanes within all intersection types (and pedestrian types). Formally, for each such vehicle and intersection type, we define the DP POMDP $\langle S_i, A_i, \Omega_i, T_i, O_i, R_i \rangle = \mathcal{P}_i \in \mathcal{P}$. $S_i = S_{av}^{\ell} \times S_{av}^{t} \times S_{ov}^{\ell} \times S_{ov}^{t} \times S_{ov}^{b} \times S_{ov}^{p}$ describes the AV's location (approaching/at/edged/inside/goal) and time spent at location (short/long), as well as the other vehicle's location (approaching/at/edged/inside/empty), time spent at location (short/long), blocking (yes/no), and priority at intersection in relation to AV (ahead/behind), respectively. Actions are simply $A_i = \{\textit{stop}, \textit{edge}, \textit{go}\}$, and encode movement by assigning desired velocity and goal points along the AV's trajectory within the intersection. Lower-level nuances in path planning [133] are optimized by other methods. $\Omega_i = \Omega_{av}^{t} \times \Omega_{av}^{b} \times \Omega_{ov}^{t} \times \Omega_{ov}^{b}$ primarily encode the noisy sensor updates in blocking detection (yes/no) but also if the time spent was updated (yes/no) for both the AV and other vehicle. $T_i : S_i \times A_i \times S_i \to [0,1]$ multiply the probabilities of a wide range of situations quantifiable and definable in the state-action space described. This includes multiplying probabilities for: (1) vehicle kindly lets AV have priority, (2) vehicle cuts AV off, (3) AV's success or failure of motion to an abstracted state based on its physical size, (4) a new vehicle arrives at an intersection lane, (6) time increments, (7) vehicle actually stops at stop sign or does a rolling stop, (8) vehicle is blocking the AV's path following the static intersection type's road structure, etc. Additionally, a dead end state (an absorbing non-goal self-loop) is reached when the AV and other vehicle both have state factor "inside" while also "blocking" each other. $O_i : A_i \times S_i \times \Omega_i \to [0,1]$ captures the sensor noise (e.g., determined via calibration and testing of the AV's sensors). This includes successful detections of: (1) other vehicle's crossing of physical locations mapped to abstracted states, (2) determining the blocking probability based on the location of the other vehicle, etc. $R_i : S_i \times A_i \to \mathbb{R}$ is defined as unit cost for all states, except the goal state.

The primary actions are $\mathcal{A} = \{\textit{stop}, \textit{edge}, \textit{go}\}$ and simply describe the AV's movement along the desired trajectory. We define a lexicographic preference $\succ_1$ over this action set $\textit{stop} \succ_1 \textit{edge} \succ_1 \textit{go}$. This preference formalizes the notion that if even one DC said to stop, then the AV should stop. Similarly, if at least one DC said to edge but none said stop, then the AV should cautiously edge forward. Otherwise, the AV should go. This enables us to apply LEAF because $A_i = \mathcal{A}$ for all A_i

(even the pedestrian DPs) and we have lexicographic preference $\succ_1$. Lastly, the defined MODIA produces Q-values that satisfy risk-sensitivity.

7.4 Theoretical Analysis

Given DPs and primary actions $\langle \mathcal{P}, \mathcal{A} \rangle$, MODIA requires the selection of an executor to minimize regret accumulated over time, in addition to solving the DPs themselves. With n unknown a priori, as well as which and when DPs are instantiated as DCs, it is impossible to perform tractable planning techniques entirely offline; again, MODIA is an category of *online* decision-making scenarios. Assume, however, that a prescient oracle provided $\langle \mathcal{C}, \phi, \tau \rangle$ a priori. While this is an impossible scenario, it is useful to understand the worst-case complexity of exploiting this information in the underlying problem of selecting a regret-minimizing executor given this normally unobtainable information. Proposition 23 formally proves this complexity.

Proposition 23. *If $\langle \mathcal{C}, \phi, \tau \rangle$ is known a priori, then the complexity to compute the optimal executor ϵ^* is $O(n^2 zmh)$ with $z = |\mathcal{A}|$, $m = \max_i |A_i|$, and $h = \max_i \tau_e(\mathcal{C}_i)$.*

Proof. Must determine the worst-case complexity to fully define executor $\epsilon^* : \bar{A} \to \mathcal{A}$ to minimize regret Equation 7.2. In the worst-case, we must explore all relevant executors, and compute the regret for each, resulting in the optimal solution.

By the definition of an executor, $\bar{A} = (\bigcup_i A_i)^*$ and $z = |\mathcal{A}|$. Given $n = |\mathcal{C}|$, the maximum *realizable* set size of $\bar{A}$ is all unique potential actions, multiplied by the maximal number of unique DCs instantiated simultaneously. In the worst-case, $A_i \neq A_j$ for all $i \neq j$, so all possible actions must be considered for each; this order bound is $m = \max_i |A_i|$. Also, all combinations of instantiated DCs must be realized, so all $\tau(\mathcal{C}_i) \neq \tau(\mathcal{C}_j)$ for all $i \neq j$. In any order, n births, n deaths, and time no DCs instantiated; thus there are $2n+1$ in total. Hence, the number of potential executors is $O(znm)$.

In the worst-case scenario, $R_i(b_i^{t_i}, \pi_i(b_i^{t_i}))$ differs for every time step for all $\mathcal{C}_i \in \mathcal{C}$. Equation 7.2 requires $O(h \max_t I^t)$ operations. Given $\mathcal{C}$, $h = \max_i \tau_e(\mathcal{C}_i)$. By definition of I^t, $\max_t I^t \leq n$. Thus, the worst-case complexity to compute an optimal ϵ^* is $O(znm) \cdot O(hn) = O(n^2 zmh)$. □

With Proposition 23, we know this impossible oracular scenario's complexity is relatively high, but not exponential. This suggests a method for computing an optimal executor, under more realistic assumptions. Thus, let $\hat{\rho}$ be a given *model* for the hardest feature of MODIA: *online instantiation*. Let $\hat{\rho} : \hat{N}_n \times \hat{T}_n \times \hat{E}_n \times \hat{N}_n \times \hat{T}_n \to [0,1]$ define the probability that a particular set of instantiated DCs $\langle \hat{n}, \hat{\tau} \rangle \in \hat{N}_n \times \hat{T}_n$, and executor selection $\hat{\epsilon} \in \hat{E}_n$, results in a successor DC instantiation state $\langle \hat{n}', \hat{\tau}' \rangle \in \hat{N}_n \times \hat{T}_n$. Here, $\hat{N}_n = \{1, \ldots, k\}^n$ are instantiation indexes (defining ϕ),

$\hat{T}_n = \{\hat{\tau} \in \{\alpha, \{1,\ldots,h\}^2, \omega\}^n | \forall i \in \mathbb{N}, \hat{\tau}_{is} < \hat{\tau}_{ie}\}$ are the instantiation start and end times (defining τ) including non-instantiated α and completed ω demarcations, and $\hat{E}_n = \{\epsilon : \bar{A} \rightarrow \mathcal{A} || \bar{A}| \leq n\}$ are all valid executors (defining ϵ). Additionally, we must assume knowledge of a maximum number of DCs n and horizon h for decidability. Given this model, Proposition 24 proves the resulting MDP's optimal policy minimizes *expected* regret, and that the problem is unfortunately computationally intractable in practice.

Proposition 24. *If n, h, and model $\hat{\rho}$ are known a priori, then: (1) the resulting MDP's optimal policy π^* minimizes expected regret, and (2) its state space is exponential in n and k.*

Proof. We must show the construction of an MDP whose optimal policy minimizes expected regret and show its complexity in the *necessity* of an exponential state space.

Let $\langle \hat{S}, \hat{A}, \hat{T}, \hat{R} \rangle$ be a finite horizon MDP with horizon $\hat{h} = h + 1$. States are $\hat{S} = \{\hat{s}^0\} \cup \hat{E}_n \times \hat{B}_n \times \hat{T}_n$ with $\hat{s}^0$ denoting the initial executor selection state and $\hat{B}_n = \{\hat{B} \in (\bigcup_i \hat{B}_i^h)^* || \hat{B}| = n\}$ be all possible reachable beliefs for $\mathcal{P}_i$ in horizon h (denoted $\hat{B}_i^h$) for all possible instantiations. For notation, we use $\hat{s} = \langle \hat{\epsilon}, \hat{b}, \hat{\tau} \rangle$, each containing instantiated values $\hat{\epsilon}_i$, $\hat{b}_i$, $\hat{\tau}_{si}$, and $\hat{\tau}_{ei}$, as well as $\hat{\theta} : \hat{B}_n \rightarrow \hat{N}_n$ mapping beliefs to their original POMDPs' indices. Actions are executor selection $\hat{A} = \hat{E}_n$. State transitions $\hat{T} : \hat{S} \times \hat{A} \times \hat{S} \rightarrow [0,1]$ have two cases. First, $\hat{T}(\hat{s}^0, \hat{a}, \hat{s}') = [\hat{s}' = \langle \hat{a}, \emptyset, \emptyset \rangle]$ captures executor selection. Second, for $\hat{s} \neq \hat{s}^0$ we have:

$$\begin{aligned}
\hat{T}(\hat{s}, \hat{a}, \hat{s}') = & [(\hat{s} = \hat{s}^0 \wedge \hat{\epsilon}' = \hat{a}) \vee (\hat{s} \neq \hat{s}^0 \wedge \hat{\epsilon}' = \hat{\epsilon})] \\
& \cdot \hat{\rho}(\hat{\theta}(\hat{b}), \hat{\tau}, \hat{\epsilon}, \hat{\theta}(\hat{b}'), \hat{\tau}') \prod_{i=1}^{n} [\hat{b}'_i = b_j^0 \wedge \hat{\tau}_i = \alpha \wedge \hat{\tau}'_i = 1] \\
& \cdot \prod_{i=1}^{n} Pr(\hat{b}'_i | \hat{b}_i, \pi_j(\hat{b}_i)) [\hat{\tau}_i \in \mathbb{N} \wedge \hat{\tau}'_i = \hat{\tau}_i + 1] \prod_{i=1}^{n} [\hat{\tau}'_i = \omega \wedge \hat{b}'_i = \hat{b}_i]
\end{aligned}$$

with $j = \hat{\theta}_i(\hat{b})$. This captures executor state assignment, the instantiation model $\hat{\rho}$, the proper initialization of belief, the belief update for active DCs, and the termination of a DC. Rewards $\hat{R} : \hat{S} \times \hat{A} \rightarrow \mathbb{R}$ describe the negative regret, $\hat{R}(\hat{s}, \hat{a}) = \sum_i Q_j(\hat{b}_i, \hat{\epsilon}_i^{-1}(\mathbf{a}^t)) - Q_j(\hat{b}_i, \pi_j(\hat{b}_i))[\hat{\tau}_i \in \mathbb{N}]$ with $R(\hat{s}^0, \hat{a}) = 0$. By construction, this is MODIA, assuming $\hat{\rho}$, n, and h were provided. By assigning $\epsilon^* = \pi^*(\hat{s}^0)$, we minimize expected regret. In the worst-case, it necessitates modeling all n DC instantiation permutations (with replacement) of the k DPs, which is $O(k^n)$. □

This illustrates the importance of the original MODIA formulation. Even with the instantiation model of Proposition 24, the problem is still unscalable. And the knowledge needed to bound the number of active DCs (e.g., n and h) is generally unavailable a priori. This intrinsic lack of information motivated our formulation that minimizes the regret at each time step. Hence, the agent is guided by the optimal DC policies from each instantiated DP, selecting the regret-minimizing action at each time step. Proposition 25 proves that LEAF minimizes the regret in risk-sensitive MODIA at each time step, enabling a tractable solution to MODIA.

Proposition 25. *If a MODIA is risk-sensitive, then LEAF minimizes regret r_ϵ^t for all t.*

Proof. By definition of regret r_ϵ^t for LEAF ϵ at time step t: $r_\epsilon^t = \sum_i Q_j(b_i^{t_i}, \pi_j(b_i^{t_i})) - Q_j(b_i^{t_i}, \epsilon_i^{-1}(\mathbf{a}^t))$ with $\phi(\mathcal{C}_i) = \mathcal{P}_j$. We must show for all $\tilde{\epsilon}$, $r_\epsilon^t \leq \tilde{r}_{\tilde{\epsilon}}^t$. For readability, hereafter, let $a_i = \epsilon_i^{-1}(\mathbf{a}^t)$, $\tilde{a}_i = \tilde{\epsilon}_i^{-1}(\tilde{\mathbf{a}}^t)$, $a_j^* = \pi_j(b_i^{t_i})$, and $b_i = b_i^{t_i}$. By definition of risk-sensitive, there always exists action a_j^* such that $Q_j(b_i, a_j^*) \geq \underline{Q}$. Thus, it is sufficient to show that for all $i \in I^t$, $Q_j(b_i, a_i) \geq Q_j(b_i, \tilde{a}_i)$, or there exists a $\mathcal{C}_i \in \mathcal{C}$ with $\phi(\mathcal{C}_i) = \mathcal{P}_j$ such that $Q_j(b_i, \tilde{a}_i) \leq \underline{Q}$. By risk-sensitivity and LEAF, consider 3 cases for ϵ and $\tilde{\epsilon}$.

<u>Case 1</u>: $a_i =_x \tilde{a}_i$ for $a_i, \tilde{a}_i \in \Lambda_x = A_j$. Trivially, we have $Q_j(b_i, a_i) = Q_j(b_i, \tilde{a}_i)$.

<u>Case 2</u>: $a_i \succ_x \tilde{a}_i$ has two cases. *Case 2.a*: If $a_i = a_j^*$, then by definition π_j's optimality, for any $\tilde{a}_i \in A_j$, $Q_j(b_i, a_i) = Q_j(b_i, a_j^*) \geq Q_j(b_i, \tilde{a}_i)$. *Case 2.b*: If $a_i \neq a_j^*$, then by LEAF Equation 7.3, $a_i \in \{a \in \Lambda_x | \exists u \text{ s.t. } \bar{a}_u = a\}$. Thus, by definition of $\bar{a} \in \bar{A}$, there exists this $u \neq i$ such that $a_i = a_u = a_v^*$ with $\phi(\mathcal{C}_u) = \mathcal{P}_v$. By risk-sensitivity, $a_v^* = a_u = a_i \succ_x \tilde{a}_i$ that implies $Q_v(b_u, \tilde{a}_i) \leq \underline{Q}$.

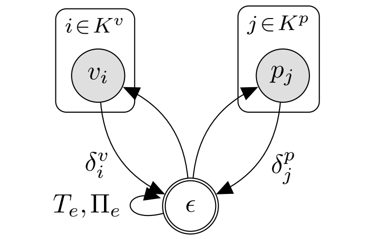
<u>Case 3</u>: $a_i \prec_x \tilde{a}_i$. By definition of risk-sensitivity, we have $\tilde{a}_i \succ_x a_i \succeq_x a_j^*$ and consequently $Q_j(b_i, \tilde{a}_i) \leq Q_j(b_i, a_i)$.

All cases proven. LEAF minimizes regret r_ϵ^t for any t. □

7.4.1 Policy Network Representation of MODIA

We now prove in Proposition 26 and Figure 7.2 how MODIA is representable as a policy network. First, we summarize the relevant components of MODIA and then prove this fact. MODIA is an example of how policy networks can describe non-stationary policy sets, as well as how models with different state spaces and a shared action space can interact. It demonstrates how policy networks can be used to effectively solve a very large problem by decomposition into smaller components.

MODIA, as described above, is for autonomous vehicle decision-making about other entities: vehicles K^v and pedestrians K^p, analyzed a posteriori. Multiple POMDP models (i.e., v_i and p_j) describe each AV-entity pairwise interaction; each model is solved offline in isolate, resulting in stationary polices π_i^v and π_j^p. An *executor* $\epsilon : A^* \rightarrow A$ maps any tuple of action recommendations to a final action performed by the executor. This action updates the other models resulting in regret; e.g., for $i \in K^v$ regret is $r_i = V_i^*(b_i) - Q_i^*(b_i, \epsilon(\mathbf{a}))$. Following Wray *et al.* (2017), we consider risk-sensitive MODIA with LEAF, which assumes an ordering exists over actions $\succ$ in terms of safety. If a riskier action is performed $\pi_i(b_i) \succ \epsilon(\mathbf{a})$ then the model i experiences a regret lower bounded by $r_i \geq Q_i^*(b_i) - \underline{Q}$. A LEAF executor that selects the safest action among the recommendations (i.e., $\forall i, \epsilon(\mathbf{a}) \succeq \pi_i(b_i)$ and $\exists j$ s.t. $\epsilon(\mathbf{a}) = \pi_j(b_j)$) minimizes the sum of one step regrets.



$v_i \sim POMDP(S_i^v, A, \Omega_i^v, T_i^v, O_i^v, R_i^v)$
$p_j \sim POMDP(S_j^p, A, \Omega_j^p, T_j^p, O_j^p, R_j^p)$
$\epsilon \sim \langle \{\pi_\epsilon : \{s_\epsilon\} \to A\}, R_\epsilon \rangle$

Figure 7.2: MODIA represented as a policy network.

Proposition 26. *Policy networks generalize MODIA.*

Proof. For any MODIA, we must construct an equivalent policy network. See Figure 7.2. Let $V = \{\epsilon\} \cup \{v_i\} \cup \{p_j\}$ with $v^0 = \epsilon$. Let each $v_i \sim POMDP(\cdot)$ and $p_j \sim POMDP(\cdot)$ as in the figure with shared A. Let $\epsilon \sim \langle \Pi_\epsilon, R_\epsilon \rangle$ for policies $\pi_\epsilon : S_\epsilon \to A$ with trivial state space $S_\epsilon = \{s_\epsilon\}$. Let $R_\epsilon(s_\epsilon, a)$ equal the index of a in the *reverse* of ordering $\succ$ over A. Let $\{\langle \epsilon, \epsilon \rangle\} \cup \{\langle v_i, \epsilon \rangle, \langle \epsilon, v_i \rangle\} \cup \{\langle p_j, \epsilon \rangle, \langle \epsilon, p_j \rangle\} \subset E$. Let $T_{\epsilon\epsilon}(s_\epsilon, \cdot, s_\epsilon) = 1$ be a self-loop transition and let $\Pi_{\epsilon\epsilon} = \{\pi | V_\epsilon^*(s_\epsilon) = V_\epsilon^\pi(s_\epsilon)\}$ select optimal executor policies. Let Π_ϵ be a non-stationary policy set that is defined by its parents' constraint edges $\Pi_{i\epsilon}^t = \{\pi : \{s_\epsilon\} \to A\} | V_i^*(b_i^t) - Q_i^*(b_i^t, \pi(s_\epsilon)) < \delta_i^v\}$, with a similarly defined $\Pi_{j\epsilon}$, that only allow executor actions with regret no greater than $\delta_i^v = V_i^*(b_i^t) - \underline{Q}$. By construction, the executor vertex ϵ's selected policy $\pi_\epsilon^* \in \Pi_\epsilon^t$ at time t maps its state s_ϵ to the safest action among recommendations. This is identical to LEAF's definition, and thus minimizes the sum of one step regrets implicitly through constraints. □

7.5 Evaluation

We present experiments with an implementation of the proposed MODIA AV formulation. First, we describe experiments on six different intersections in an industry-standard vehicle simulation. Next, we describe four real-world experiments using MODIA on our fully-operational AV prototype vehicle. Finally, we discuss the evaluation's experiments, implementation, and overall conclusions.

Intersection Scenarios					MODIA		Baselines	
Scenario Name	RS	V	P	PI	$\|\mathcal{C}\|$	$\mathcal{M}$	$\mathcal{I}$	$\mathcal{N}$
Crosswalk Pedestrian	4	0	1	1	4	21.1	16.7	30.1
Vehicle & Pedestrian	3	1	1	1	3	16.8	13.6	37.1
Walk & Run Pedestrians	3	1	2	2	6	19.1	13.3	23.3
Multi-Vehicle Interaction	4	2	0	2	5	19.0	13.2	20.9
Bike Crossing	3	0	1	1	3	16.4	13.8	19.8
Jay Walker	4	0	1	1	4	17.7	14.4	24.3

Table 7.1: Results for 6 intersection problems described by the number of road segments (RS), vehicles (V), pedestrians (P), and potential incidents (PI). Metrics: time to complete intersection (seconds) for MODIA AV $\mathcal{M}$ (including the number of DCs $|\mathcal{C}|$), ignorant baseline algorithm $\mathcal{I}$, and naive baseline algorithm $\mathcal{N}$.

7.5.1 Experiments in an Industry-Standard Simulation Environment

We begin with experiments on six different intersections in an industry-standard vehicle simulation developed by Realtime Technologies, Inc. that accurately simulates vehicle dynamics with support for ambient traffic and pedestrians. We evaluated MODIA on real map data at six different intersections, each highlighting a commonly encountered real-world scenario. Table 7.1 describes each scenario by name and provides details regarding the road segments, vehicles, and pedestrians that exist. The number of potential incidents describes how many risks exist, which MODIA perfectly obviates.

We compare a MODIA AV with *ignorant* and *naive* AV baseline algorithms. The ignorant AV follows the law but ignores the existence of all vehicles and pedestrians, acting as if the intersections are empty. The naive AV follows the law and cautiously waits until all others have cleared the intersection beyond 15 meters before attempting to go. These two baselines implement extremes of rule-based AVs [60] and serve as a form of bound for AV behavior to understand MODIA AV's performance.

We evaluate each by their time to complete an intersection, which includes the observations while approaching, decisions at the intersection, and travel within the intersection. In Table 7.1, we observe the MODIA AV successfully completes intersections faster than the cautious naive AV. While the MODIA AV takes longer than the ignorant AV, the ignorant AV encounters each potential incident and the MODIA AV safely avoids them.

7.5.2 Experiments on a Fully Operational AV Prototype

We will now present real experiments with our fully-operational AV prototype which took place in Mountain View and Sunnyvale in California during 2017. These robot experiments illustrate the diverse array of situations that MODIA can handle. Given the complexity of each scenario, and the variety of each sub-problem within each scenario, it also demonstrates the necessity of MODIA and, more generally, policy networks. Specifically, it is simply intractable to attempt to model all scenarios that a long-term autonomous agent can encounter; there are simply too many scenarios each with a distinct state-action space and objective. Instead, breaking the problem into tractable pieces and integrating them seamlessly together results in a truly scalable solution that enables reasoning under uncertainty for long-term autonomy.

3-Way Stop Figures 7.4 and 7.5 depict an interaction with 3 vehicles at a 3-way stop sign intersection on a real public road. The figures show both a high-level route as well as the mid-level MODIA decision-making in two visualizations. In Figure 7.4 (a), the AV begins travelling towards a 3-way stop, initially not detecting any other vehicles, having a low belief over their existence. Figure 7.4 (b), the AV approaches and detects two new vehicles. It determines that at least one is likely to block its desired path. Next, Figure 7.4 (c) shows that the left vehicle decides to traverse the intersection first. Since the AV is only at the stop line, its location state factor enables it to know it can move forward safely. However, in Figure 7.4 (d), we see that the AV has successfully claimed priority in the intersection, but must slow down and edge forward now that it is inside the intersection. Figures 7.4 (e) and (f) show the AV successfully completing the intersection once the path is clear and safe.

4-Way Stop Figure 7.6 depicts an interesting negotiation with 3 vehicles at a 4-way stop sign intersection. It begins as the AV approaches the intersection in Figure 7.6 (a). When it arrives in Figure 7.6 (b), it observes two other vehicles (left and center) which both arrived at the same time as the AV. Uncertain, the AV edges forward slightly, before yielding to the center vehicle as shown in Figure 7.6 (c). Once the center vehicle clears the intersection, Figure 7.6 (d) shows the AV edge forward again to claim priority in the intersection, only to have the left vehicle commit before the AV. This forces the AV to yield again, until this left vehicle has also cleared the intersection. Figures 7.6 (e) and (f) show that once the AV detects the clear intersection, with priority over the new vehicle on the left, it commits to go through the intersection. It safely completes the intersection after this nuanced sequential decision-making and negotiation process.

Partially Observable T-Intersection Figures 7.7 and 7.8 depict an interaction with a partially observable vehicle a T-intersection. Figure 7.7 (a) begins arriving at a T-intersection where the AV slows to a stop in Figure 7.7 (b). The POMDP DCs here have a belief over the existence of vehicles based on observing occlusions or not. In this case, the left road is fully occluded by foliage and a hill. In Figure 7.7 (c) we observe the POMDP DC's uncertain belief about the existence of another vehicle results in the edge action being performed to slowly gain visibility. As it slowly edges forward, the AV's belief shifts after observing an actual vehicle behind the occlusion, as shown in Figure 7.7 (d). Figures 7.7 (e) and (f) show the AV yielding to the vehicle until eventually the road is clear. The AV has enough visibility at this point to have a strong belief that the road is clear, so it moves onto the main road in Figure 7.7 (g).

4-Way Stop With Pedestrian Figure 7.9 depicts an interaction with a vehicle and a pedestrian at a 4-way stop sign intersection. As the AV arrives at the intersection in Figures 7.9 (a) and (b), it detects a vehicle ahead of it and a stationary pedestrian on the opposite crosswalk. It instantiates a vehicle and pedestrian DC accordingly. After a brief wait for the other vehicle to take initiative, the AV believes it has priority and enters the intersection via the go action, as shown in Figure 7.9 (c). Just as the AV advances, however, it detects the pedestrian moving with a high belief that it is blocking the AV's path, as shown in Figure 7.9 (d). Figure 7.9 (e) then depicts the AV's stop action, which slows the AV to stop safely in front of the crosswalk on the other side. This entire time, the other vehicle is yielding to the AV. Once the pedestrian crosses the crosswalk, as shown in Figure 7.9 (f), the AV completes the intersection and allows the other vehicle to also complete the intersection.

7.5.3 Discussion

MODIA is shown to provide a scalable solution. To concretely illustrate this, consider Figure 7.3, which depicts a common 4-way intersection with our fully-operational AV prototype that operates on real public roads and contains an implementation of MODIA and LEAF. This real-world scenario illustrates MODIA's success in addressing scalability concerns while simultaneously handling the nuanced aspects of online decision-making. Each described vehicle DP POMDP has 400 states (265 with additional pruning), with a rich well-structured belief space. In MODIA AVs, the POMDP's size is constant and applies to any intersection. In comparison, a single all-encompassing POMDP with these state factors quickly becomes utterly infeasible, and will vary greatly among intersections. For example, the 4-way stop from Figure 7.3 that only considers the AV and 3 other vehicles

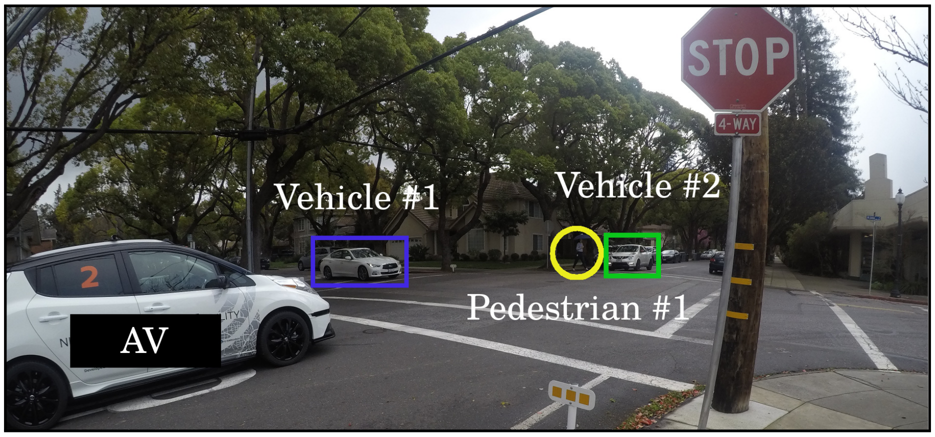


Figure 7.3: Our fully-operational AV prototype at a 4-way stop intersection that implements AV MODIA and LEAF.

(no pedestrians) would the state space $S = S_{av}^{\ell} \times S_{av}^{t} \bigtimes_{i=1}^{3} (S_{ovi}^{\ell} \times S_{ovi}^{t} \times S_{ovi}^{b} \times S_{ovi}^{p})$. This has $|S| =$ 640,000 states, exemplifying notions from Proposition 24. Conversely, MODIA AVs scale *linearly* with the number of vehicles, and would only be 795 states evenly distributed over three POMDPs. On modest hardware, a DP can take < 1 minute to solve using *nova* [136]. Monolithic POMDPs, like the one described, are unequivocally intractable; however, MODIA enables the now realized POMDP solution for AV decision-making.

7.6 Conclusion

MODIA is a principled theoretical model designed for direct practical use in online decision-making for autonomous robots. It has a number advantages over the direct use of a massive monolithic POMDP for planning and learning. Namely, it remains tractable by growing linearly in the number of decision-making problems encountered. Its component-based form simplifies the design and analysis, and enables provable theoretical results for this class of problems. MODIA is shown to successfully solve a challenging AV interaction problem. Future work will explore more executors and models beyond LEAF and risk-sensitive MODIA, develop additional AV-related DPs, and tackle other intractable robotic domains such as humanoid service robots using MODIA as a scalable online decision-making solution.

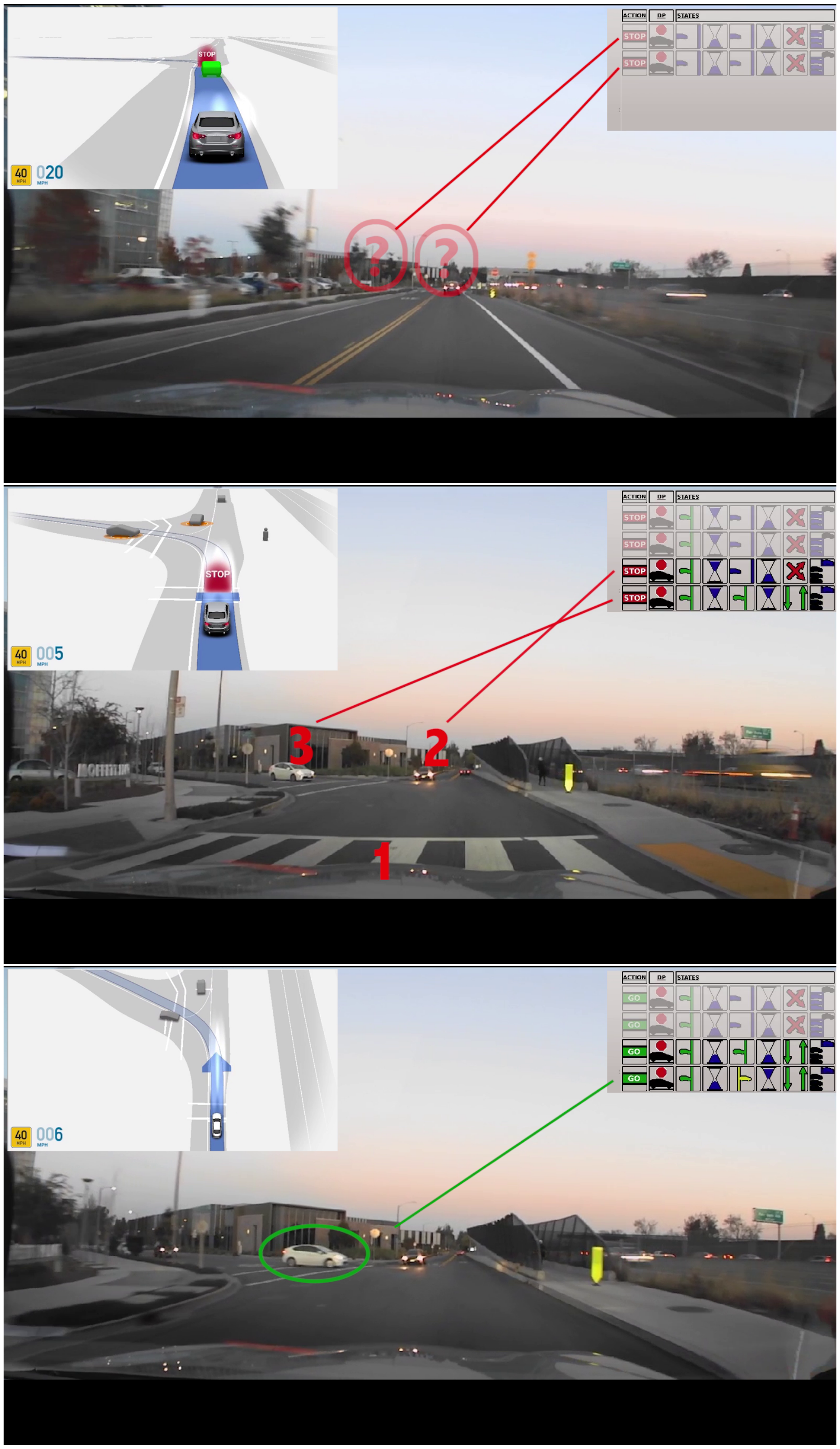


Figure 7.4: Experiment *3-Way Stop* (Part 1 of 2) in Sunnyvale, California during 2017. This is MODIA, as part of a policy network, interacting with real traffic on a public road. The left corner shows a visual of the high-level route plan. The right corner shows a visual of the mid-level MODIA decision-making; each row represents one DC. Images provided courtesy of Nissan Research Center - Silicon Valley.

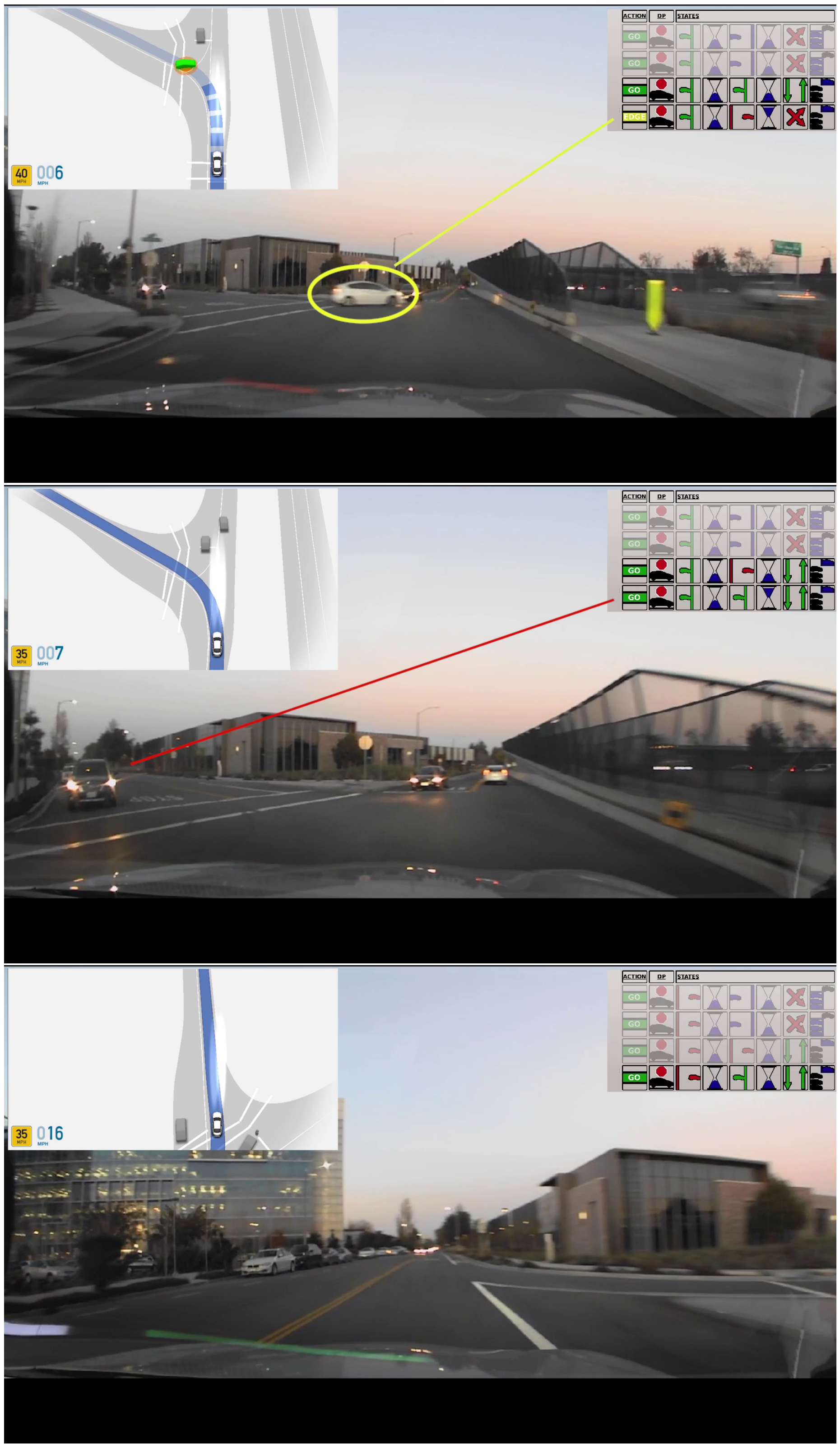


Figure 7.5: Experiment *3-Way Stop* (Part 2 of 2) in Sunnyvale, California during 2017. This is MODIA, as part of a policy network, interacting with real traffic on a public road. The left corner shows a visual of the high-level route plan. The right corner shows a visual of the mid-level MODIA decision-making; each row represents one DC. Images provided courtesy of Nissan Research Center - Silicon Valley.

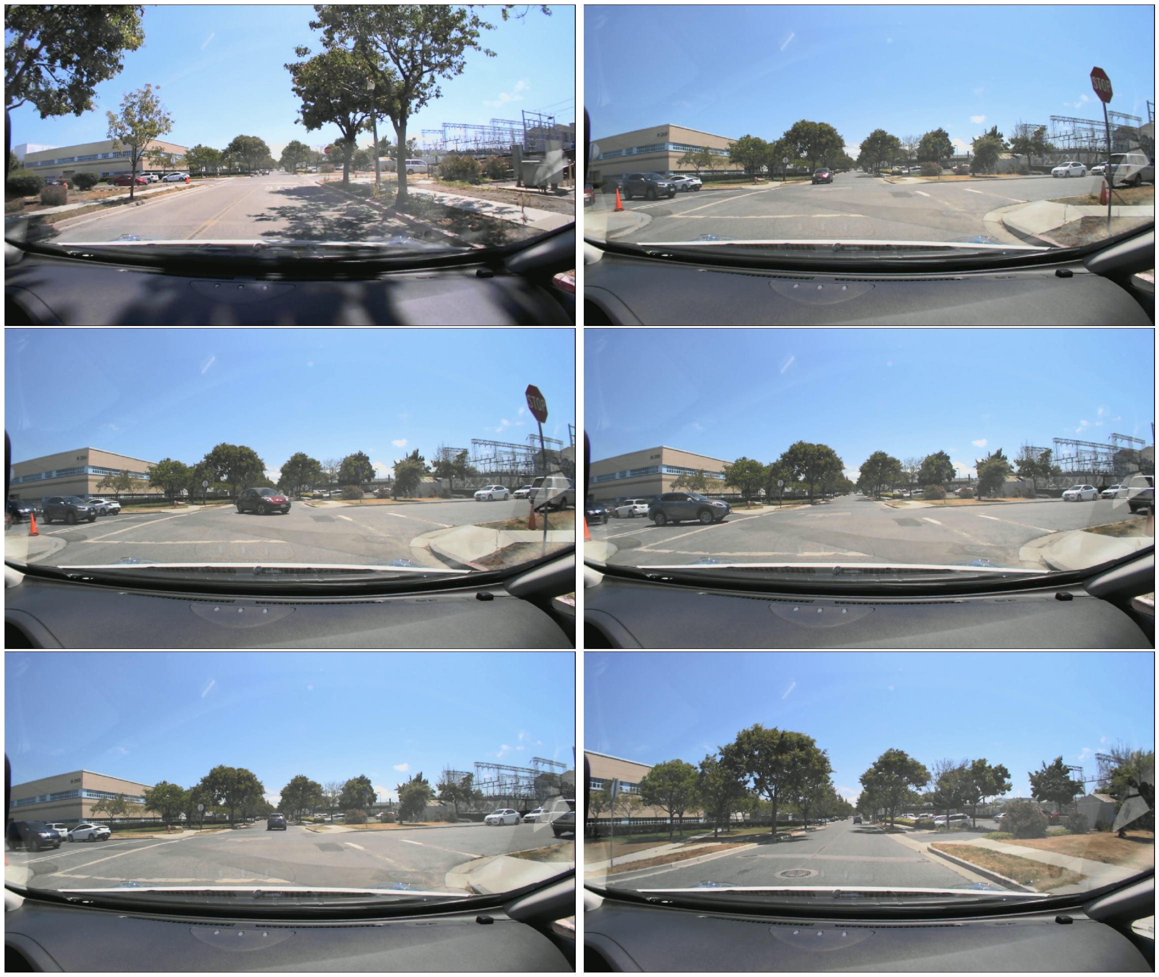


Figure 7.6: Experiment *4-Way Stop* in Mountain View, California during 2017. This is MODIA, as part of a policy network, interacting with real traffic (left vehicles) and one staged researcher (center vehicle). Images provided courtesy of Nissan Research Center - Silicon Valley.

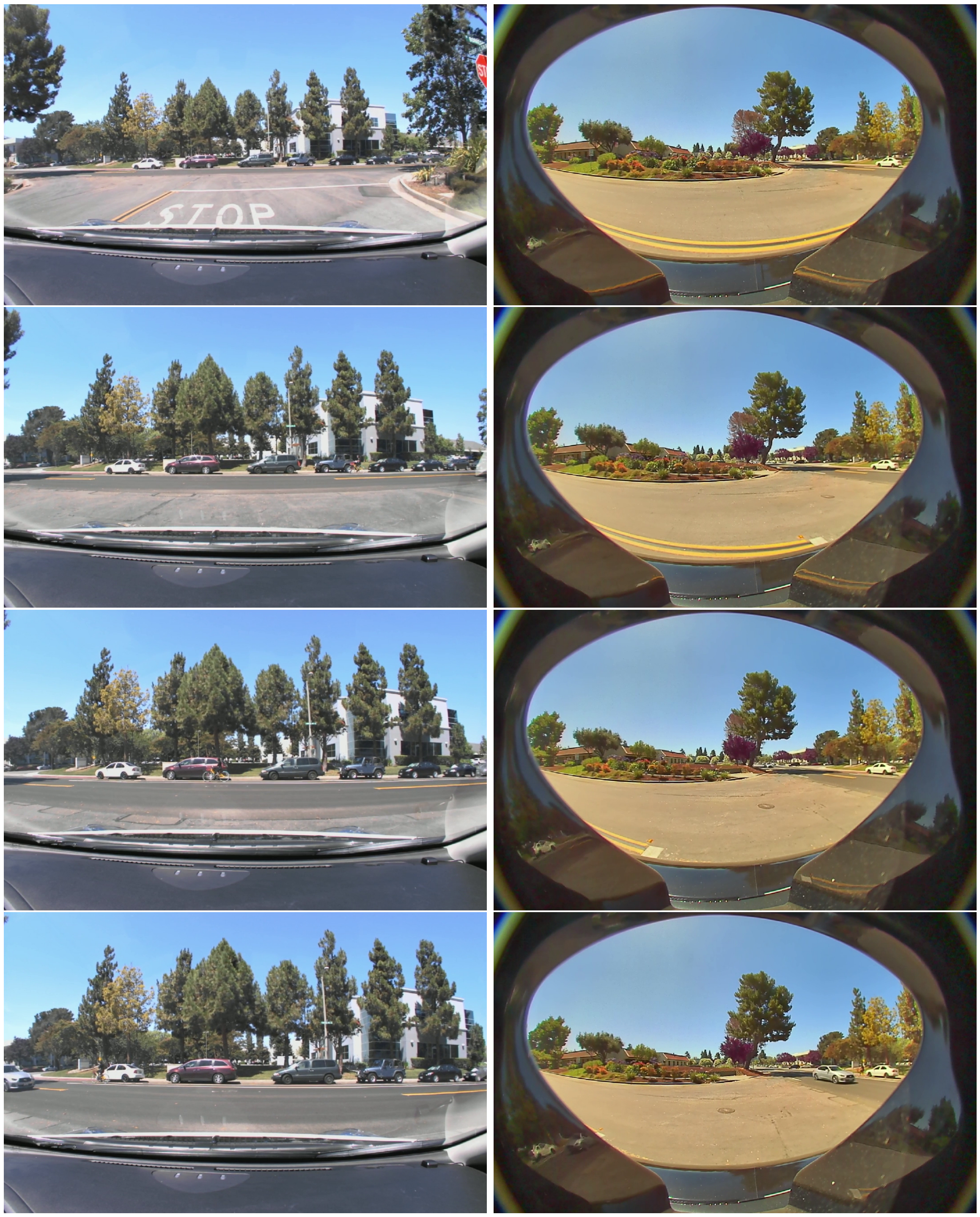


Figure 7.7: Experiment *Partially Observable T-Intersection* (Part 1 of 2) in Sunnyvale, California during 2017. This is MODIA, as part of a policy network, acting on real public roads with one staged researcher (partially observable left vehicle). The left column shows the front view and the right column shows the side view. Images provided courtesy of Nissan Research Center - Silicon Valley.

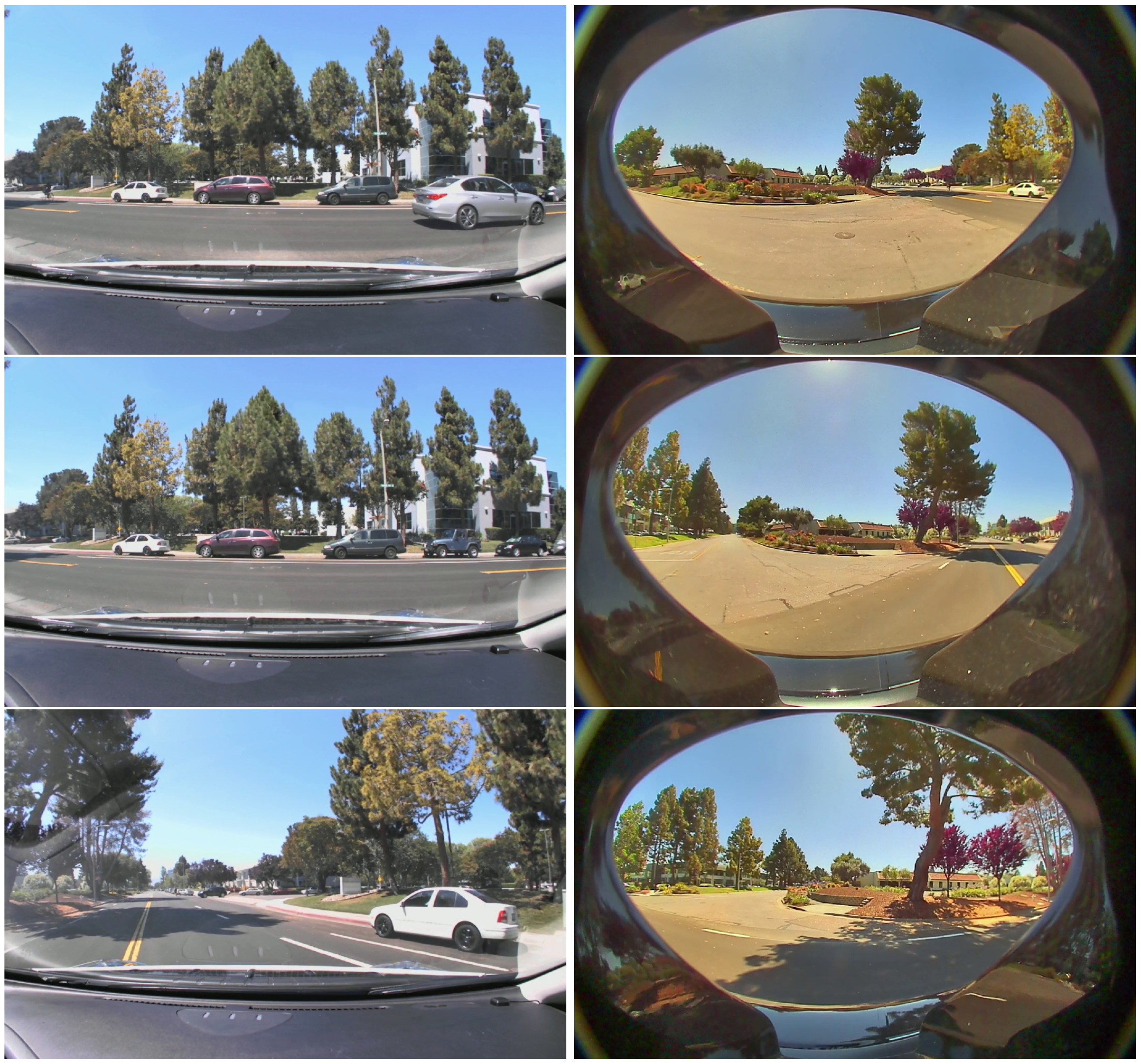


Figure 7.8: Experiment *Partially Observable T-Intersection* (Part 2 of 2) in Sunnyvale, California during 2017. This is MODIA, as part of a policy network, acting on real public roads with one staged researcher (partially observable left vehicle). The left column shows the front view and the right column shows the side view. Images provided courtesy of Nissan Research Center - Silicon Valley.

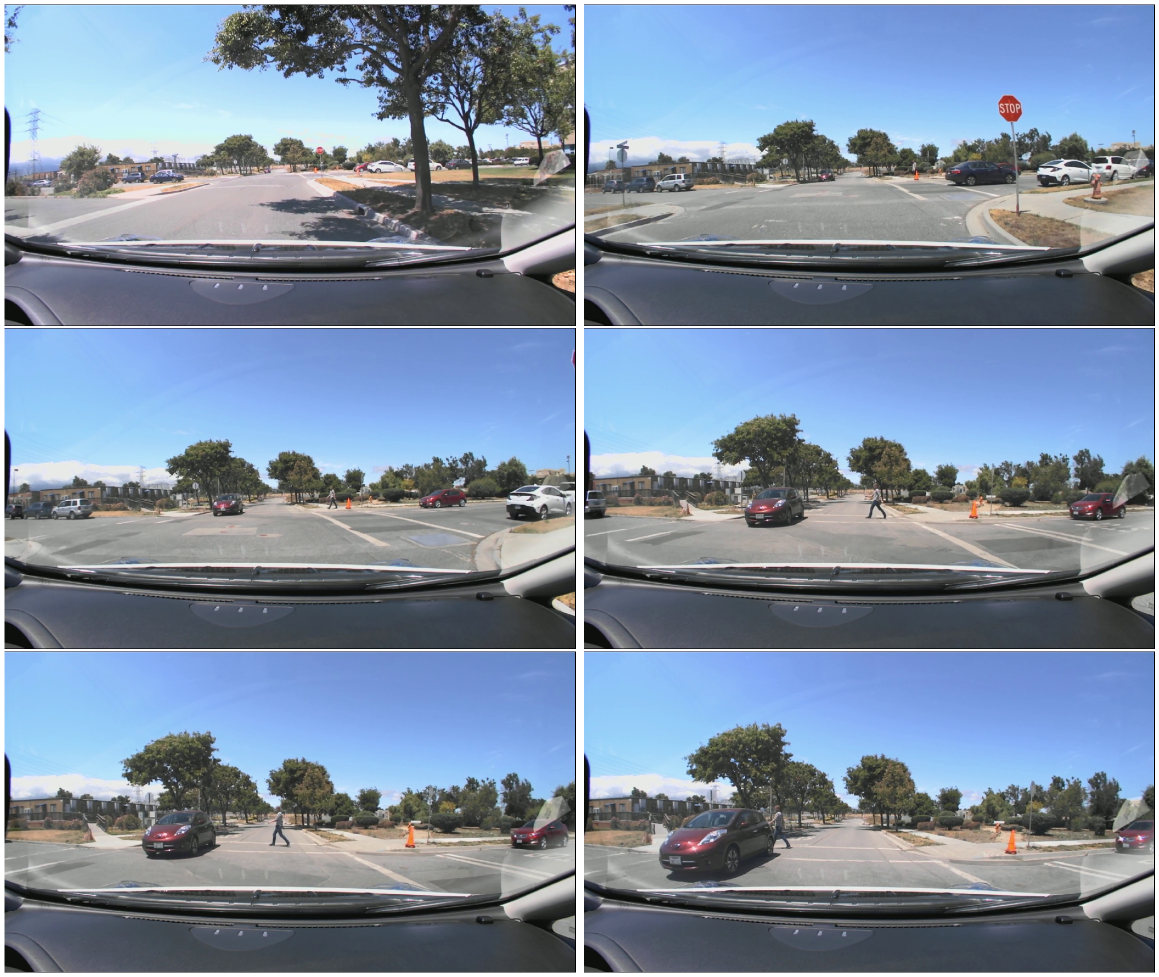


Figure 7.9: Experiment *4-Way Stop With Pedestrian* in Mountain View, California during 2017. This is MODIA, as part of a policy network, interacting with two staged researchers (vehicle and pedestrian). Images provided courtesy of Nissan Research Center - Silicon Valley.

CHAPTER 8
CONCLUSION

The goal of this thesis was to provide scalable mathematical foundations for integrating multiple decision-making components for hierarchical and multi-objective reasoning. In pursuit of this goal, policy networks were proposed as a framework to generalize previously established models such as CMDPs and options, as well as novel models such as TMDPs, SAS, and MODIA presented in this thesis. TMDPs enable the construction of preference orderings across multiple objectives. SAS augments an autonomous agent with the means to transfer control to a human or other agents in order to improve its ability to safely complete its objectives. MODIA allows an unbounded number of decision-making components to be seamlessly integrated online for efficient scalability.

Throughout the thesis, autonomous vehicles (AVs) were used as a common application domain. TMDPs were used to enable the AV to reason about minimizing time and maximizing autonomy in high-level route planning. SAS was used to augment the AV with a human driver reasoning in its high-level route planning, with a mid-level capability to safely transfer control between both. MODIA was used to allow the AV to reason about an unbounded number of vehicles at intersections, lane changes, merges, passing obstacles, and pedestrians for its mid-level decision-making. Formal models were presented for each of the distinct problems. These solutions were evaluated using real-world map data in simulation and/or demonstrated on a fully operational AV prototype driving on real public roads. Policy networks served as a shared underlying framework for all three, enabling their seamless integration as parts of an overall solution for rich, real-world, scalable decision-making in agents with long-term autonomy.

8.1 Future Work

Many promising avenues exist for expanding the models presented in this thesis. Policy networks allow for many interesting interactions among multiple models, both in terms of hierarchies and multiple objectives and in terms of shared and different state-action spaces. The three example models discussed in the chapters provide a template for how to not only build such policy networks. More importantly, they teach how to embrace the thought process required to solve large problems by integrating multiple tractable models together, continuing the traditions set by techniques like options [122] and constrained MDPs [2]. In this way, one could define a general version of the SSP-POMDP hierarchy used in SAS that analyzes integrating SSPs, MDPs, and POMDPs at varying levels of abstraction. Similarly, MODIA could be adjusted to include same and different state-action space DPs, exploring the effect of both. Lastly, all of these approaches can be combined to provide template structures for policy networks that can be combined to enable rapid development.

A multi-agent version of policy networks, and even TMDPs [131], is a natural extension that should be explored. Intuitively, it is quite simple to devise; however, we will leave the details to future work. In brief, first define one controller for each agent. Second, define actions as joint actions, mapping each agent to one of the action factors. Third, require that all agents perform an action at the same time step in order to perform a state transition. Fourth, enforce any such actions performed in a joint action space to be held until all agents perform an action. In the mean time, these held agents are not performing action and so do not induce a state transition or experience rewards. Ideally, this should generalize the Dec-POMDP and partially observable stochastic game (POSG). Initial formalizations of this solution are devised and will be explored in detail once policy networks are established.

Many new application domains can be explored beyond the discussed home healthcare robots and autonomous vehicles solutions. For example, a robot assistant in a work environment that completes a variety of tasks such as delivering packages, giving tours, and generally aiding employees [11]. Another example, warehouse robots must plan and learn in local pick-and-place tasks, local navigation, and long-term navigation [31]. In other words, these domains must seamlessly integrate task, motion, and route planning to facilitate long-term deployments in a real-world setting.

Another important line of future research broadly in long-term autonomy is to devise generalized metrics to evaluate the agent's long-term deployments [54]. Policy networks can begin to provide a natural approach to metrics. Namely, for any given LTA agent using a policy network, we can record the performance of the various component models individually. From this we can measure: (1) the number of times certain components were used or had an impact on policies executed; (2) how did this number change over time if learning occurred; (3) how many times did any transfer of control succeed or fail; (4) how many times did a constraint; (5) did constraints produce their intended result; and (6) what was the effect on components usage and impact after a new model vertex was added or removed? These types of metrics help evaluate an LTA agent over the weeks, months, or even years of its long-term deployment. A formal definition of generalizable metrics in LTA, perhaps levering policy networks in this manner, is a useful contribution to explore.

8.2 Final Thoughts

Autonomous agents are increasingly deployed in the real-world with the potential for great societal impact. Autonomous vehicles represent one of the most impactful domains currently in development, as well as the burgeoning area of home robotics. However, there are still many challenges in low-level control, mid-level decision-making, and high-level route planning that still must be overcome to achieve the goal of their long-term autonomous deployments. Unfortunately, the community's focus recently has been too heavily skewed on the potential of yet-unproven technologies and methods, rather than the grounded perspective needed for designing the actual mathematical models of reasoning and learning that will realistically enable success in our endeavors.

This thesis represents a small contribution to these areas of reasoning in long-term autonomy, in particular for autonomous vehicles. Throughout these chapters, we have covered the challenges in scalably integrating the many decision-making models necessary to build these agents. Solutions were provided for properly describing multi-objective reasoning, designing safe proactive transfer of control in semi-autonomy, and seamlessly fusing models online in a real deployed robot. Many decades of work still remain to formally define, prove, build, and deploy agents with these kinds of integrated models. It is merely a hope that other researchers find value, however small, in the ideas presented in this thesis, as we work in modest but determined steps towards the goal building intelligent agents that fully actualize true long-term autonomy.

BIBLIOGRAPHY

[1] Aissani, Nassima, Beldjilali, Bouziane, and Trentesaux, Damien. Dynamic scheduling of maintenance tasks in the petroleum industry: A reinforcement approach. *Engineering Applications of Artificial Intelligence 22*, 7 (2009), 1089–1103.

[2] Altman, Eitan. *Constrained Markov decision processes.* Chapman & Hall/CRC Press, England, 1999.

[3] Amato, Christopher, Bernstein, Daniel S., and Zilberstein, Shlomo. Optimizing fixed-size stochastic controllers for POMDPs and decentralized POMDPs. *Autonomous Agents and Multi-Agent Systems 21*, 3 (2010), 293–320.

[4] Anderson, Sterling J., Peters, Steven C., Iagnemma, Karl D., and Pilutti, Tom E. A unified approach to semi-autonomous control of passenger vehicles in hazard avoidance scenarios. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics* (2009), pp. 2032–2037.

[5] Bai, Haoyu, Cai, Shaojun, Ye, Nan, Hsu, David, and Lee, Wee Sun. Intention-aware online POMDP planning for autonomous driving in a crowd. In *2015 IEEE International Conference on Robotics and Automation* (2015), pp. 454–460.

[6] Bandyopadhyay, Tirthankar, Won, Kok Sung, Frazzoli, Emilio, Hsu, David, Lee, Wee Sun, and Rus, Daniela. Intention-aware motion planning. In *Proceedings of the 10th Workshop on Algorithmic Foundations of Robotics* (2013), pp. 475–491.

[7] Barto, Andrew G., and Mahadevan, Sridhar. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems 13*, 4 (2003), 341–379.

[8] Bellman, Richard E. *Dynamic Programming.* Princeton University Press, Princeton, NJ, 1957.

[9] Bernstein, Daniel S., Givan, Robert, Immerman, Neil, and Zilberstein, Shlomo. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research 27*, 4 (2002), 819–840.

[10] Bertsekas, Dimitri P., and Tsitsiklis, John N. An analysis of stochastic shortest path problems. *Mathematics of Operations Research 16*, 3 (1991), 580–595.

[11] Biswas, Joydeep, and Veloso, Manuela M. Localization and navigation of the CoBots over long-term deployments. *International Journal of Robotics Research 32*, 14 (2013), 1679–1694.

[12] Blackwell, David. Discrete dynamic programming. *Annals of Mathematical Statistics 33* (1962), 719–726.

[13] Bonet, Blai, and Geffner, Héctor. Planning and control in artificial intelligence: A unifying perspective. *Applied Intelligence 14*, 3 (2001), 237–252.

[14] Bonet, Blai, and Geffner, Héctor. Labeled RTDP: Improving the convergence of real-time dynamic programming. In *Proceedings of the 13th International Conference on Automated Planning and Scheduling* (2003), pp. 12–21.

[15] Boutilier, Craig, Brafman, Ronen I., Domshlak, Carmel, Hoos, Holger H., and Poole, David. CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research 21* (2004), 135–191.

[16] Boutilier, Craig, Dean, Thomas, and Hanks, Steve. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research 11* (1999), 1–94.

[17] Boutilier, Craig, and Poole, David. Computing optimal policies for partially observable decision processes using compact representations. In *Proceedings of 13th National Conference on Artificial Intelligence* (1996), pp. 1168–1175.

[18] Bradtke, Steven J., and Duff, Michael O. Reinforcement learning methods for continuous-time Markov decision problems. In *Proceedings of Advances in Neural Information Processing Systems* (1995), pp. 393–400.

[19] Braziunas, Darius. POMDP solution methods. Tech. rep., University of Toronto, 2003.

[20] Brechtel, Sebastian, Gindele, Tobias, and Dillmann, Rüdiger. Probabilistic decision-making under uncertainty for autonomous driving using continuous POMDPs. In *Proceedings of the 17th International IEEE Conference on Intelligent Transportation Systems* (2014), pp. 392–399.

[21] Broadbent, Elizabeth, Tamagawa, Rie, Kerse, Ngaire, Knock, Brett, Patience, Anna, and MacDonald, Bruce. Retirement home staff and residents' preferences for healthcare robots. In *Proceedings of the 18th IEEE International Symposium on Robot and Human Interactive Communication* (2009), pp. 645–650.

[22] Brooks, Rodney. A robust layered control system for a mobile robot. *IEEE Journal on Robotics and Automation 2*, 1 (1986), 14–23.

[23] Castelletti, Andrea, Pianosi, Francesca, and Soncini-Sessa, Rodolfo. Water reservoir control under economic, social and environmental constraints. *Automatica 44*, 6 (2008), 1595–1607.

[24] Chatterjee, Krishnendu, Majumdar, Rupak, and Henzinger, Thomas A. Markov decision processes with multiple objectives. In *Proceedings of the Annual Symposium on Theoretical Aspects of Computer Science* (2006), pp. 325–336.

[25] Chen, Chenyi, Seff, Ari, Kornhauser, Alain, and Xiao, Jianxiong. DeepDriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the 15th IEEE International Conference on Computer Vision* (2015), pp. 2722–2730.

[26] Cohn, Robert, Durfee, Edmund, and Singh, Satinder. Comparing action-query strategies in semi-autonomous agents. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems* (2011), pp. 1287–1288.

[27] Connell, Jonathan, and Viola, Paul. Cooperative control of a semi-autonomous mobile robot. In *1990 IEEE International Conference on Robotics and Automation* (1990), pp. 1118–1121.

[28] Connolly, Christopher I. Harmonic functions and collision probabilities. *International Journal of Robotics Research 16*, 4 (1997), 497–507.

[29] Connolly, Christopher I., Burns, John B., and Weiss, Rich. Path planning using Laplace's equation. In *1990 IEEE International Conference on Robotics and Automation* (1990), pp. 2102–2106.

[30] Connolly, Christopher I., and Grupen, Roderic A. The applications of harmonic functions to robotics. *Journal of Robotic Systems 10*, 7 (1993), 931–946.

[31] Correll, Nikolaus, Bekris, Kostas E., Berenson, Dmitry, Brock, Oliver, Causo, Albert, Hauser, Kris, Okada, Kei, Rodriguez, Alberto, Romano, Joseph M., and Wurman, Peter R. Analysis and observations from the first amazon picking challenge. *IEEE Transactions on Automation Science and Engineering 15*, 1 (2018), 172–188.

[32] Czyzyk, Joseph, Mesnier, Michael P., and Moré, Jorge J. The NEOS Server. *IEEE Computational Science and Engineering 5*, 3 (July 1998), 68–75.

[33] Decker, Keith. TAEMS: A framework for environment centered analysis & design of coordination mechanisms. *Foundations of Distributed Artificial Intelligence* (1996), 429–448.

[34] Dietterich, Thomas G. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research 13* (2000), 227–303.

[35] Dolgov, Dmitri, Thrun, Sebastian, Montemerlo, Michael, and Diebel, James. Path planning for autonomous vehicles in unknown semi-structured environments. *International Journal of Robotics Research 29*, 5 (2010), 485–501.

[36] Drake, Alvin W. *Observation of a Markov Process Through a Noisy Channel.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1962.

[37] Erol, Kutluhan. *Hierarchical Task Network Planning: Formalization, Analysis, and Implementation.* PhD thesis, University of Maryland, College Park, MD, 1996.

[38] Erol, Kutluhan, Hendler, James, and Nau, Dana S. HTN planning: Complexity and expressivity. In *Proceedings of the 12th National Conference on Artificial Intelligence* (1994), pp. 1123–1128.

[39] Feinberg, Eugene A., and Shwartz, Adam. Constrained Markov decision models with weighted discounted rewards. *Mathematics of Operations Research 20*, 2 (1995), 302–320.

[40] Feyzabadi, Seyedshams, and Carpin, Stefano. Planning using hierarchical constrained Markov decision processes. *Autonomous Robots 41*, 8 (2017), 1589–1607.

[41] Fikes, Richard E., and Nilsson, Nils J. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence 2*, 3/4 (1971), 189–208.

[42] Fong, Terrence, Nourbakhsh, Illah, and Dautenhahn, Kerstin. A survey of socially interactive robots. *Robotics and Autonomous Systems 42*, 3-4 (2003), 143–166.

[43] Gábor, Zoltán, and Szepesvári, Zsolt Kalmárand Csaba. Multi-criteria reinforcement learning. In *Proceedings of the 15th International Conference on Machine Learning* (1998), pp. 197–205.

[44] Gill, Philip E., Murray, Walter, and Saunders, Michael A. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Review 47*, 1 (2005), 99–131.

[45] Givan, Robert, Dean, Thomas, and Greig, Matthew. Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence 147*, 1 (2003), 163–223.

[46] Gonzales, Christophe, Perny, Patrice, and Dubus, Jean-Philippe. Decision making with multiple objectives using GAI networks. *Artificial Intelligence 175*, 7-8 (2011), 1153–1179.

[47] Gopalan, Nakul, desJardins, Marie, Littman, Michael L., MacGlashan, James, Squire, Shawn, Tellex, Stefanie, Winder, John, and Wong, Lawson L.S. Planning with abstract Markov decision processes. In *Proceedings of the International Conference on Automated Planning and Scheduling* (2017), pp. 480–488.

[48] Gordon, Geoffrey J. Stable function approximation in dynamic programming. In *Proceedings of the 12th International Conference on Machine Learning* (1995), pp. 261–268.

[49] Graf, Birgit, Hans, Matthias, and Schraft, Rolf D. Care-o-bot ii—development of a next generation robotic home assistant. *Autonomous Robots 16*, 2 (March 2004), 193–205.

[50] Grosz, Barbara J., and Kraus, Sarit. Collaborative plans for complex group action. *Artificial Intelligence 86*, 2 (1996), 269–357.

[51] Hansen, Eric A. Solving POMDPs by searching in policy space. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence* (1998), pp. 211–219.

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

[53] Hauskrecht, Milos, Meuleau, Nicolas, Kaelbling, Leslie Pack, Dean, Thomas, and Boutilier, Craig. Hierarchical solution of Markov decision processes using macro-actions. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence* (1998), pp. 220–229.

[54] Hawes, Nick, Burbridge, Chris, Jovan, Ferdian, Kunze, Lars, Lacerda, Bruno, Mudrová, Lenka, Young, Jay, Wyatt, Jeremy, Hebesberger, Denise, Körtner, Tobias, Ambrus, Rares, Bore, Nils, Folkesson, John, Jensfelt, Patric, Beyer, Lucas, Hermans, Alexander, Leibe, Bastian, Aldoma, Aitor, Fäulhammer, Thomas, Zillich, Michael, Vincze, Markus, Chinellato, Eris, Al-Omari, Muhannad, Duckworth, Paul, Gatsoulis, Yiannis, Hogg, David C., Cohn, Anthony G., Dondrup, Christian, Fentanes, Jaime Pulido, Krajnik, Tomáš, Santos, João M., Duckett, Tom, and Hanheide, Marc. The strands project. *IEEE Robotics & Automation Magazine 24*, 3 (2017), 146–156.

[55] Howard, Ronald A. *Dynamic Programming and Markov Processes.* Technology Press and Wiley, New York, NY, 1960.

[56] Howard, Ronald A. *Dynamic Probabilistic Systems: Semi-Markov and Decision Processes.* Wiley, New York, NY, 1971.

[57] Isom, Joshua D., Meyn, Sean P., and Braatz, Richard D. Piecewise linear dynamic programming for constrained POMDPs. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence* (2008), pp. 291–296.

[58] Ji, Qiang, Zhu, Zhiwei, and Lan, Peilin. Real-time nonintrusive monitoring and prediction of driver fatigue. *IEEE Transactions on Vehicular Technology 53*, 4 (2004), 1052–1068.

[59] Ji, Shihao, Parr, Ronald, Li, Hui, Liao, Xuejun, and Carin, Lawrence. Point-based policy iteration. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence* (2007), pp. 1243–1249.

[60] Jo, Kichun, Kim, Junsoo, Kim, Dongchul, Jang, Chulhoon, and Sunwoo, Myoungho. Development of autonomous car–Part II: A case study on the implementation of an autonomous driving system based on distributed architecture. *IEEE Transactions on Industrial Electronics 62*, 8 (2015), 5119–5132.

[61] Jung, Cláudio R., and Kelber, Christian R. A lane departure warning system based on a linear-parabolic lane model. In *Proceedings of the IEEE Intelligent Vehicles Symposium* (2004), pp. 891–895.

[62] Kaelbling, Leslie Pack. Hierarchical learning in stochastic domains: Preliminary results. In *Proceedings of the 10th International Conference on Machine Learning* (1993), pp. 167–173.

[63] Kaelbling, Leslie Pack, Littman, Michael L., and Cassandra, Anthony R. Planning and acting in partially observable stochastic domains. *Artificial Intelligence 101*, 1 (1998), 99–134.

[64] Kavraki, Lydia E., Švestka, Petr, Latombe, Jean-Claude, and Overmars, Mark H. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation 12*, 4 (1996), 566–580.

[65] Kim, Dongho, Lee, Jaesong, Kim, Kee-Eung, and Poupart, Pascal. Point-based value iteration for constrained POMDPs. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence* (2011), pp. 1968–1974.

[66] Klein, Laura, young Kwak, Jun, Kavulya, Geoffrey, Jazizadeh, Farrokh, Becerik-Gerber, Burcin, Varakantham, Pradeep, and Tambe, Milind. Coordinating occupant behavior for building energy and comfort management using multi-agent systems. *Automation in Construction 22* (2012), 525–536.

[67] Kochenderfer, Mykel J. *Decision Making Under Uncertainty: Theory and Application.* MIT Press, 2015.

[68] Kolobov, Andrey, Mausam, and Weld, Daniel. A theory of goal-oriented MDPs with dead ends. In *Proceedings of the 28th Annual Conference on Uncertainty in Artificial Intelligence* (2012), pp. 438–447.

[69] Kuffner, James J., and LaValle, Steven M. RRT-connect: An efficient approach to single-query path planning. In *2000 IEEE International Conference on Robotics and Automation* (2000), vol. 2, pp. 995–1001.

[70] Kumar, Akshat, and Zilberstein, Shlomo. History-based controller design and optimization for partially observable MDPs. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling* (2015), pp. 156–164.

[71] Kunze, Lars, Hawes, Nick, Duckett, Tom, Hanheide, Marc, and Krajnik, Tomáš. Artificial intelligence for long-term robot autonomy: A survey. *IEEE Robotics and Automation Letters 3*, 4 (2018), 4023–4030.

[72] Kurniawati, Hanna, Hsu, David, and Lee, Wee Sun. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Robotics: Science and systems* (2008).

[73] Kurniawati, Hanna, and Yadav, Vinay. An online POMDP solver for uncertainty planning in dynamic environment. In *Proceedings of the 16th International Symposium Robotics Research* (2013), pp. 611–629.

[74] Kuwata, Yoshiaki, Teo, Justin, Fiore, Gaston, Karaman, Sertac, Frazzoli, Emilio, and How, Jonathan P. Real-time motion planning with applications to autonomous urban driving. *IEEE Transactions on Control Systems Technology 17*, 5 (2009), 1105–1118.

[75] 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 11th International Conference on Autonomous Agents and Multiagent Systems* (2012), pp. 21–28.

[76] Li, Xin, Cheung, William K.W., Liu, Jiming, and Wu, Zhili. A novel orthogonal NMF-based belief compression for POMDPs. In *Proceedings of the 24th International Conference on Machine Learning* (2007), pp. 537–544.

[77] Littman, Michael L., Dean, Thomas L., and Kaelbling, Leslie Pack. On the complexity of solving Markov decision problems. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence* (1995), pp. 394–402.

[78] Littman, Michael L., Sutton, Richard S., and Singh, Satinder. Predictive representations of state. In *Proceedings of Advances in Neural Information Processing Systems 14* (2002), pp. 1555–1561.

[79] Lovejoy, William S. Computationally feasible bounds for partially observed Markov decision processes. *Operations Research 39*, 1 (1991), 162–175.

[80] Minsky, Marvin. *The Society of Mind.* Simon and Schuster, New York, NY, 1986.

[81] Mitten, L. G. Preference order dynamic programming. *Management Science 21*, 1 (1974), 43–46.

[82] Mouaddib, Abdel-Illah. Multi-objective decision-theoretic path planning. In *2004 IEEE International Conference on Robotics and Automation* (2004), pp. 2814–2819.

[83] Murphy, Kevin P. A survey of POMDP solution techniques. Tech. rep., University of California Berkeley, 2000.

[84] Natarajan, Sriraam, and Tadepalli, Prasad. Dynamic preferences in multi-criteria reinforcement learning. In *Proceedings of the 22nd International Conference on Machine Learning* (2005), pp. 601–608.

[85] Nissan Motor Company Ltd. Together we ride. http://youtu.be/9zZ2h2MRCe0, April 2016.

[86] Papadimitriou, Christos H., and Tsitsiklis, John N. The complexity of Markov decision processes. *Mathematics of Operations Research 12*, 3 (1987), 441–450.

[87] Parr, Ronald. *Hierarchical Control and Learning for Markov Decision Processes.* PhD thesis, Princeton University, Princeton, NJ, 1998.

[88] Parr, Ronald, and Russell, Stuart J. Reinforcement learning with hierarchies of machines. In *Proceedings of Advances in Neural Information Processing Systems* (1998), pp. 1043–1049.

[89] Pineau, Joelle, Gordon, Geoff, and Thrun, Sebastian. Policy-contingent abstraction for robust robot control. In *Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence* (2002), pp. 477–484.

[90] Pineau, Joelle, Gordon, Geoffrey, and Thrun, Sebastian. Anytime point-based approximations for large POMDPs. *Journal of Artificial Intelligence Research 27* (2006), 335–380.

[91] Pineau, Joelle, Montemerlo, Michael, Pollack, Martha, Roy, Nicholas, and Thrun, Sebastian. Towards robotic assistants in nursing homes: Challenges and results. *Robotics and Autonomous Systems 42*, 3 (2003), 271–281.

[92] Pineda, Luis, Wray, Kyle Hollins, and Zilberstein, Shlomo. Revisiting multi-objective MDPs with relaxed lexicographic preferences. In *Proceedings of the AAAI Fall Symposium on Sequential Decision Making for Intelligent Agents* (2015), pp. 63–68.

[93] Piunovskiy, Alexei B., and Mao, Xuerong. Constrained Markovian decision processes: the dynamic programming approach. *Operations Research Letters 27*, 3 (2000), 119–126.

[94] Poupart, Pascal. *Exploiting Structure to Efficiently Solve Large Scale Partially Observable Markov Decision Processes.* PhD thesis, University of Toronto, Canada, 2005.

[95] Poupart, Pascal, and Boutilier, Craig. Bounded finite state controllers. In *Proceedings of Advances in Neural Information Processing Systems 16* (2004), pp. 823–830.

[96] Poupart, Pascal, Malhotra, Aarti, Pei, Pei, Kim, Kee-Eung, Goh, Bongseok, and Bowling, Michael. Approximate linear programming for constrained partially observable Markov decision processes. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence* (2015), pp. 3342–3348.

[97] Pradhan, Anuj Kumar, Hammel, Kim R., DeRamus, Rosa, Pollatsek, Alexander, Noyce, David A., and Fisher, Donald L. Using eye movements to evaluate effects of driver age on risk perception in a driving simulator. *Human Factors 47*, 4 (2005), 840–852.

[98] Puterman, Martin L. *Markov decision processes: Discrete stochastic dynamic programming.* John Wiley & Sons, New York, NY, 1994.

[99] Rajamani, Rajesh, and Zhu, Chunlin. Semi-autonomous adaptive cruise control systems. *IEEE Transactions on Vehicular Technology 51*, 5 (2002), 1186–1192.

[100] Rangcheng, Jia, Yuanyao, Ding, and Shaoxiang, Tang. The discounted multi-objective Markov decision model with incomplete state observations: lexicographically order criteria. *Mathematical Methods of Operations Research 54*, 3 (2001), 439–443.

[101] Åström, Karl J. Optimal control of markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications 10*, 1 (1965), 174–205.

[102] Robinson, Hayley, MacDonald, Bruce, and Broadbent, Elizabeth. The role of healthcare robots for older people at home: A review. *International Journal of Social Robotics 6*, 4 (November 2014), 575–591.

[103] Roijers, Diederik M., Vamplew, Peter, Whiteson, Shimon, and Dazeley, Richard. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research 48* (2013), 67–113.

[104] Roijers, Diederik M., Whiteson, Shimon, and Oliehoek, Frans A. Linear support for multi-objective coordination graphs. In *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-Agent Systems* (2014), pp. 1297–1304.

[105] Rosenblatt, Julio K. DAMN: A distributed architecture for mobile navigation. *Journal of Experimental & Theoretical Artificial Intelligence 9*, 2-3 (1997), 339–360.

[106] Ross, Stéphane, Pineau, Joelle, Paquet, Sébastien, and Chaib-draa, Brahim. Online planning algorithms for POMDPs. *Journal of Artificial Intelligence Research 32*, 1 (2008), 663–704.

[107] Roy, Nicholas, Gordon, Geoffrey, and Thrun, Sebastian. Finding approximate POMDP solutions through belief compression. *Journal of Artificial Intelligence Research 23* (2005), 1–40.

[108] Sacerdoti, Earl D. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence 5*, 2 (1974), 115–135.

[109] Sacerdoti, Earl D. A structure for plans and behavior. Tech. rep., Artificial Intelligence Center, SRI International, Menlo Park, CA, 1975.

[110] Shani, Guy, Pineau, Joelle, and Kaplow, Robert. A survey of point-based POMDP solvers. *Autonomous Agents and Multi-Agent Systems* (2013), 1–51.

[111] Sivaraman, Sayanan, and Trivedi, Mohan M. Looking at vehicles on the road: A survey of vision-based vehicle detection, tracking, and behavior analysis. *IEEE Transactions on Intelligent Transportation Systems 14*, 4 (2013), 1773–1795.

[112] Smallwood, Richard D., and Sondik, Edward J. The optimal control of partially observable Markov processes over a finite horizon. *Operations Research 21*, 5 (1973), 1071–1088.

[113] Smith, Trey, and Simmons, Reid. Point-based POMDP algorithms: Improved analysis and implementation. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence* (2005), pp. 542–549.

[114] Sobel, Matthew J. Ordinal dynamic programming. *Management Science 21*, 9 (1975), 967–975.

[115] Soh, Harold, and Demiris, Yiannis. Evolving policies for multi-reward partially observable Markov decision processes (MR-POMDPs). In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation* (2011), pp. 713–720.

[116] Soh, Harold, and Demiris, Yiannis. Multi-reward policies for medical applications: Anthrax attacks and smart wheelchairs. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation* (2011), pp. 471–478.

[117] Sondik, Edward J. The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs. *Operations Research 26*, 2 (1978), 282–304.

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

[119] Sridharan, Mohan, Wyatt, Jeremy, and Dearden, Richard. Planning to see: A hierarchical approach to planning visual actions on a robot using POMDPs. *Artificial Intelligence 174*, 11 (2010), 704–725.

[120] Sun, Zehang, Bebis, George, and Miller, Ronald. On-road vehicle detection: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence 28*, 5 (2006), 694–711.

[121] Sutton, Richard S., and Barto, Andrew G. *Reinforcement learning: An introduction.* MIT Press, 1998.

[122] 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.

[123] Tambe, Milind. Towards flexible teamwork. *Journal of Artificial Intelligence Research 7* (1997), 83–124.

[124] Tao, Yong, Wang, Tianmiao, Wei, Hongxing, and Chen, Diansheng. A behavior control method based on hierarchical POMDP for intelligent wheelchair. In *Proceedings of IEEE/ASME International Conference on Advanced Intelligent Mechatronics* (2009), pp. 893–898.

[125] Tate, Austin. Generating project networks. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence* (1977), pp. 888–893.

[126] Theocharous, Georgios, and Mahadevan, Sridhar. Compressing POMDPs using locality preserving non-negative matrix factorization. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence* (2010), pp. 1147–1152.

[127] Thrun, Sebastian, Montemerlo, Mike, Dahlkamp, Hendrik, Stavens, David, Aron, Andrei, Diebel, James, Fong, Philip, Gale, John, Halpenny, Morgan, Hoffmann, Gabriel, et al. Stanley: The robot that won the DARPA Grand Challenge. *Journal of Field Robotics 23*, 9 (2006), 661–692.

[128] Ulbrich, Simon, and Maurer, Markus. Probabilistic online POMDP decision making for lane changes in fully automated driving. In *Proceedings of the 16th International IEEE Conference on Intelligent Transportation Systems* (2013), pp. 2063–2067.

[129] Vamplew, Peter, Dazeley, Richard, Berry, Adam, Issabekov, Rustam, and Dekker, Evan. Empirical evaluation methods for multiobjective reinforcement learning algorithms. *Machine Learning 84*, 1-2 (2011), 51–80.

[130] Wang, Zhuoran, Crook, Paul A., Tang, Wenshuo, and Lemon, Oliver. On the linear belief compression of POMDPs: A re-examination of current methods. *arXiv preprint arXiv:1508.00986* (2015).

[131] Wray, Kyle Hollins, Kumar, Akshat, and Zilberstein, Shlomo. Integrated cooperation and competition in multi-agent decision-making. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence* (2018), pp. 4751–4758.

[132] Wray, Kyle Hollins, Pineda, Luis, and Zilberstein, Shlomo. Hierarchical approach to transfer of control in semi-autonomous systems. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence* (2016), pp. 517–523.

[133] Wray, Kyle Hollins, Ruiken, Dirk, Grupen, Roderic A., and Zilberstein, Shlomo. Log-space harmonic function path planning. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2016), pp. 1511–1516.

[134] Wray, Kyle Hollins, Witwicki, Stefan J., and Zilberstein, Shlomo. Online decision-making for scalable autonomous systems. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence* (2017), pp. 4768–4774.

[135] Wray, Kyle Hollins, and Zilberstein, Shlomo. Multi-objective POMDPs with lexicographic reward preferences. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence* (2015), pp. 1719–1725.

[136] Wray, Kyle Hollins, and Zilberstein, Shlomo. A parallel point-based POMDP algorithm leveraging GPUs. In *Proceedings of the 2015 AAAI Fall Symposium on Sequential Decision Making for Intelligent Agents* (2015), pp. 95–96.

[137] Wray, Kyle Hollins, and Zilberstein, Shlomo. A POMDP formulation of proactive learning. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence* (2016), pp. 3202–3208.

[138] Wray, Kyle Hollins, and Zilberstein, Shlomo. Approximating reachable belief points in POMDPs. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2017), pp. 117–122.

[139] Wray, Kyle Hollins, and Zilberstein, Shlomo. Policy networks for reasoning in long-term autonomy. In *Proceedings of the AAAI Fall Symposium on Reasoning and Learning in Real-World Systems for Long-Term Autonomy* (2018), pp. 103–110.

[140] Wray, Kyle Hollins, and Zilberstein, Shlomo. Generalized controllers in POMDP decision-making. In *2019 IEEE International Conference on Robotics and Automation* (2019). To appear.

[141] Wray, Kyle Hollins, and Zilberstein, Shlomo. Policy networks: A framework for scalable integration of multiple decision-making models (extended abstract). In *Proceedings of the 18th International Conference on Autonomous Agents and Multiagent Systems* (2019). To appear.

[142] Wray, Kyle Hollins, Zilberstein, Shlomo, and Mouaddib, Abdel-Illah. Multi-objective MDPs with conditional lexicographic reward preferences. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence* (2015), pp. 3418–3424.

[143] Yadav, Amulya, Marcolino, Leandro S., Rice, Eric, Petering, Robin, Winetrobe, Hailey, Rhoades, Harmony, Tambe, Milind, and Carmichael, Heather. Preventing HIV spread in homeless populations using PSINET. In *Proceedings of the 27th Conference on Innovative Applications of Artificial Intelligence* (2015), pp. 4006–4011.

[144] Ye, Nan, Somani, Adhiraj, Hsu, David, and Lee, Wee Sun. DESPOT: Online POMDP planning with regularization. *Journal of Artificial Intelligence Research 58* (2017), 231–266.

[145] Zhang, Nevin L., and Zhang, Weihong. Speeding up the convergence of value iteration in partially observable Markov decision processes. *Journal of Artificial Intelligence Research 14* (2001), 29–51.

[146] Zilberstein, Shlomo. Building strong semi-autonomous systems. In *Proceedings of the 29th Conference on Artificial Intelligence* (2015), pp. 4088–4092.

[147] Zilberstein, Shlomo, Washington, Richard, Bernstein, Daniel S., and Mouaddib, Abdel-Illah. Decision-theoretic control of planetary rovers. In *International Seminar on Advances in Plan-Based Control of Robotic Agents* (2002), pp. 270–289.