Title
Learned Human-in-the-Loop Decision Making

Author
Basso, Brandon

Publication Date
2012

Peer reviewed|Thesis/dissertation

Learned Human-in-the-Loop Decision Making

by

Brandon Basso

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Engineering - Mechanical Engineering

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor J. Karl Hedrick, Chair
Professor Hugh Durrant-Whyte
Professor David Auslander
Professor Pieter Abbeel

Fall 2012

Learned Human-in-the-Loop Decision Making

Abstract

Learned Human-in-the-Loop Decision Making

by

Brandon Basso

Doctor of Philosophy in Engineering - Mechanical Engineering

University of California, Berkeley

Professor J. Karl Hedrick, Chair

Human beings are decision making engines. From low level automatic activities, such as walking, to high-level reasoning, we readily solve complex perception and decision problems in real time. On the other end of the spectrum, computers and robots have proven particularly useful in dull, dirty, and dangerous domains—inspecting nuclear disaster sites, solving vast optimization problems, and generally accomplishing tasks difficult for humans. Recent advances in robotics and computer science have only served to underscore the difference between humans and robots, namely that robots tend to excel at lower level structured tasks, while humans are unmatched at higher level decision making, particularly when problems involve unstructured data, soft constraints, and ambiguous objectives.

This thesis explores decision problems that lie in the gap between current robot capability and human ability. A generic framework for representing large-scale decision problems, referred to as the Generalized Planning Problem (GPP), is defined. A learning-based GPP solution method is presented that captures multiple stochastic problem elements and soft constraints, which arise in many real world problems. An example routing problem is presented to showcase the expressiveness of the GPP framework. To leverage human intuition with such problems, the GPP framework allows for incorporating human input in a structured way as an external reward signal. Adding human feedback into the learning process results in a variably autonomous decision making engine that can vary continuously between fully autonomous, fully manual, and everywhere in between.

The only way to be truly satisfied is to do what you believe is great work,
and the only way to great work is to love what you do.

If you haven't found it yet, keep looking, and don't settle.

As with all matters of the heart you'll know when you find it,
and like any great relationship, it just gets better and better as the years roll on.

So keep looking, don't settle.

- Steve Jobs

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

ACFR Australian Centre for Field Robotics. 8

AP Assignment Problem. 8, 13, 14, 16, 17, 22, 24–26, 88

C_3UV Center For Collaborative Control of Unmanned Vehicles. 3, 8

CVRP Capacity-constrained VRP. 21, 26

DP Decision Problem. 13, 14

DVRP Distance-constrained VRP. 21, 25

ET Eligibility Trace. 52

FSP Flow Shop Scheduling Problem. 28–30

GAP Generalized Assignment Problem. 14, 16, 22, 32, 116

GPP Generalized Planning Problem. 10–12, 32–34, 36, 67–69, 71, 75, 76, 80, 81, 85, 87–92, 111, 116–118, 120, 123, 126, 128, 129, 136, 138–143

HAM Hierarchy of Abstract Machines. 79, 80

HHMM Hierarchical Hidden Markov Model. 76, 78, 81

HMI Human-Machine Interaction. 2, 9–11

HMM Hidden Markov Model. 41, 76, 78

HRI Human-Robot Interaction. 122

JSP Job Shop Scheduling Problem. 12, 16, 27–30, 75, 128

MC Monte Carlo. 37, 43–45, 52, 53

MDP Markov Decision Process. 10, 37–41, 45, 48, 49, 53, 70, 75–77, 79–85, 87, 89, 105, 111–113, 116, 117, 124, 125, 140

mTSP Multiple Traveling Salesman Problem. 19, 22, 24

NP-hard Non-polynomial hard. 14, 19, 28, 31, 117

OSSP Open Shop Scheduling Problem. 28, 29

POMDP Partially Observable Markov Decision Process. 41, 70, 80

PVRP Prioritized VRP. 21

Q(λ) Q(λ) off-policy temporal difference learning algorithm. 53, 58, 63, 67, 91, 129

SARSA(λ) SARSA(λ) on-policy temporal difference learning algorithm. 91

SMDP Semi-Markov Decision Process. 10, 70, 76, 78, 80–90, 105, 111–113, 116, 117, 140, 142

SMDP-Q SMDP Q-Learning off-policy temporal difference algorithm. 81, 86, 87, 105, 112–114, 117, 129

SVRP Stochastic VRP. 23, 24

TD Temporal Difference. 44–46, 52, 53, 85, 91

TMDP Time-dependent Markov Decision Process. 84

TSP Traveling Salesman Problem. 8, 10, 16–24, 28, 70, 107

UAS Unmanned Autonomous System. 4

UAV Unmanned Aerial Vehicle. 3, 8

UGV Unmanned Ground Vehicle. 8

VRP Vehicle Routing Problem. 8, 10–12, 16, 19–25, 28–30, 33, 70, 75, 76, 88, 92, 94–98, 100–103, 106–108, 117, 120, 126, 129, 137, 138, 141

VRPPD VRP with Pickup and Delivery. 21, 34, 88, 126, 128, 129, 136, 140, 141, 145

VRPSC VRP with Stochastic Customers. 23, 25, 26

VRPSD VRP with Stochastic Demand. 23, 25

VRPST VRP with Stochastic Travel Time. 23, 25

VRPTW VRP with Time Windows. 21, 30

Acknowledgments

I would like to thank everyone who helped me get here. First, my parents, who always supported my education: summer reading lists, scrutinized homework, and an attention to detail that can only be found in the appendices of advanced parenting handbooks—it was all worth it, I'm a doctor!

Thanks Luc Frechette, my first academic adviser, who took me on as a freshman research assistant. I'm pretty sure that I didn't get any real research done, but I learned what real research looks like, and learned to enjoy it.

Thank you Karl Hedrick, my graduate adviser of six plus years. I may have taken an extra year or so to graduate, but I enjoyed all of it. A large part of that was due to my labmates in C_3UV—I can not imagine spending multiple summers in the hot Paso Robles sun, barbecuing massive amounts of tri-tip, and flying very expensive robot airplanes with any other crew. Jack, Allison, Xiao, Steve, and everyone else, thanks all of you.

Not many grad students have the opportunity to take a sabbatical, but I did at the Australian Centre for Field Robotics, and it was outstanding. Thanks Professor Hedrick for helping me set it up and thanks Hugh Durrant-Whyte for taking me on. I not only found my thesis topic at the ACFR, I learned how to do quality robotics research, and learned the difference between good work and great work. Again, the people made the experience for me, from the the men of the cube next door, Ari, Ash, Lachlan, John, and Dan, to the Orca guys, Alex M., Alex B., and Tobi. I learned a ton from you guys, and thoroughly enjoyed all events academic, Newtown, and Zanzibar related.

This thesis is thanks to some of the world's greatest teaching and advising. The fact it even exists is thanks to great friends making grad school worth the time. Rob, you found me an excellent first apartment, despite the clown art. Deena and Zoe, you kept me sane while teaching 8-9 Monday-Thursday my first semester. I will think of you every time my coffee tastes salty. Chris and Athena, thanks for saving me from Berkeley Craigslist roulette; I don't expect to ever have better housemates. Tom, your company and occasional support in the form of food, clothing, or BART fare on morning bike rides made for some seriously happy and seriously cold memories. The only colder one was with Dr. Scott Harrington on Mt. Shasta, where we collectively had just enough brain power to convert -40 Fahrenheit to Celsius. Rob, Scott and Tom, thanks for making every Type 3 trip at least a Type 2, and often a Type 1.

Thanks everyone for helping me accomplish all I did in grad school, it wouldn't have been the same without you. The last six years have been unduplicateable.

Chapter 1

Introduction

Human beings are decision-making engines. From low level automatic activities, such as walking, to high-level reasoning, our brains readily solve complex perception and decision problems in real time. On the other end of the spectrum, computers and robots have proven particularly useful in dull, dirty, and dangerous domains—storing and processing large amounts of data, inspecting disaster sites, and generally accomplishing tasks unsuitable for humans. Recent advances in robotics and computer science have underscored the difference between humans and computers, namely, that computers tend to exceed at lower level structured tasks, while humans remain unmatched at higher level decision making, particularly when problems involve unstructured data, soft constraints, randomness, and ambiguous objectives. Figure 1.1 captures the human-machine spectrum.

This thesis addresses gap between current computation capability and human ability. The overall goal is to advance the field of autonomous decision making by learning from and leveraging human aptitude in high-level decision making.

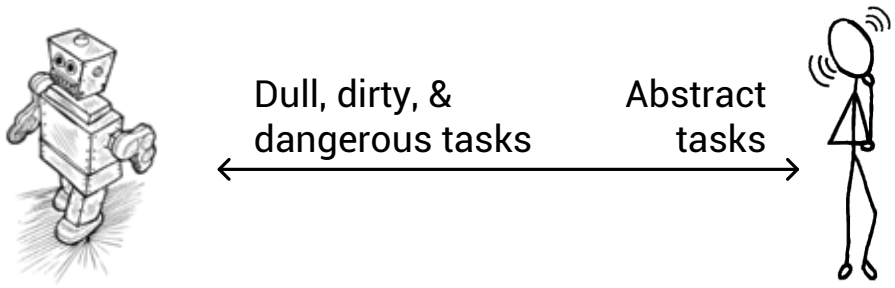


Figure 1.1: The spectrum of tasks that are suited for computers and robots versus humans.

Humans tend to excel at abstract and high-level tasks, whereas computers excel at highly structured data and processing heavy tasks. Oftentimes robots are better suited than humans at performing tasks that are either repetitive and boring (dull), undesirable in some respect (dirty), or potentially life-threatening (dangerous).

Robotics as a field has benefited from advances in computer science and optimization. Solving vast and complex problems has become possible with better hardware and better algorithms. However, high-level tasks in fields such as visual recognition, assignment and scheduling, and generic decision making remain difficult to even pose as optimization problems. Yet it is exactly these problems that humans seem to excel at over their automated counterparts. The core of the human-computer ability gap can be attributed to two human cognitive traits: contextualization and abstraction. Our innate ability to contextualize events (*I am driving on the highway, therefore pedestrians on the road are an unlikely obstacle*) and abstract observations (*The gait of this pedestrian leads me to think they are not running fast enough to intersect my course*) about the world sits squarely in the middle of the human-computer gap.

Inspired by how humans learn through repetition, the focus of this thesis is on solutions to high-level decision problems through reinforcement learning, which closely mirrors human experiential learning. Posing generic decision problems in a learning framework is considered in three sub-problems: *representation*, *modeling and learning*, and *Human-Machine Interaction (HMI)*. Representation addresses the underlying problem parameterization and how the problem is reasoned about in general. Modeling places the problem in a specific formalism for solution, which is learning based in this case. The final subproblem, HMI, addresses methods for incorporating human feedback into the learned solution.

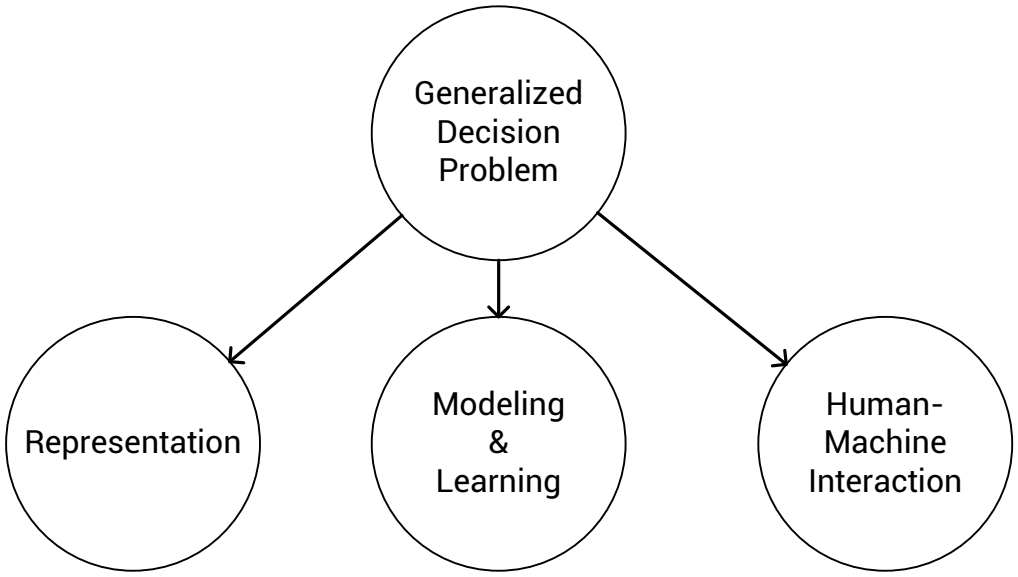


Figure 1.2: Top-level thesis organization.

1.1 Motivation

This research has been motivated by field robotics experiments carried out by the Center For Collaborative Control of Unmanned Vehicles (C_3UV) at UC Berkeley. Large-scale demonstrations performed between 2006 and 2010 involved tasking a heterogeneous fleet of Unmanned Aerial Vehicles (UAVs) to accomplish certain tasks, such as patrolling areas and detecting intrusions [23, 60, 58, 59, 57]. An example scenario is depicted in Figure 1.3. The fixed-wing aircraft used in the scenario, shown in Figure 1.4, collectively possess a variety of capabilities in terms of operational altitude, endurance, sensor configuration, and processing power.

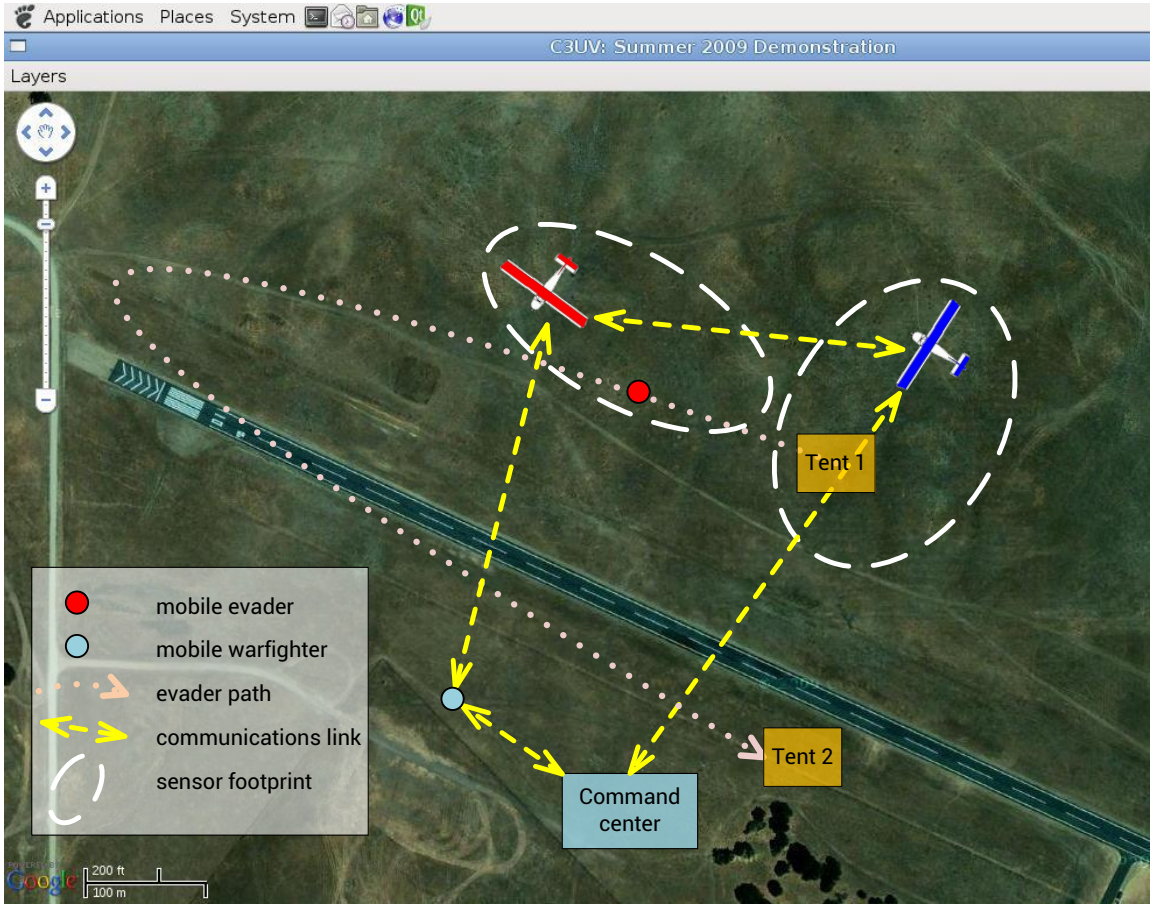


Figure 1.3: A multi-agent UAV mission planning scenario.

A human operator in the Command Center tasks the aircraft with watching the area around Tent 1 for pedestrian activity. Upon detection of a pedestrian leaving Tent 1, a new task is automatically spawned to track the pedestrian. The underlying allocation problem—which aircraft should take each task—motivates this research in high-level decision making with teams of mobile agents, dynamic scenarios, and human operators.

(a) Foam zagi (b) Sig Rascal 110 (c) MLB Bat III

Figure 1.4: Three fixed-wing aircraft operated by the C_3UV lab at UC Berkeley.

The C_3UV Unmanned Autonomous System (UAS) is designed specifically to demonstrate the benefits of autonomy. However, many existing military UASs, incorporating ground, sea, or air robots, operate on the paradigm of many (operators) to one (autonomous agent). The C_3UV UAS reverses the paradigm with many agents to one operator. The enabling technology for such a system is a robust decision making engine, capable of handling multiple human and robotic agents with explicit treatment of uncertainty at the agent, task, and environment level. Several desirable properties such a hybrid human-autonomous system are as follows:

High-Level Human operators often want to address autonomous systems at a high level of abstraction, specifying what needs to be accomplished in terms of objectives and constraints, but not necessarily enumerating how it should be performed, or who should do it.

Rational Autonomous systems are only useful insofar as they appear to make rational decisions. Human operators are quick to distrust and entirely abandon a system that does not appear to be making rational decisions.

Transparent In many cases, explaining *why* the system made a certain choice can prevent an operator from abandoning computer-generated decisions, insofar as the decisions appear rational after explanation.

Adjustable Based on feedback, operators oftentimes would like to make adjustments in real-time to computer-generated decisions, either because they are dissatisfied with the generated solution, or because they would like to change the problem objectives or constraints.

These observations on the interaction between human operators and autonomous systems motivate research into human-in-the-loop autonomous decision making. The focus of this thesis is on high-level autonomous decision making augmented with human guidance.

1.2 Justification and Relevance

The primary function of autonomy is to unburden humans by automating tasks. Examples herein focus on autonomy through the specific use case of decision making, although autonomy in general need not be limited simply to decision making. Taking the scenario in Figure 1.3 as an example, several technologies enable the unmanned aerial vehicles (UAVs[1]) in Figure 1.4 to accomplish the objectives. A core technology in mission planning is autonomous decision making.

What is autonomy? Autonomy is the ability to synthesize sensory input and a-priori information to make a decision. Autonomy and autonomous decision making can take place over a single atomic event with a clear start and stop (eg. deciding to go left or right at an intersection), an episodic event (eg. finding the sequence of moves that will result in checkmate for a game of chess), or an infinitely long event (eg. allocating processing time on a server to an incoming stream of computation tasks). Autonomy can be applicable to many levels of abstraction, from low-level decisions (beat heart once every second), to high-level decisions (plan a walking route from my office to the vending machine on the 2nd floor).

Why autonomy? Autonomy unburdens humans from performing tasks that are in some way undesirable. The often-cited mantra of *dull, dirty, and dangerous* refers to the areas in which autonomous agent excel; namely, boring tasks, unpleasant tasks, and life-threatening tasks.

Why robots? Many of the tasks we would like to automate involve interacting with the physical world. In the most abstract sense, a robot is any device that takes on human abilities of perception—sensing the world—or action—interacting with the world. Robots are most useful when given some level of autonomy, mapping high-level requests into low-level actions (eg. setting the timer on a vacuuming robot), as opposed to direct operation ("teleoperation"). It is this transfer of abstract human desire to real-world action and task accomplishment that is both the definition of and argument for autonomy.

Why multiple agents? Autonomous agents such as robots become even more compelling work performers when combined in teams. The reasons for teams of robots are no different than the reasons for teams in general: redundancy and robustness, potential cost benefits, and potential productivity benefits. This thesis makes an argument not only for teams of robots, but also team-level autonomy in the form of coordination and cooperation, which are closely related concepts

[1]For purposes here, the term UAV implies some level of autonomy. To be precise, a UAV is a type of Remotely Piloted Vehicle (RPV), which does not necessarily posses any autonomous capabilities.

[27]. The goal in grouping robots into teams is to leverage individual capabilities such that a sets of tasks, unaccomplishable by any one robot, becomes feasible for the team as a whole.

Why hybrid human-robot systems? Many autonomous systems have some interaction with a human operator. The role of the human can range from passive and supervisory to an active sensory input element of the system as a whole, as depicted in Figure 1.5. The best way to incorporate a human operator or human agent into an autonomous system is to consider them at the outset of design.

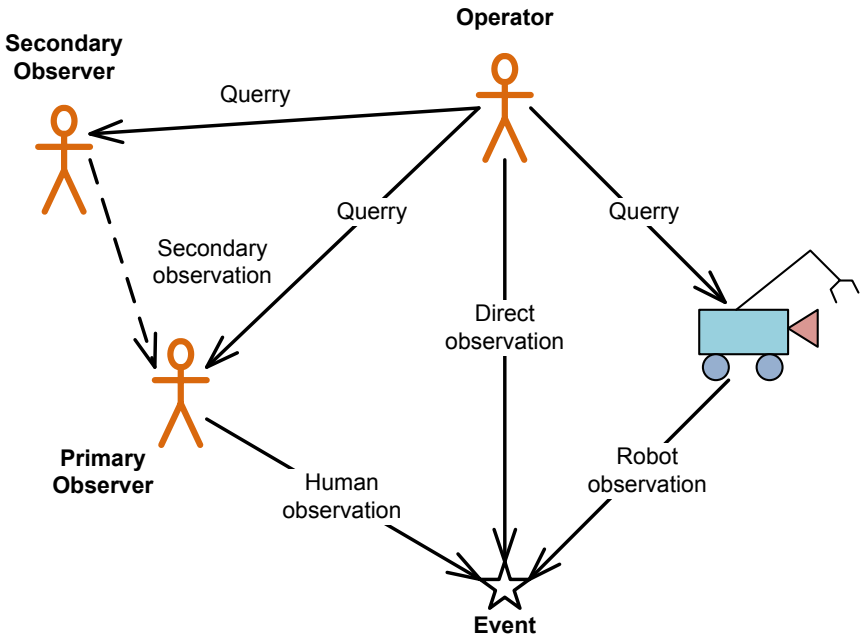


Figure 1.5: Use-case diagram for human interaction with an autonomous system.

In this hybrid human-robot planning scenario, one human operator aggregates data from multiple sources of variable reliability to assess the situation and choose a course of action.

The last point raises the question: What roles can humans play in autonomous systems? It is worth considering the specific roles of all system elements as early as possible in the design phase so that subsequent algorithms, infrastructure, and other implementation details can be designed appropriately. The potential human use-cases in autonomous systems are as follows:

Initialization: The initial problem definition typically comes from a human operator in charge of a team of autonomous agents. As a result, a translation layer is required that can take the abstract problem definition and put it into a formal language and instruction set.

Operation (open-loop): Once an autonomous system is set in motion, the operator may wish to start, stop, or change the system goals or constraints due to changing priorities or unforeseen circumstances.

Feedback (closed-loop): In scenarios in which the autonomous system is continually sensing and reacting to the environment to accomplish a task, the human may wish to take an active role at any time during execution. In this scenario, the operator is an active element in feedback with the system, and therefore must be considered in the design and analysis if stability or performance guarantees are to be applied to the system.

Learning: Consider the previous scenario in which the operator observes the run-time of the autonomous system and chooses to make modifications to the computed decisions. One point of view would be to consider the human as a disturbance to the system that needs to be compensated for or rejected. In some scenarios though, the human input may be be better defined as corrective guidance. In this case, information should be extracted from the human input to either correct the solution, or correct the decision-making engine driving the autonomous system.

1.3 Related Fields

The work in this thesis draws upon research in several fields. Associated fields can be grouped according to the organizational chart in Figure 1.2. The classification given below groups related work functionally as *Design*, *Theory*, and *Application*.

1.3.1 Design

1.3.1.1 Representation

Decision problems are inextricably linked to their underlying representation, or state. The choice of state may be obvious in certain highly-structured domains: The state for a pendulum is commonly parameterized by the angle of rotation, θ, for example. However, in fields such as computer vision, the underlying states can be the intensity value for each pixel, some higher level features of the state, or a combination of features. General feature identification and extraction is an active area of research for many fields that deal with large amounts of unstructured data.

The choice of representation is the first and most important step in solving abstract decision problems such as the mission planning scenario in Figure 1.3. As many of the examples herein are inspired by robotics, multi-agent systems, and vehicle routing, representational inspiration comes from these fields as well [72]. Ultimately, the choice of representation is largely a matter of design, and is a primary contribution of this work.

1.3.2 Theory

1.3.2.1 Decision Problems and Machine Learning

This thesis relies upon the formalism of Markov Decision Processes and several associated variants. Many of the algorithms and example problems are from Sutton and Barto [66], Thrun, Burgard, and Fox [71], and Bertsekas and Tsitsiklis [6]. Semi-Markov Decision Processes are an important extension that underpins much of this research [63]. Several of the examples presented later hinge upon Reward Shaping techniques developed by Ng, Harada, and Russell to incorporate human guidance [50].

1.3.2.2 Human-Machine Interaction

A central goal of this thesis is to learn solutions to abstract decision problems with human guidance. This concept is closely related to supervisory learning in reinforcement learning [56]. Human-Robot Interaction (HRI) is nearly synonymous, and deals specifically with information communicated between human users and robotic agents [30]. HRI has been studied in the specific context of mission planning by the authors of [31]. This research makes use of concepts developed in HRI/HMI fields, powered by tools from reinforcement learning.

1.3.3 Application

1.3.3.1 Assignment, Routing, and Scheduling

The problem of optimally allocating agents to tasks is generally known as an Assignment Problem (AP) [10, 33]. The examples motivating this work are closely related to AP variants, including the Traveling Salesman Problem (TSP) [38] and Vehicle Routing Problem (VRP) [72]. Both the TSPs and VRPs address the optimal routing of agents to destinations, and many variants are based around the number and type of agents, number and type of destinations, spatial and temporal constraints, and other ordering constraints. Scheduling is a closely-related field to routing that addresses the optimal assignment of jobs to machines [13]. The types of problems considered here can best be described as spatiotemporal decision problems that combine elements from both routing and scheduling.

1.3.3.2 Mission Planning

Several motivating examples in this work draw upon the broad field of mission planning. Much of mission planning research has theoretical underpinnings in multi-agent task and resource allocation [25], consensus [51], and multi-agent optimization [6, 43].

Large-scale field robotics mission planning demonstrations have been performed by the C_3UV group with teams of UAVs [60, 57, 23], and by the Australian Centre for Field Robotics (ACFR) with a combination of UAVs and Unmanned Ground Vehicles (UGVs) [32], to name several examples. A comprehensive review of existing robots and applications that

can broadly be characterized as *distributed information gathering* problems can be found in [40].

1.4 Objectives

The overarching goal of this research is to close the gap between computation capability and human ability in the area of high-level decision making. This goal is predicated on the fact that computers (servers, desktops, smartphones, robots, etc.) are particularly adept at lower-level computationally intensive tasks, while humans exceed at tasks involving abstract though. This work seeks to identify intelligent means of combining computer and human decision making capabilities for learning solutions to complex decision problems. To this end, associated objectives are as follows:

1. To develop representations for high-level spatiotemporal decision problems involving multiple agents, multiple tasks, and soft temporal and spatial constraints.
2. To identify a modeling formalism best suited for the decision problems considered herein.
3. To learn generalizeable solutions across problem domains such that the algorithms require minimal tuning.
4. To learn scalable solutions that do not grow intractably large as the problem size changes.
5. To incorporate feedback into the developed solutions for the dual purposes of reducing computation time and improving solution quality.
6. To motivate the choice of representation, underlying model, and solution approach through simulation of several canonical example problems.
7. Based on analysis of the example problems and performance relative to known heuristics, to state the relative merits of the human-in-the-loop reinforcement learning approach to solving decision problems.

1.5 Contributions

This thesis makes contributions in three subfields: multi-agent decision making, machine learning, and HMI.

1.5.1 Multi-Agent Decision Making Contributions

Choosing the correct perspective on a problem can often mean the difference between tractability and intractability. A central contribution of this thesis lies in choosing an effective perspective for problems such as the mission planning example in Figure 1.3. Rather than encapsulating the world in an unscalable set of states, this thesis presents a lower dimensional representation based on features of the overall state. The features are chosen to be broad enough to capture a variety of routing and scheduling-like problems, which are referred to as Generalized Planning Problems (GPPs). The chosen states represent a perspective, which is fundamentally based grouping of like states. Results borne out in simulation make an argument for the effectiveness (scalability, generality, etc.) of this particular representation for several example problems.

1.5.2 Machine Learning Contributions

Diverging from standard optimization and search-based strategies for solving established problems such as the TSP and VRP (both specific decision problem variants), this thesis presents learning-based solutions. Markov Decision Process (MDP) based models provide a well-studied and robust means of capturing many of the problem attributes listed in Section 1.4. However, factors present in many complex decision making scenarios, including time varying problem dynamics, stochastic environments, and action outcome uncertainty, necessitate a more general modeling formalism. As such, the Semi-Markov Decision Process (SMDP) model is proposed as a means to capture temporal problem elements. Known learning algorithms for SMDPs are also customized here to come up with efficient and scalable solutions. Learned solutions are tested against benchmarks and known optimal solutions where they exist.

This work provides a conduit for incorporating human feedback into learned solutions based on reward shaping theory [50]. To the knowledge of the author, this is the first time human feedback has been formally incorporated in such a way so as to provide a method for guaranteeing optimality.

1.5.3 Human-Machine Interaction Contributions

This thesis makes several contributions in the field of HMI. A formal structure is proposed for how humans can interact with an autonomous system, both during solution development ("learn time") and solution execution ("run time"). Human input is guaranteed to improve learning time while preserving policy optimality under reward shaping theory.

1.6 Thesis Structure

This thesis is organized roughly in the same order as the listed contributions. Chapter 2 presents an overview of decision problems, assignment problems, and popular variants. For-

mal problem descriptions are given, as well as motivating examples from real-world scenarios. Related and well established problems are discussed. From these examples, two canonical routing problems are defined and used as benchmark examples in subsequent chapters.

Chapter 3 focuses on algorithms for learning optimal policies on MDPs and SMDPs. These include standard dynamic-programming approaches to reinforcement learning such as on-policy Value Iterating off-policy Q-learning. Motivated by the nature of the problems considered, a justification is given for the choice of algorithm for particular problem-types.

Chapter 4 introduces the GPP as a hybrid routing-scheduling problem framework that is capable of representing all of the basic learning problems discussed in Chapter 3. To demonstrate the benefits of the GPP framework, an example VRP is cast as a learning problem and solved.

Chapter 5 addresses the inclusion of human feedback in learned solutions. This includes both high-level architectures for human-machine interaction and low-level algorithmic details. An example GPP introduced here aims to show the flexibility, generality, and performance of the representation, modeling, learning, and HMI employed here to solve high-level decision problems.

Chapter 6 concludes with a summary of contributions and future perspectives on associated fields of research.

Chapter 2

Decision and the Generalized Planning Problem

Some smart quote

2.1 Introduction

Decision problems underpin nearly all activities in systems ranging from individual robots, to networked power grids. Small robotic systems are primarily concerned with motion and sensing decisions, while large and possibly multi-agent systems need to consider a broader context, often making macro actions that have a hierarchy of associated consequences. For both large and small systems, two fundamental decision problems are routing and scheduling. Routing addresses the problem of moving entities, such as vehicles, from one location to another in an efficient manner—so as to minimize fuel usage, for example. Scheduling is focused on efficient time usage rather than space usage, and typically involves an ordering of tasks such that the overall completion time is minimized. Many problems beyond the canonical routing and scheduling formulations live at their intersection.

Section 2.2 places routing and scheduling in their appropriate contexts with respect to decision problems in general. Section 2.3 focuses specifically on routing and Section 2.4 expands on research in scheduling. Section 2.5 discusses the relationship between scheduling and routing, particularly for two popular problems: the Job Shop Scheduling Problem (JSP) and the Vehicle Routing Problem (VRP). Section 2.6 introduces a new formulation, linking the two fields, and referred to as the Generalized Planning Problem. The remainder of this work focuses primarily on the formulation and solution of specific illustrative GPPs, which combine elements of both routing and scheduling. The discussion of the space of routing and scheduling problems concludes in Section 2.6.2 with example problems from each domain that will be revisited in proceeding chapters.

2.2 Decision Problems

A problem in which the goal is to determine an outcome or action based on input parameters can most broadly be classified as a Decision Problem (DP). For autonomous systems, decisions can come in a variety of forms at various levels of hierarchy. Figure 2.1 shows a generic decision problem as well as several germane to robotics and control applications.

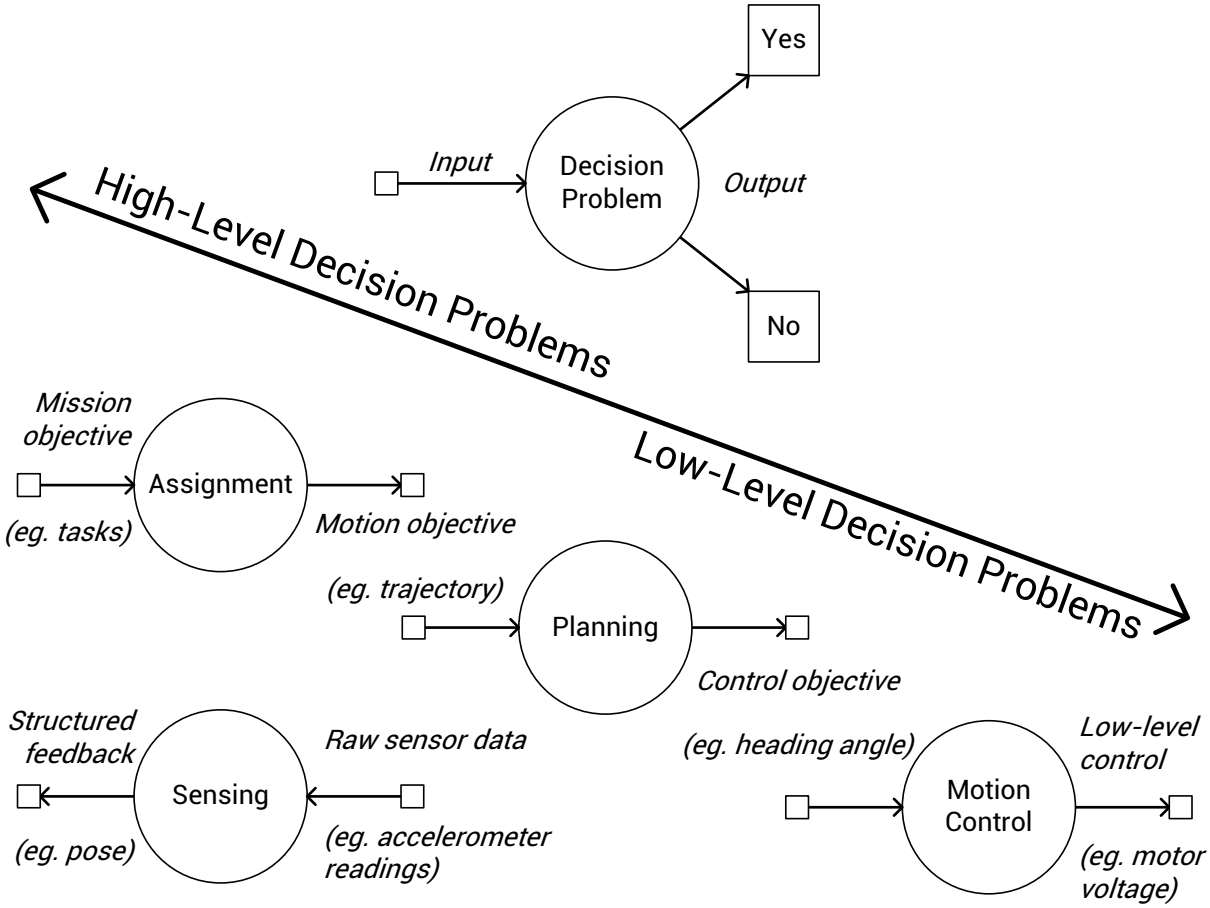


Figure 2.1: Decision problem hierarchy in robotics and control.

For physical systems such as those in robotics, the lowest level DP is motion control. The goal is to calculate a motor actuation signal given reference input or motion command, such as an angle or velocity. One level up in the hierarchy is typically referred to as planning, which determines motion commands based on a desired path or trajectory reference. Above planning, the highest level in the control hierarchy is typically referred to as assignment. APs in robotics involve the allocation of resources (computation, actuation, sensing, etc.)

to tasks. In a single-robot setting, the tasks can be specific actions the robot can take at a given time that require a shared resource. In a multiple robot setting, tasks can be locations to visit, for example, and the AP is thus to find an appropriate routing of robots though the points of interest.

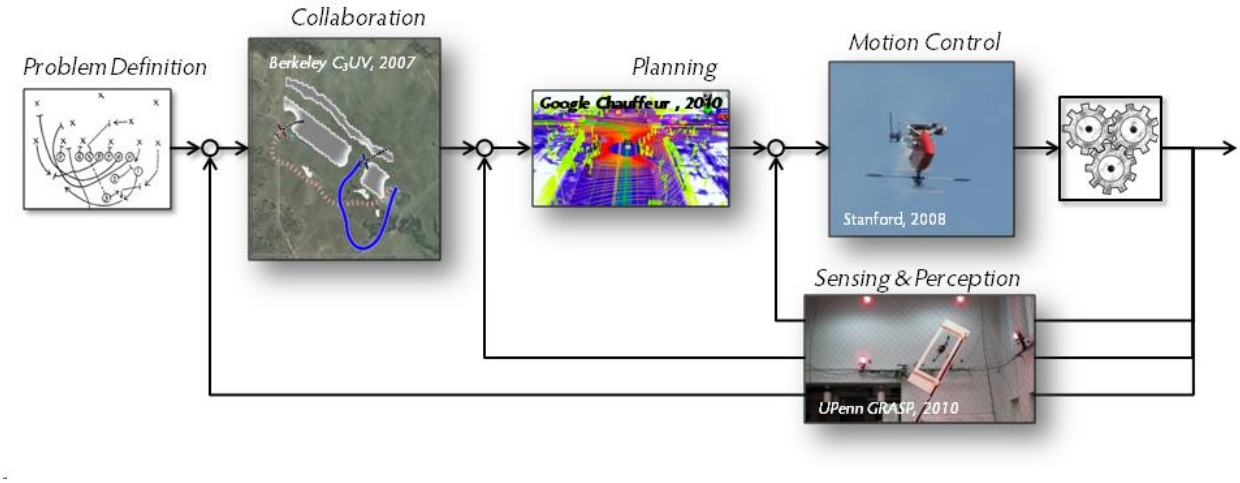


Figure 2.2: Control system perspective of typical robotics decision problems.

Figure 2.2 draws the interconnection between typical robotics decision problems. Each of the DPs described in 2.2 have many subproblems. Several broad categories are depicted in Figure 2.3. As a result of their breadth, DPs are addressed in many bodies of research, including control theory, optimization, and operations research. The focus in this work is on the highest level of DPs, which is referred to as assignment.

2.2.1 Assignment Problems

Perhaps the most broad class of DPs is in the field of assignment. In assignment, the goal is to determine an allocation of a set of tasks to a set of agents that satisfies constraints and meets objectives. A popular APs instance is the Generalized Assignment Problem (GAP), which is formally stated in Definition 1.

The GAP is an NP-hard combinatorial optimization problem, with specific formulations and algorithms depending on the application [11]. Figure 2.4 depicts a GAP in which agents assigned to tasks produce a cost. At this level of abstraction, agents and tasks can live in a physical space, where cost can be interpreted as distance, for example, or in a nonphysical space, where cost may be time or some other task completion metric. Sections 2.3 and 2.4 further explore spatial and temporal APs respectively through routing and scheduling.

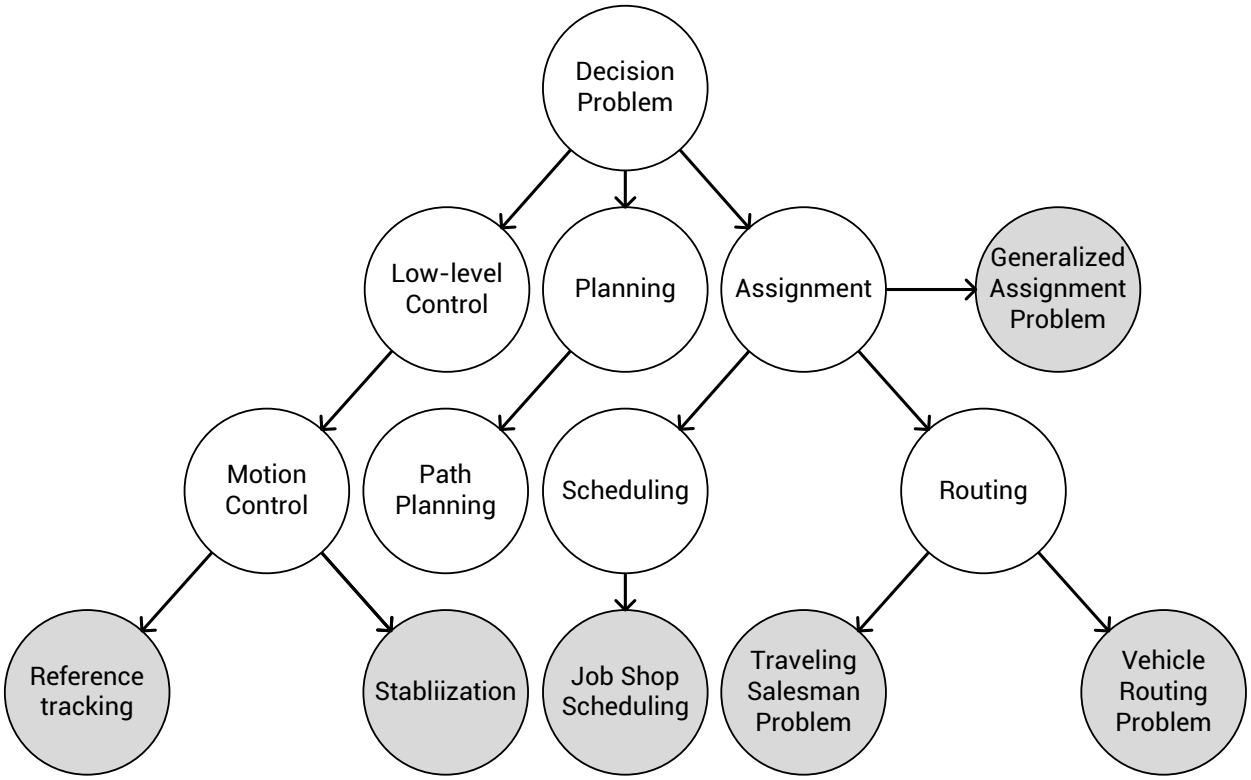


Figure 2.3: Hierarchy of decision problems common in control applications

White nodes are problem classifications (eg. routing), whereas shaded nodes are specific and well-known problems instantiations (eg. The Vehicle Routing Problem).

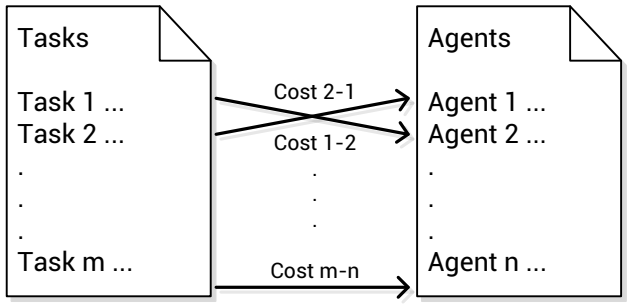


Figure 2.4: Deception of an Assignment Problem.

Tasks are assigned to agents according to some optimization strategy (eg. minimum cost).

Definition 1 (Generalized Assignment Problem)**.**

A GAP is composed of n agents m tasks and a cost c_{ij} that results from the pairing of agent i to task j. Agents and tasks both have size in the most general case, which is to say that each agent can have a unique capacity and each task can have a unique size. The goal is to find a full assignment of of tasks to agents such that reward is maximized while size constraints are obeyed.
The principle components of a GAP are *agents*, *tasks*, and *costs*. Subproblems, such as routing and scheduling inherit the same elements with the following consistent interpretation:

Agent A work performer with attributes that determine the efficiency with which it completes work.

Task An indivisible unit of work. Task properties such as size affect the ability of agents to accomplish tasks.

Cost The product of an agent-task pairing, reflecting the fitness of a specific agent to perform a specific task. A common interpretation is that agents seek to reduce cost, which corresponds to task accomplishment.

2.2.2 Canonical Problems

APs are roughly divided into two categories: spatially-defined problems, commonly referred to as *routing*, and temporally-defined problems, commonly referred to as *scheduling*. Canonical routing problems include the Traveling Salesman Problem (TSP) and the Vehicle Routing Problem (VRP). Canonical scheduling problems include the Job Shop Scheduling Problem (JSP). Routing and scheduling have many similarities, and differ mainly in historical context, representation, and solution method. Section 2.5 details several of these points of comparison.

2.3 Routing

Routing problems commonly appear in everyday life. In basic routing, the goal is to find a path through a set of waypoints that is efficient in some manner (time, fuel expenditure, etc.). Many variants place constraints on the number of vehicles, number of destinations, manner in which destinations are visited, time spent at each destination, and so on. Routing problems are commonly represented as convex optimization problems and then solved by various algorithms or heuristic.

Several representative real-world multi-agent routing problems are listed below. In all cases, the core problem remains the same while spatial and temporal constraints differentiate each instance.

Package Delivery A fleet of trucks seeks to deliver a typically much larger set of packages from several depot locations to destinations around the country. Constraints include package-carrying capacity of each truck, fuel consumption, and deadlines.

Air Traffic Control Given a set of aircraft, start locations, end locations, and intermediate locations, the goal is to find routes such that all planes take off an land on time while obeying fuel constraints.

Taxi Dispatch Perhaps the most complicated of the listed examples, a fleet of taxis seeks to maximize revenue, which is directly correlated to aggregate utilization over all taxis. Pickup locations and destinations may be known or unknown. Constraints can be placed on the length or duration of each route.

As is true of all APs, routing problems are comprised of the principle elements listed in Definition 1—agents, tasks, and costs. The routing-specific interpretation is as follows:

Routing Agent :Typically a mobile entity, such as a person or vehicle, that seeks to accomplish tasks. Agents have a velocity which may be variable and may decrease to zero (ie. ground vehicles). Additional common agent parameters include capacity and other specializations that render an agent more or less fit to accomplish a certain task.

Routing Task A location to be visited. Task accomplishment involves visiting the location for some amount time. Much like agents, tasks may be mobile, but are commonly stationary. Additional task parameters can include a difficulty factor, priority, or other dynamic characteristics as a function of the environment.

Cost Euclidean distance, or some other spatiotemporal metric on which a general notion of distance can be defined. Cost in routing is predominately distance, and may be stochastic in cases of uncertain task or agent location.

2.3.1 History

Routing problems have a long history and draw upon several interrelated fields of research. The first known formulation of a routing-like problem is attributed to the French mathematician, Gaspard Monge, who posed the transportation problem in 1781 [45]. Monage's problem involved moving piles of rubble from one location to an excavation site with minimum work, defined as the average distance traveled.

The first known formalized multi-vehicle routing problem was posed by Dantzig and Ramser in 1959 as the truck dispatch problem [15]. The authors point out that their formulation is special case of the TSP, which is discussed in subsection 2.3.2. The truck dispatch problem is formally stated in Definition 2.

(a) Gaspard Monge, 1781

666 · MÉMOIRES DE L'ACADÉMIE ROYALE

MÉMOIRE
SUR LA
THÉORIE DES DÉBLAIS
ET DES REMBLAIS.

Par M. MONGE.

LORSQU'ON doit transporter des terres d'un lieu dans un autre, on a coutume de donner le nom de *Déblai* au volume des terres que l'on doit transporter, & le nom de *Remblai* à l'espace qu'elles doivent occuper après le transport.

(b) *Mémoire sur la théorie des déblais et de remblais*

Figure 2.5: Gaspard Monge, *Mémoire sur la théorie des déblais et de remblais*, 1781

Definition 2 (Truck Dispatching Problem)**.**

"The Truck Dispatching problem considers the optimum routing of a fleet of gasoline delivery trucks between a bulk terminal and a large number of service stations supplied by the terminal. The shortest routes between any two points in the system are given and a demand for one or several products is specified for a number of stations within the distribution system. It is desired to find a way to assign stations to trucks in such a manner that station demands are satisfied and total mileage covered by the fleet is a minimum." [15]

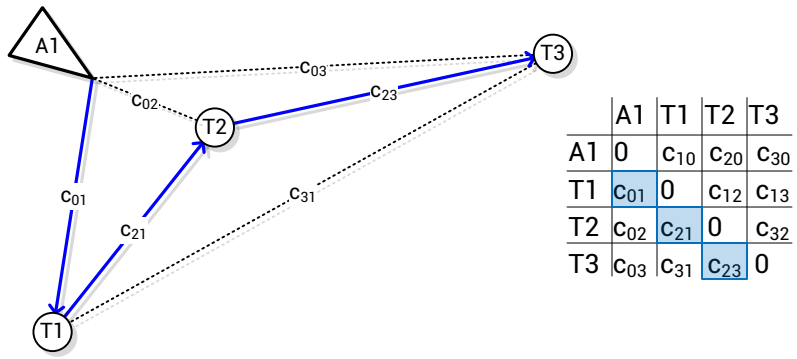
2.3.2 The Traveling Salesman Problem (TSP)

The TSP is perhaps the most studied benchmark problems in optimization and operations research. Formalized by Flood in 1956, the TSP is stated in Definition 3 [21].

Definition 3 (Traveling Salesman Problem)**.**

A salesman starting in a city must visit m other cities once and only once, returning to the city of origin at the end of the tour. Costs are associated with traveling between any two locations, including cities and the origin. The goal is to find the minimum cost tour through the cities such that constraints are satisfied.

The TSP is an NP-hard optimization problem, for which multiple exact and approximate algorithms exist. Additionally, many variants exist based on agent capacity (how many cities the salesman can visit), time, and symmetry in terms of the cost incurred from traveling from city A to city B versus city B to city A.



	A1	T1	T2	T3
A1	0	c_{10}	c_{20}	c_{30}
T1	c_{01}	0	c_{12}	c_{13}
T2	c_{02}	c_{21}	0	c_{32}
T3	c_{03}	c_{31}	c_{23}	0

Figure 2.6: A standard TSP.

Bold arrowed lines indicate the solution: the minimum cost route between cities. In this example, the agent is not required to return to its origin. The cost matrix reflects the costs associated with traversing edges. If costs are symmetric, edges are undirected, and the cost matrix is symmetric.

Perhaps the largest limitation of the TSP is the assumption of a single agent (salesman). The Multiple Traveling Salesman Problem (mTSP) is a variant allowing for multiple salesmen. The mTSP is closely related to the VRP. A full discussion of routing problem variants can be found in subsection 2.3.3.

2.3.3 The Vehicle Routing Problem (VRP)

The VRP is a logical generalization of routing problems such as the TSP to n agents (vehicles). The truck dispatch problem of Dantzig and Ramser is closely related, as it considers "optimum routing of a fleet of gasoline delivery trucks..." and is described as "a generalization of the TSP" in [15]. Like the TSP, the VRP is a known NP-hard combinatorial optimization problem [22]. A comprehensive modern review of VRP algorithms, heuristics, and variants is provided by Toth and Vigo [72].

2.3.3.1 Standard Formulation

The VRP is formally stated in Definition 4, adapted from Laporte [35]. A typical VRP and solution is depicted in Figure 2.7

Definition 4 (Vehicle Routing Problem)**.**

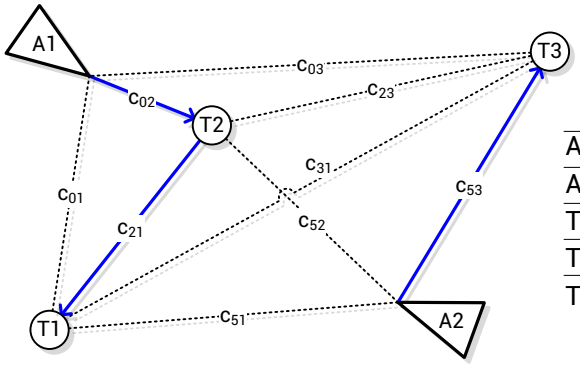
Let $G = (V, A)$ be a graph of vertices V and arcs A. Vertices represent cities to visit with a starting depot at vertex 1 by convention. Arcs connect vertices and are used to represent path costs. Let $C = c_{ij}$ be a matrix containing the path costs for represented by arcs (i, j) for $i \neq j$. For symmetric C (ie. symmetric path costs), arcs A can be replaced with undirected edges E. The number of vehicles n at the depot is fixed. All vehicles are identical in terms of capacity, speed, and other attributes.
The goal of the VRP is to find the set of least cost tours with respect to the problem definition while obeying the following constraints:

C1. Each city is visited once and only once with equal priority.

C2. All vehicle tours start and terminate at the depot.

C3. The number of cities and vehicles remains fixed.

C4. The cost C is deterministic and fully observable.



	A1	A2	T1	T2	T3
A1	0	-	c_{10}	c_{20}	c_{30}
A2	-	0	c_{15}	c_{25}	c_{35}
T1	c_{01}	c_{51}	0	c_{12}	c_{13}
T2	c_{02}	c_{52}	c_{21}	0	c_{32}
T3	c_{03}	c_{53}	c_{31}	c_{23}	0

Figure 2.7: A two-agent three-task VRP.

Bold arrowed lines indicate the solution: the set of minimum cost tours visiting each waypoint only once. The cost matrix C reflects the costs associated with traversing edges. If cost are symmetric, edges are undirected, and C is symmetric ($c_{ij} = c_{ji}$).

Clearly the TSP and VRP have a similar inspiration and interpretation. It is however useful to point out where the two classic problems differ:

1. The TSP is inherently single-agent, while the standard VRP is multi-agent.

2. Historically and in practice, salesmen in TSP have infinite capacity, while vehicles in the VRP may have capacity constraints.

3. Many formulations of the VRP lift the "return to depot" constraint, while the TSP often requires the salesman to return to the starting city.

2.3.3.2 Common VRP extensions

Several common extensions to the standard VRP formulation follow directly from relaxing the constraints listed in Definition 4. Acronyms are adapted from [35]; the precise origin and name of each variant is difficult to determine.

Capacity-constrained VRP (CVRP) Also known as the capacitated VRP, each vehicle is assigned a carrying capacity q_i, and each city a demand d_j. Vehicle capacity must not be exceeded along any route. Closely related, the total number of cities along a route may be upper bound (a special case of the CVRP).

Distance-constrained VRP (DVRP) The length or time of any route is upper bounded. Route length includes transit time between cities and stoppage time at each city. For fixed vehicle velocity, distance and time constraints are effectively interchangeable.

VRP with Time Windows (VRPTW) Specifies a visit interval $[t_1 t_2]$ during which a specific city must be visited.

Prioritized VRP (PVRP) The order in which cites may be visited is constrained.

VRP with Pickup and Delivery (VRPPD) A set of capacity-constrained vehicles must complete a set of pickup and delivery tasks, each defined by and origin and a destination. This formulation differs from the standard VRP in that each origin has an associated known destination.

Undoubtedly more variants exist. Extensions are possible by relaxing several constraints at once—the Capacitated Vehicle Routing Problem with Time Windows (CVRPTW), for example. An extensive list of VRP special cases and known solution methods is provided by Golden and Assad in [26]. The notation given in Table 2.1 will be used for the remainder of this work to describe routing and routing-like problems.

Variable	Definition
n	number of agents or vehicles
m	number of tasks, cities, or customers
$\mathbf{d} = [d_1...d_m]$	non-negative demand weight applied to tasks
$\mathbf{q} = [q_1...q_n]$	non-negative vehicle capacity weight
$\mathbf{v} = [v_1...v_m]$	graph vertices representing tasks
$\mathbf{C} = [c_{11}...c_{nm}]$	cost edges between vertices

Table 2.1: Routing problem notation.

2.3.3.3 Routing Problems Solutions

Many exact and heuristic algorithms exist for solving the VRP and its variants. Laporte and Nobret highlight three exact algorithm classes in [34] and [35]: *tree search*, *dynamic programming*, and *linear programming*. Some solutions hinge on problem relaxations that yield TSPs or GAPs, which are then solved by one of the three listed techniques. A description of exact algorithm classes is follows:

1. *Tree search*: A brute-force search through possible vehicle routes. Methods such as branch-and-bound and others from TSP and AP literature apply here.

2. *Dynamic programming*: Proposed as a solution method in [20]. Due to vast number of states required to represent a VRP, effective use of dynamic programming methods requires reduction in state space size, function approximation, or other compact problem formulations.

3. *Integer Linear programming*: The VRP is formulated in terms of sets of cities, decision variables, and parameters. The objective is defined to be linear and the decision variables are integer. Sets are clusters of cities to be visited by a vehicle. Decision variables include binary variables on each arc representing whether or not the arc is on the tour. Parameters defining the remainder of the problem include demand, vehicle capacity, arc costs, etc. Constraints are then defined on the decision variables and parameters.

In addition to the exact algorithms, several heuristic algorithms, including nearest neighbor methods, incremental improvement, and others have been shown to effectively solve the VRP. Clarke and Wright are accredited with developing a popular heuristic, which incrementally builds routes based on relative cost savings [12]. Many algorithms derive from TSP literature, particularly the closely related mTSP. Heuristic approaches, while leveraging problem structure to reduce computation and complexity, come at the penalty of suboptimality or loss of feasibility.

2.3.3.4 Stochastic Vehicle Routing

The VRP is an extremely general and flexible framework for describing routing problems. However, absent from the standard formulation is a notion of stochasticity, which every real-world problem exhibits to some degree. Randomness in the VRP can enter through several avenues:

Random Cities Referred to as the VRP with Stochastic Customers (VRPSC) (or Cities), each vertex v_i exists with probability p_i. Demand is deterministic and fixed.

Random Demand Referred to as the VRP with Stochastic Demand (VRPSD), the demand d_i associated with each city v_i is a random variable. City existence is fixed and known. Demands may be independent and identically distributed $\mathbf{d} = (d_1 ... d_m)$, $\mathbf{d} \sim IID$, or may have some dependence relationships.

Random Travel Time Referred to as the VRP with Stochastic Travel Time (VRPST) in [72]. Random travel times are expressed as random cost edges, which can be interpreted as stochastic inter-city travel times or stochastic distances.

The set of problems describe here make up the class of Stochastic VRPs (SVRPs). Traveling salesman version exist for nearly every variant discussed above, as well as for multiple vehicles (TSPSC, mTSPSC, etc.) A comprehensive review is provided by Gendreau, Laporte, and Sguin in [24] and Toth and Vigo in [72].

SVRP literature consistently highlights the complexity incurred by introducing randomness into the VRP. Dror points out in [19] that all but the smallest routing problems are impossible to solve with stochastic programming. Several algorithms consider two-step solution approach in which an *a priori* solver finds the optimal solution over all possibilities, and an *a posteriori* solves over specific realizations of random variables.

The effect of deleting and adding vertices and edges at random can produce counter-intuitive results. Two such examples of the SVRP are highlighted in [24]. In the VRPSC and VRPSD, the end-cost of a particular route is found to be direction-dependent, even when path costs are symmetric (ie. $c_{ij} = c_{ji} \quad \forall i, j \quad i \neq j$). Additionally, increasing vehicle capacity (ie. vehicles can visit more cities per tour) can result in larger solution cost.

The body of research in SVRPs indicates that solving stochastic routing problems of a meaningful size is challenging. The price of stochasticity is complexity, which ultimately reduces the size of feasible problems. Heuristic approaches are necessary to solve large problems, or problems for which exact solutions do not exist at any scale.

2.3.4 Routing Limitations and Summary

The TSP and VRP including all variants surely cover a wide swath of possible routing scenarios. Variants are generally based on relaxing constraints from the most general case. Distance, time, and capacity constraints are all possible with relatively minor changes to

standard routing solvers. Despite the breadth of the VRP and its variants, several limitations diminish the expressiveness of the problem, indicating room for a broader approach to routing and assignment problems. VRP limitations are as follows:

Soft Constraints The standard VRP does not support soft constraints, such as soft task priorities, soft deadlines, and sliding time windows. Such constraints arise in many real-world scenarios where the problem designer is either 1) unsure of task priority, or 2) explicitly wants to build ambiguity into the problem. For example, consider the problem of choosing a driving route in the presence of anticipated traffic. A particular route may vary continuously from more desirable to less desirable as a function of known quantities such as time of day and stochastic quantities such as traffic distribution.

Stochasticity The VRP has been extended to include random elements via the SVRP, which can be solved by stochastic programming methods, or approximately with heuristics. However, SVRPs are difficult to pose and can produce unexpected results. Few solutions exist for problems that incorporate multiple stochastic elements, such as random vehicle dynamics and capacity, random environmental elements, and random task location and demand.

Hybrid/Stochastic Constraints Little specific work exists in combining stochastic problem attributes with hard or soft constraints. For example, a randomly appearing time window, which could be used to model weather, would be difficult to pose in a classical VRP setting.

Beyond the specific limitations of the VRP, namely lack of soft or hybrid constraints and stochastic representations, perhaps the most limiting shortcomings lie in problem definition and solution generalization. In most cases, even slight constraint modification requires resolving from scratch. Solutions tend to generalize poorly, even in cases involving effectively equivalent problems.

Table 2.2 summarizes the landscape of VRPs in terms of the ability of each variant to capture different problem dynamics. The TSP is shown for reference; the mTSP is effectively equivalent to the VRP for the purposes of this analysis.

Table 2.2 additionally considers certain macro problem characteristics: agent, task, and environment dynamics. Each of these characteristics can be either deterministic or stochastic. Dynamics in the context of APs (scheduling, routing) have the following interpretation: Problem dynamics can be either deterministic or stochastic. Stochasticity implies that one or more of the defining agent, task, or environment attributes is a random variable. According to standard VRP formulation, agents and tasks are defined mainly on space and time dimensions. Agents may have an additional capacity dimension (q_i), and tasks may have an additional demand dimension (d_i). Certainly, many other SVRP variants exist with additional random problem attributes.

- *Agent Dynamics*: At a certain level of abstraction, it is reasonable to assume that agent motion is roughly deterministic. For example, it may be unnecessary to consider flight

dynamics when solving an air traffic control problem over the entire United States. At small spatial scales however, stochastic agent motions can reflect uncertainty in the interaction between agents and their environment; wind effects on aircraft, for example.

- *Task Dynamics*: In practical application, tasks are commonly defined by a human user and are therefore known. Stochastic task dynamics would capture either uncertainty in the definition process or uncertainty in the environment-task interaction. The VRPSC could be considered a VRP with dynamic tasks (customers) in that tasks appear and disappear at random. The VRPSD also exhibits task dynamics in that the demand—effectively the difficulty factor of the task—is a random quantity.

- *Environment Dynamics*: The environment is the underlying medium that binds agents to tasks. In a deterministic environment, the cost associated with an agent-task pairing, c_{ij}, is deterministic. Stochastic environments essentially amount to random c_{ij}, which is typically defined as a function of distance. The VRPST may be interpreted as a routing problem with stochastic environment dynamics.

The concept of stochasticity as applied to APs, and particularly routing problems, is of central focus in this work and will be addressed in subsequent chapters. Larsen describes a class of Dynamic Vehicle Routing Problems (DVRPs in his work, not to be confused with the Distance-constrained VRP here) which define a superset over classical static VRPs [36, 37]. Larsen defines a dynamic VRP as a VRP in which 1) not all information is available to the planner at the time the problem is defined, and 2) information can change after routes have been defined. To the knowledge of the author, no work has addressed routing problems with both multiple stochastic or unknown problem variables coupled with soft constraints.

	city/customer v_i	task demand d_i	vehicle capacity q_i	travel time/distance c_{ij}	number of tasks	number of agents	agent dynamics	task dynamics	environment dynamics
Fixed Constraints									
TSP	F	F	F	F	F	F	D	D	D
VRP	F	F	F	F	F	F	D	D	D
CVRP[1]	F	C	C	F	F	F	D	D	D
DVRP	F	F	F	C	F	F	D	D	D
VRPTW	F	F	F	C	F	F	D	D	D
Stochastic Constraints[2]									
VRPSC	S	F	F	F	S	F	D	S	D
VRPSD	F	S	F	F	S	F	D	S	D
VRPSCD	S	S	F	F	S	F	D	S	D
VRPST	F	F	F	S	S	F	S	S	S

Table 2.2: Routing problem summary

Several variants of routing problems summarized according to functional and physical metrics. F=fixed, C=constrained, D=deterministic, S=stochastic. "Fixed" implies that the particular variable is a constant quantity (eg. a fixed number of tasks), while "Deterministic" implies that the variable is calculated from a known function (eg. deterministic agent dynamics).

2.4 Scheduling

Scheduling is in many respects and even broader type of AP than routing. Scheduling is the process of deciding how to allocate a limited set of resources to accomplish a set of tasks. The differences between scheduling and routing are primarily due to their history and associated

[1]The CVRP applies a weight to the demand d_i at each city, with all vehicle capacities being uniform. This is however referred to as a capacity constraint: the sum of weights must not exceed the vehicle capacity, q_i. Therefore both d_i and q_i are marked as constrained variables.

[2]To the knowledge of the author, no work addresses stochastic vehicle capacity as opposed to demand (the "C" in VRPSC stands for "cities" as opposed to "capacity' as it does in the CVRP. Though it is likely that the demand at a city is more unknown than the vehicle capacity, it may prove useful in some settings to define vehicle capacity as unknown with a particular distribution, separate from demand stochasticity.

applications: Routing problems involve spatial elements, while scheduling focus on temporal elements. Scheduling here will primarily be considered through the lens of the The Job Shop Scheduling Problem, also commonly referred to as the Job Shop Problem (both referred to as the JSP herein).

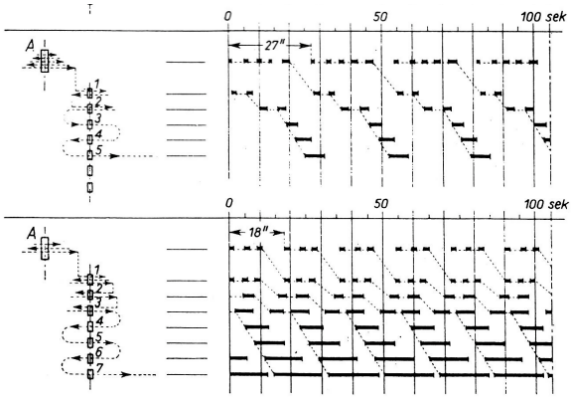
Scheduling, like routing, has broad applications beyond the classical problems such as the JSP. Scheduling theory often addresses problems in the following domains:

- Power grid: Given known and estimated demand, when should utilities be activated?
- Transit systems: What is the appropriate, optimal, or least cost timetable to meet expected demand?
- Cloud computing: What is the best allocation of processing jobs to processors?

2.4.1 History

Scheduling is as old as the understanding of activities and sequencing. Just as with routing, modern scheduling theory evolved from an application. Adamiecki, a Polish economist, is accredited with project management advances in the area of improving production schedules [3]. Adamiecki published an article in 1931 describing a project management tool called a *harmonogram*, a precursor to the modern day Gantt chart.

(a) Karol Adamiecki, 1886-1933



(b) Adamiecki's Harmonogram [3].

Figure 2.8: Karol Adamiecki and the Harmonogram, an early scheduling tool.

The Polish economist, Karol Adamiecki, is credited with the development of one of the first formal scheduling tools, the harmonogram, laying the groundwork for modern scheduling theory.

Modern scheduling theory, like routing theory, draws heavily upon work in optimization, search, and dynamic programming. A classic review is provided by Conway, Maxwell, and

Miller in [13]. The modern literature surveyed here pertains specifically to the JSP, discussed the following sections.

2.4.2 The Job Shop Scheduling Problem (JSP)

The JSP, like its routing counterparts, is a NP-hard combinatorial optimization problem. The JSP is precisely stated as follows in Definition 5.

Definition 5.

Let $J = [j_1...j_m]$ be a set of m jobs and $M = [m_1...m_n]$ be a set of n identical machines. Let $C : M \times J \rightarrow [0, \infty]$ be a matrix representing the cost for machine m to to job j.
The goal is to find the set of sequential assignments of jobs to machines such that overall cost is minimized while obeying the following constraints:

C1. Jobs are atomic and can not be subdivided.

C2. Each machine can perform exactly one job at a given time.

C3. All machines are identical in terms of work throughput.

The standard JSP formulation places no restriction on the number of operations each machine performs on each job. However, a standard assumption is that each of the m jobs requires exactly n operations to complete—one on each machine. The classic objective in the JSP is minimum makespan, where makespan is defined as the overall length of the schedule. From Definition 5, it is clear that the JSP can be considered a generalization of the TSP (Definition 3, ie. the TSP, is a JSP with $n = 1$ (one agent).

2.4.2.1 Common JSP Extensions

As with the TSP and VRP, many JSP problem variants are possible. Listed below are several popular variations, which are devised, like in routing, by relaxing certain constraints.

Non-identical machines: Each machine can have a unique capacity.

Non-atomic jobs: Jobs can be subdivided.

Job priority/ordering: The user can specify a partial ordering of jobs

Two popular scheduling problem variants are the Open Shop Scheduling Problem (OSSP) and the Flow Shop Scheduling Problem (FSP) [49]. In the OSSP each of the m jobs may be processed on any of the n machines in any order, unlike in the JSP where an ordering

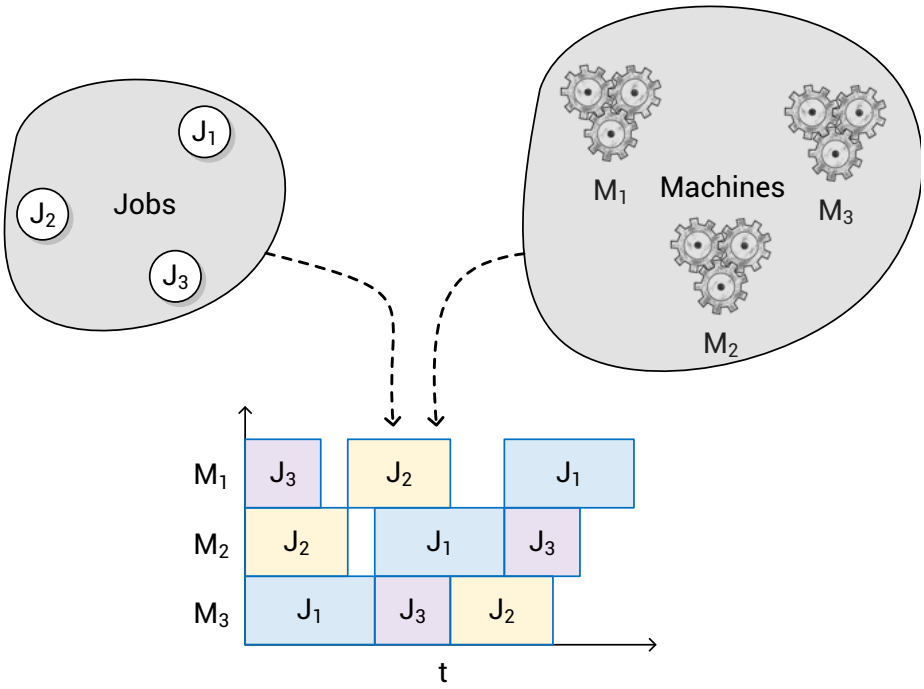


Figure 2.9: A Job Shop Scheduling Problem (JSP) depiction.

All jobs must be executed on each machine exactly once such that makespan, the overall completion time, is minimized.

is specified. In the FSP each of the m jobs has the same unidirectional "flow" ie. requires processing in the same sequence of machines. By contrast, the JSP places no restriction on the flow and is therefore a less constrained and potentially more difficult to solve problem (more options to enumerate). The goal in all cases—JSP, OSSP, and FSP—is minimum makespan, which is equivalent to and often stated as minimum idle time.

2.5 From Scheduling to Routing

Figure 2.3 highlights two key insights into scheduling and routing: both problems are related through assignment problems, and both define closely related subproblems. The difference is the historical context, underlying representation, typical assumptions, and intended applications.

Based on routing and scheduling literature, the following qualitative comparisons can be made between the two problem classes (adapted from Beck, Prosser, and Selensky [5]). This analysis primarily addresses the VRP and JSP, but can be applied to routing and scheduling in general in most cases.

Agent Specialization Vehicles typically have a low degree of specialization in that any vehicle is suitable to perform a given task. However, a common extension of the JSP is machine specialization. The difference in specialization is primarily due to the original historical contexts: trucks are typically all suited to deliver packages, whereas factories with production lines have specialized machines to perform specific tasks. VRPs can be formulated for heterogeneous fleets (trucks, planes, etc.) and JSPs are often homogeneous when the machine is a computer (eg. processor scheduling). That being said, example routing and scheduling problems in literature tend to bifurcate, with homogeneous vehicles for routing and heterogeneous machines for scheduling.

Task ordering In routing problems, tasks typically have no particular ordering in which they need to occur. Contrastingly, jobs in scheduling problems typically need to occur in a particular order. The standard JSP as stated in Definition 5 asserts no task ordering, however, in practice jobs involve multiple sequential operations and ordering constraints need to be imposed. The FSP is a well-studied scheduling problem variant with a strong sense of ordering.

Transit Time and Visit Duration In general, the majority of the cost of a routing task is associated with the time spent getting there, reflecting vehicle fuel costs, with zero time spent at the task site. Contrastingly, in scheduling, nearly all of the cost is associated with doing the task, and zero cost is associated with transit time between tasks.

Temporal Constraints Neither the standard VRP nor the JSP constrain task time. However, a popular VRP variant, the VRPTW specifies that tasks occur during a specific time window, which makes sense in a delivery context in which packages may only be delivered at particular times of the day, for example. In a scheduling context, time constraints typically come in the form of deadlines, due to task ordering (one task must finish before another begins) and the overall objective of minimum makespan. One perspective is that job priority ordering specifies a moving temporal constraint. This point is mostly semantic, as deadlines and time windows are closely related.

Table 2.3 summarizes the differences between routing and scheduling.

A comprehensive study performed by Beck, Prosser, and Selensky abstracts the JSP and VRP to a common level in an attempt to determine the differences between solutions coming from scheduling theory versus those coming from routing theory. The metric for evaluation is the ratio of cost in terms of CPU time between the routing solver and the scheduling solver. Their analysis shows that the routing solver performs better on problems that are more "routing-like" as defined in table 2.3, and the same for scheduling [5]. Perhaps a more significant result, the research indicates that hybrid routing-scheduling problems are possible by abstracting spatial and temporal problem elements to a common level.

	Routing (VRP)	**Scheduling (JSP)**
Agent	vehicle	machine/resource
Task	city/customer	job/activity
Objective	min distance/time	min makespan
Complexity	NP-hard	NP-hard
Agent specialization	homogeneous	specialized
Task ordering	weak	strong
Transit time	long	short
Visit duration	short	long
Temporal constraints	time windows	deadlines

Table 2.3: The VRP and JSP compared qualitatively across several soft metrics.

2.6 The Generalized Planning Problem (GPP)

Previous sections on scheduling and routing highlight advantages and shortcomings of the two problem archetypes. Scheduling, drawing upon lineage in work flow and operations, is inherently focused on temporal aspects of problems. Routing, inspired by the transit of vehicles between locations, tends to focus more on spatial elements. And beyond the standard definitions are a myriad of problem variants. Variants arise by either 1) relaxing constraints, 2) adding constraints, or 3) adding stochastic elements.

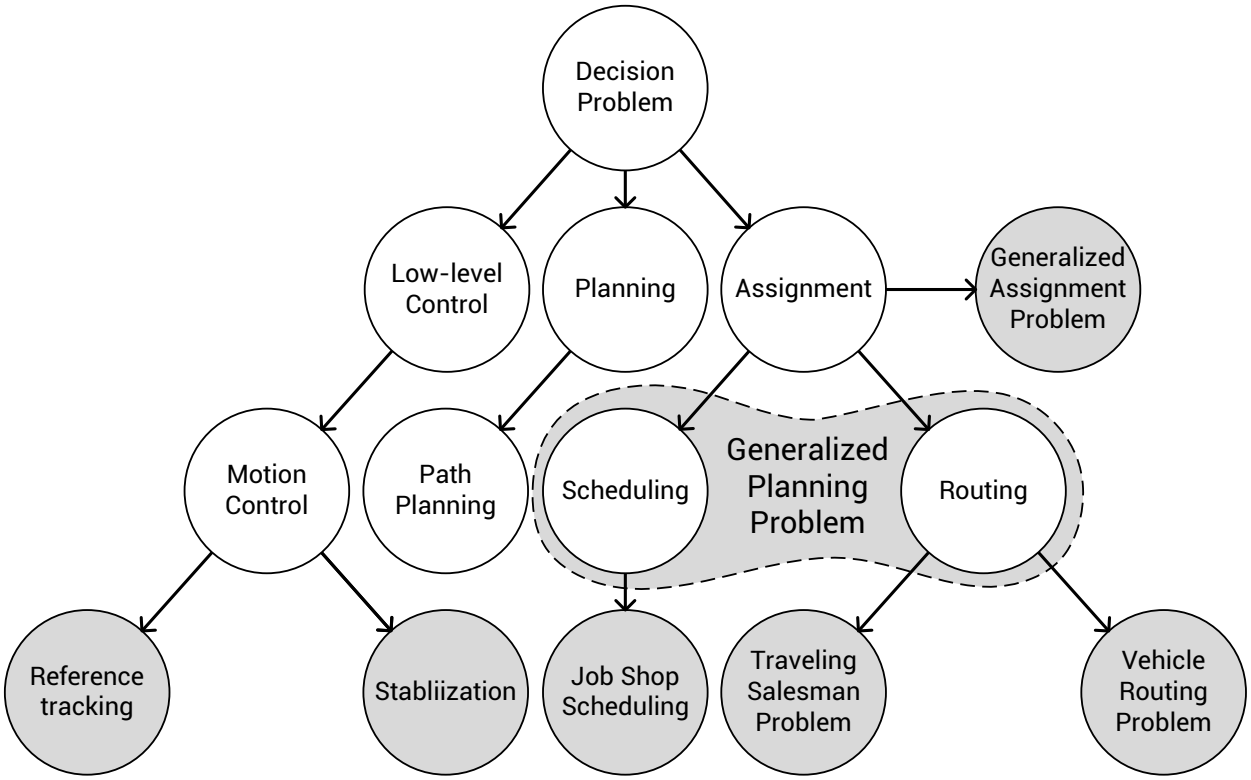


Figure 2.10: The Generalized Planning Problem

The shortcomings of the mostly binary perspective on routing and scheduling problems motivates the need for a new hybrid perspective. The Generalized Planning Problem, as the name suggests, is an application-centric (planning) top down rather than bottom up perspective on assignment problems. It is specific neither towards the spatial aspects of routing nor the temporal aspects of scheduling, and is based around the notion that all problem dimensions can be jointly random. Certainly this will pose significant modeling and solver challenges. Figure 2.10 places the GPP in context with decision problems. Distinct from the GAP, the GPP is concerned specifically with planning problems such as those that commonly arise in multi-agent systems of taskable robots or other work performers.

2.6.1 Definition

The GPP is concisely stated as follows in Definition 6.

Definition 6 (Generalized Planning Problem)**.**

Let the following tuple define a GPP:

$$GPP = [A_1^n, T_1^m, C_{11}^{mn}, f, g] \tag{2.1}$$

The set of agents, $A_1^n = [a_1, a_2, ..., a_n]$ seeks to complete a set of tasks $T_1^m = [t_1, t_2, ..., t_m]$ with minimum overall cost. Cost is defined as a measure of effort or resource expenditure resulting from an agent-task pairing. The cost for agent i to perform task j will be defined as c_{ij}, the collection of which forms the cost matrix C.
Cost is defined at all times as a function of the state of the problem, $x \in X$ and chosen action $u \in U$. The state x encapsulates all agent positions, task positions, constraints, and environmental dynamics.

$$C_k = f_k(x_k, u_k) \quad st. \quad f : X \times U \to \mathbb{R} \tag{2.2}$$

$$x_{k+1} = g_k(x_k, u_k) \quad st. \quad g : X \times U \to X \tag{2.3}$$

The following assumptions are placed on the functions f and g

A1. The cost emission function f is deterministic and known.

A2. The state transition function g can be of any form (linear, nonlinear, time-varying, probabilistic) but must be realizable (in simulation).

The cost for agent i to perform task j (c_{ij}) is the sum aggregate the of 1) cost associated just with the agent regardless of task (eg. startup cost), 2) costs associated just with the task regardless of agency (eg. terminal cost, mandatory loiter time, "demand" in the VRP sense.), and 3) costs associated specifically with the pairing of agent to task (eg. distance between the two). Thus c_{ij} can be represented withe the following sum:

$$c_{ij} = f_k^{ij}(x, u) + f_k^i(x, u) + f_k^j(x, u) \tag{2.4}$$

The state X contains all information about the problem. For the purposes of calculating cost, the state is reducible to the following tuple:

$$X = [S, T, P] \tag{2.5}$$

Where $s \in S$ represents spatial problem elements, $t \in T$ represents temporal problem elements, and $p \in P$ represents additional parameters associated with the environment or specific details of agent-task pairings. Equation 2.4 may be reduced further with this state definition by separating independent elements.

2.6.2 GPP Example Problem

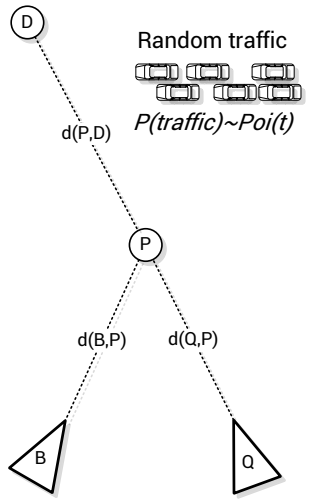
From Definition 6 and Figure 2.10 it should be clear that GPPs combine many characteristics of routing and scheduling. A real-world routing-like scenario is provided blow in Example 1. This scenario is inspired by taxi routing and the fair schedule in New York City. It combines both spatial and temporal elements. Taxi dispatch is not a new problem, and is commonly formulated as a VRP with Pickup and Delivery.

Example 1 described below may be considered as a VRPPD with stochastic agent dynamics (action outcome uncertainty), stochastic environment dynamics (traffic), and soft time windows (drop-off deadlines). To the knowledge of the author, no work has yet attempted to represent or solve a SVRPPDTW (stochastic pickup and delivery VRP with time windows) with traditional vehicle routing techniques.

Figure 2.11: Taxi dispatch in New York City can be posed as a stochastic vehicle routing problem with pickup and delivery

Example 1 (NYC Taxi Routing)**.**

As the owner of a taxi company in New York City you, would like to route cabs as efficiently as possible so as to maximize profit. In the scenario depicted in the figure below, two taxis, one located in Brooklyn (**B**) and the other in Queens (**Q**) may be allocated to pick up the fare at pickup site **P**. The destination of the fare (**D**) is unknown until the time of pickup, t_p.



Param.	Value	Description
$d(.,.)$	$[0, \text{inf})$	distance function
v_M	$(6, 60]$	cab free velocity
v_T	$6mph$	cab traffic velocity
t	$[0, 24]$	time of day
r_f	\$2.50	fixed pickup cost
r_d	$\$2/mile$	moving cost ($v > v_T$)
r_t	$\$.40/min$	stopped cost ($v < v_T$)
r_p	\$1.00	peak cost ($16 < t < 20$)
w_m	\$1000000	cab license cost
w_c	\$20000	cab cost
$p(d\|P)$	$Poi(P)$	destination distance
$p(v\|t)$	$Poi(t)$	moving velocity dist.
p_{fail}	.01	random cab motion

The r parameters denote reward (to be maximized) and w parameters denote costs (fixed in this case). We assume that the overall cost for a particular agent (taxi) allocated to a particular task (pickup) is defined at all times by a combination of deterministic quantities and known probability distributions that can be sampled. The state transition g function is defined as follows: All agents move towards tasks to which they are allocated at velocity v wp. $1 - p_{fail}$. Agents move in a random direction at v wp. p_{fail}, and at $v = 0$ when unallocated.

Solution 1.

The overall cost for agent i to perform task j can thus be expressed as follows, where c_{ij} is defined in Equation 2.4:

$$f^i = w_m + w_c \tag{2.6}$$

$$f^j = -r_f \tag{2.7}$$

$$f^{ij} = \begin{cases} \mathbf{E}[c_{iP} + c_{PD}] & \text{if } t < t_P \\ \dfrac{d(P,D) \times r_t}{v_T} + d(P,D) \times r_d & \text{if } t \geq t_P \end{cases} \tag{2.8}$$

The agent specific costs (f^i) are the cost of the cab and cost of license. The task specific costs (f^j) are the fixed pickup costs. All other costs are a joint function of the particular agent and particular task (f^{ij}). Equation 2.8 separates f^{ij} into two cases: before the pickup ($t < t_p$), when the destination is unknown and treated as a random variable, and after the pickup, when the destination is known ($t \geq t_p$). The full solution can be found in Appendix A.

2.6.3 Summary

The GPP provides a new perspective on routing and scheduling problems. It assumes no specific solution method and is left fairly unconstrained, with many of the constraints pushed into the state as a cost. Other salient GPP attributes include the following:

- Inspired by elements of routing, scheduling, and assignment
- Not tied to a specific solution methodology or tool set
- Inherently based on agents, tasks, and their coupling through the environment and assignment.
- Potentially constrained in several dimensions at once.
- Potentially stochastic in several dimensions at once.

Reinforcement learning as a solution to decision problems will be introduced in Chapter 3. Chapter 4 formulates the GPP as a decision problem and present learned solutions.

Chapter 3

Modeling and Learning

A computer would deserve to be called intelligent if it could deceive a human into believing that it was human. - Alan Turing

3.1 Introduction

Learning is a an innate human process by which everyday experience is internalized as behavior. The process of learning as applied to solving decision problems is commonly referred to as Machine Learning. Reinforcement learning, temporal difference learning, supervisory learning, and many other variants make up a set of techniques for solving decision problems.

Learning theory is often contrasted with control theory, which is a a separate but historically intertwined research thread, also concerned with solving decision problems. Classic control theory dates back to mechanical control in the 1800s, World War II guidance navigation and control systems, and theory developed by Hendrik Wade Bode, Harry Nyquist, and others at Bell Laboratories. Control theory is fundamentally concerned with analysis of the input-output behavior dynamic systems, and the subsequent design of controllers based on this analysis. Properties of interest include stability, controllability, observability, and other metrics that aim to separate the hidden parts of a system from the knowable parts, and then describe how these parts will behave under different conditions. Based on the analysis, classic as well as modern control techniques, including optimal control, model predictive control, and nonlinear control, attempt to regulate dynamic systems so as to achieve some desired behavior.

The following chapter is organized as follows. Section 3.2 provides detail on how learning differs from classical control theory and motivates the need for learning-based controllers. Section 3.3 introduces modeling formalisms typically used in learning research, including the MDP. Section 3.4 discusses learning methods for MDPs based on dynamic programming and Monte Carlo (MC) methods. Having provided a basis and background for learning, Section 3.5 introduces a canonical grid-based maze world example problem. The maze problem is selected as a means for exploring learning variations that improve upon the learning pro-

cess. Section 3.6 concludes the chapter with a review of the results, benefits, and limitations of the learning framework.

3.2 Motivation: Why Learning?

The focus of control theory beyond analysis lies in designing controllers. Closed loop controllers can regulate around a set point (regulation) or follow a reference signal in time (tracking) to meet some objective, often in the presence of perturbations (disturbance rejection). Objectives are either based on performance–rise time, settling time, etc.–or robustness/accuracy–overshoot, undershoot, steady-state error, etc.

The perspective of classical control theory on controller design is certainly analysis first, design second. The learning approach is the opposite: controllers, often referred to as policies, are designed by "training" (learning) on examples gathered from experience.

Learning is closely related to optimal control. Richard Bellman is credited with developing iterative methods for computing optimal policies on a class of deterministic discrete time models known as MDPs. Policies derived from learning are typically implemented as a lookup table, which is analogous to a feedback rule in the control setting. Learning is effectively the process by which information contained in the system dynamics and objective (reward function) is translated into a policy. Several beneficial properties of learning-based controller design are as follows:

Randomness Randomness in the environment or system dynamics is directly encoded in a transition model or simulator. The learning process automatically incorporates this randomness into learned policies.

Experience-based Learning can produce control policies directly from examples, which is especially useful in cases where precise dynamics are unknown or difficult to model.

High-level As a result of only having to generate single examples rather than exact dynamic models, relatively complex behaviors can be learned from an otherwise high-level process.

Tuning Tuning a learning algorithm can often be an unguided laborious process. By the same token, however, many tuning variables allow for direct and precise control over the learning process and eventuality policy quality.

The rich body of literature in control theory has surely addressed all of the above points to some degree. It would be a challenge to find a control problem that is completely unsolvable by one methodology over another. However, just as particular problems lend themselves well to solutions from control theory, particularly those requiring a-priori knowledge and analysis, the learning perspective is often better suited for solving problems with the above characteristics.

3.3 Modeling

Before solving a problem it must be cast in a particular modeling framework. Control theory, for example, makes heavy use of discrete and continuous systems of differential equations. Many analysis tools and constructive design methods are based around these particular modeling formalisms. For historical and practical reasons, the starting point for many learning problems is the Markov Decision Process, introduced in the following section.

3.3.1 The Markov Decision Process

The Markov Decision Process is the cornerstone of reinforcement learning. MDPs incorporate a notion of state, actions, probabilistic state dynamics, and a high-level reward function. The MDP model is formally described in Definition 7.

Definition 7 (Markov Decision Process)**.**

A Markov Decision Process is completely specified by the tuple $< S, A, P, R >$. S is a set of states. A is a set of actions. $R(s', a, s)$ is a reward function that assigns a value for transitioning from state $s \in S$ to state $s' \in S$ under action $a \in A$. The transition model $P(s'|a, s)$ is a conditional distribution on successor states that describes the likelihood of making a particular state transition.

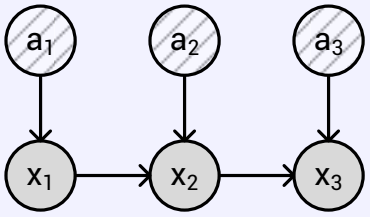


Figure 3.1: Markov Decision Process graphical model.

Significant MDP properties are as follows:

Markov The past and future are are conditionally independent given the present. Successor states are completely determined by one-step predecessor states

One-Step Actions deterministically last exactly one step. MDPs have no notion of history.

Optimality Dynamic programming computes optimal policies on MDPs.

MDPs are best understood as graphical models in the context of companion models. Figure 3.2 shows several graphical models divided according to state observability and actions.

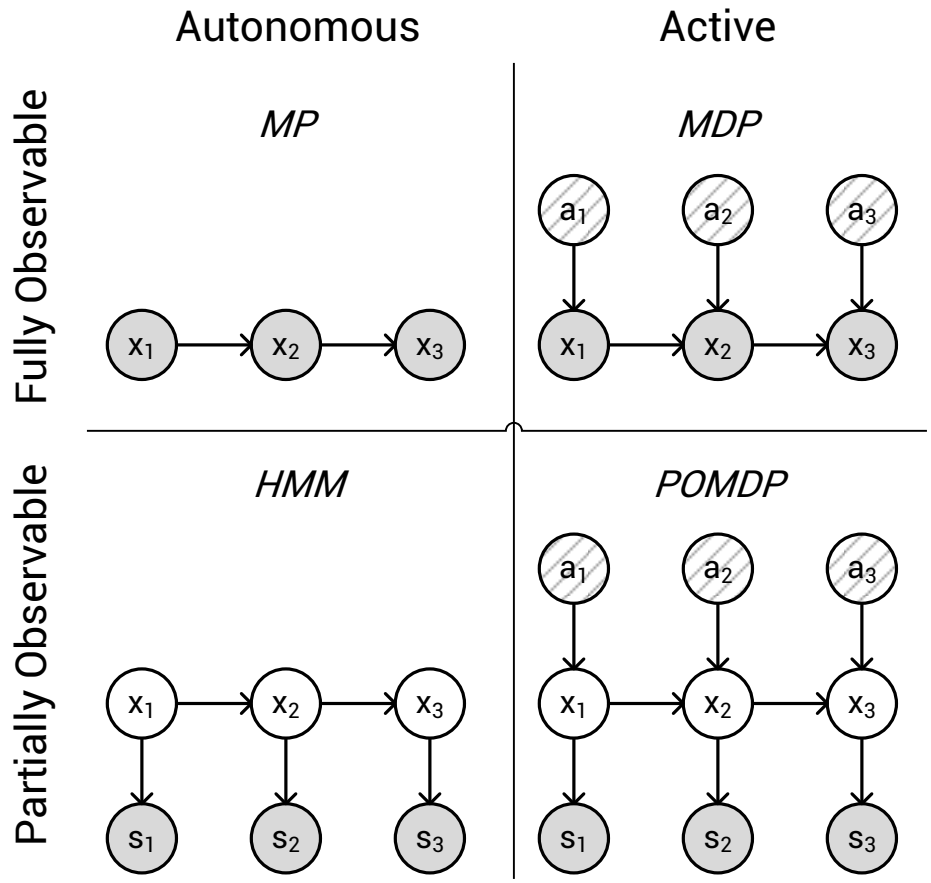


Figure 3.2: Graphical models of canonical decision process models.

The base model in Figure 3.2 is the Markov process, or Markov chain. It asserts the important conditional independence property, $\mathbf{P}(x_1, x_3|x_2) = \mathbf{P}(x_1|x_2)\mathbf{P}(x_3|x_2)$, which is to say that knowing x_2 is all that is required for knowing x_3; x_1 is irrelevant, and x_2 is said to separate the graph. Since MDPs typically represent a sequence in time, an equivalent statement is that *the past and future are independent, conditioned on the present.*

While the Markov processes are useful in many applications for capturing autonomous state evolution, a model of decision problems must incorporate a notion of actions. The MDP model makes this extension. It retains the Markov property from the Markov processes, with the addition that successor states are the produced by actions and governed by a transition model, $\mathbf{P}(s'|a, s)$.

In many applications, the state x of a problem can not be observed. Rather, some lower dimension projection of the state is available. The link between the state and so called

observations is an observation or emission model, $\mathbf{P}(s|x)$, which can be inverted by Bayes rule to determine the true state x. The Hidden Markov Model in Figure 3.2 is a simple autonomous (no actions) graphical model with partial state observability. Hidden Markov Models (HMMs) are widely used in inference problems, such as speech recognition, where the hidden variable represents what was actually said and observations are sounds ("phonemes") that are heard by a microphone.

The most complicated model possible in this framework is the Partially Observable Markov Decision Process (POMDP). POMDPs combine the partial state observability of HMMs with the actions of MDPs. In many cases, POMDPs closely represent real-world decision processes in which actions must be taken under state uncertainty. In robot navigation, for example, a noisy range-bearing measurement may be the observation, the true state is the location of the robot, and actions must be taken so as to safely navigate towards a goal state.

POMDPs combine the expressiveness of MDPs and HMM as well as their aggregate complexity. MDPs, already plagued by the curse of dimensionality, only becomes more complex with the addition of partial state observability. POMDPs typically make value calculations on a *belief state*, $\mathbf{B}(s)$ which is a distribution over a non-random state. Replacing all the states in an MDP with probabilistic states greatly complicates the process of learning. Despite their complexity, much modern research, particularly in robotics, is interested in efficient and approximate calculation of optimal policies on POMDPs [71].

3.4 Learning

The following sections introduce the theoretical underpinnings for learning optimal policies on MDPs. In all cases, the goal is to transfer experiential knowledge into knowledge on how to act in future scenarios. The two algorithmic frameworks considered are dynamic programming and Monte Carlo. Benefits and drawbacks of each are discussed and methods that combine favorable aspects of both are introduced.

3.4.1 Dynamic Programming

Dynamic programming is a technique that can compute optimal policies on MDPs. The salient feature of dynamic programming is that it is an iterative method, updating state values based on future state values. This process is typically referred to as bootstrapping and is depicted in Figure 3.3. The general formulation requires a system that can be represented as in Equation 3.1.

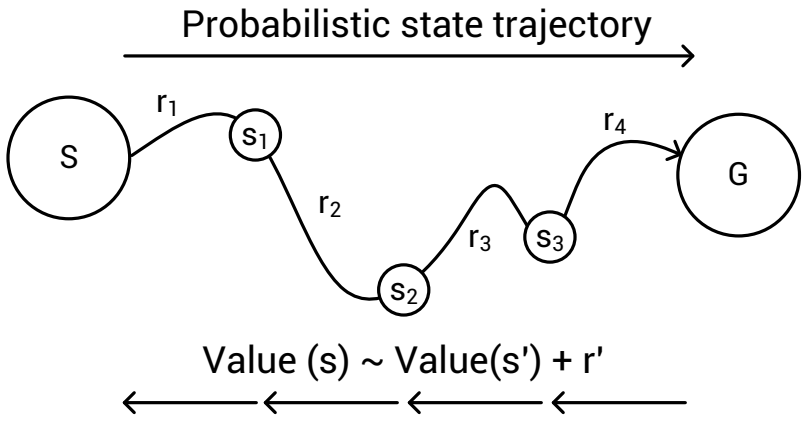


Figure 3.3: Dynamic programming-based learning.

$$s_{t+1} = f(s_t) + g(a_t) \tag{3.1}$$

Equation 3.1 is a discrete time dynamic system. States evolve in time according to f and g in a deterministic manner, or probabilistically ($\mathbf{P}(s_{t+1}|s_t, a_t)$). The goal in dynamic programming is to compute the value for each state or state-action pair. The value for being in a particular state can be defined as the expected reward that will be received in the future starting from that state.

$$V^{\pi}(s) = \mathbf{E}[R_t|s_t] \tag{3.2}$$

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... + \gamma^{T-1} r_T = \sum_{k=0}^{T} \gamma^k r_{t+k+1} \tag{3.3}$$

Equation 3.2 is the value equation for each state under policy π. Equation 3.2 can also be written as a function of itself in Equation 3.5. In this case, $V^{\pi}(s+t+1)$ represents the "cost-to-go" before reaching the goal.

$$V^{\pi}(s) = \mathbf{E}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... + \gamma^{T-1} r_T|s_t = s] \tag{3.4}$$

$$V^{\pi}(s) = \mathbf{E}[r_{t+1} + \gamma V^{\pi}(s_{t+1})|s_t = s) \tag{3.5}$$

It would be useful to be able to calculate V iteratively, based on previous values of V and an expected future reward. That algorithm is known as value iteration and is given with the following bellman update equation.

$$V_{k+1}(s) = \max_a \mathbf{E}[r_{t+1} + \gamma V^{\pi}(s_{t+1}|s_t = s, a_t = a] \tag{3.6}$$

$$V_{k+1}(s) = \max_a \sum_{s'} P(s'|a, s)[R(s', a, s) + \gamma V_k(s')] \tag{3.7}$$

V will converge to V^*. Replacing V with V^* in Equation 3.7 gives Bellman's optimality equation. Sutton and Barto provide a complete review and derivation of value iteration and other learning algorithms in [66]. A detailed treatment of dynamic programming in general is given by Bertsekas and Tsitsiklis in [6].

3.4.2 Monte Carlo

The term Monte Carlo refers to a class of simulation-based algorithms. In the context of learning, MC methods solve for optimal value functions and associated policies. Several key traits of MC, differing from dynamic programming, are as follows:

- **Modeless** No explicit probabilistic model is required, learning is completely experience-based.
- **Simulation-based** A simulator generates state transitions.
- **No Bootstrapping** Whereas DP methods bootstrap, MC methods no not update based on previous estimates.
- **Episodic** Learning occurs at the end of simulated episodes.

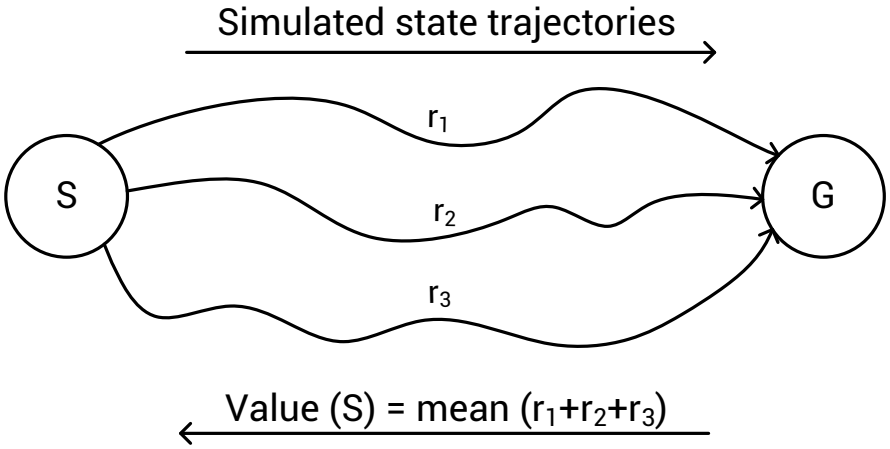


Figure 3.4: Monte Carlo-based learning.

A simulator generates state trajectories from start to goal and the total path reward is recorded. At the conclusion of each episode, the reward adds to the value of the start state. Successive iterations of MC-based value learning results in an averaging of path rewards from many simulated episodes.

A depiction of a Monte Carlo-based learning algorithm at work is given in Figure 3.4. The key assumption in MC value function learning is that a simulator can generate episodes

(sample paths from any start state to the goal state), and learning occurs at the end of episodes. The fact that MC methods do not require a model, such as the probabilistic state transition model in dynamic programming, may seem of dubious benefit, since the simulator must run some sort of model. However, the simulator need only generate a single successor state for each state-action pair, whereas dynamic programming requires a complete conditional distribution for all state transitions.

As an example, consider the card game blackjack as a learning problem. The state, for simplicity, can be the sum of the player's cards and the dealer's showing card. The following types of situations then arise: Given that player's total is 16 and the dealer is showing 10, what is the probability that if the player hits, they win? This type of knowledge is what dynamic programming would require for all possible combinations. Writing out a conditional probability table for blackjack would, at the very least, be a laborious task. Simulating blackjack, however, is very easy. The problem designer need only keep a card deck in memory, simulate random deals and hits, and sum card combinations. Other problems that lend themselves well to simulation are those for which training data does not exist or would be complicated to gather. Learning a controller for a complex vehicle that is yet to be built or even modeled would be a challenge for dynamic programming. However, MC methods could be applied with the highest fidelity simulator available.

Perhaps an even subtler point, simulation is also a beneficial means of learning because it implicitly identifies relevant states. The learning process therefore will necessarily visit the most relevant states and generate the most likely state trajectories. Those states will therefore be updated most frequently and be the most refined at the conclusion of the learning process.

3.4.3 Temporal Difference Learning

The previous sections introduce two methodologies and perspectives on learning optimal value functions. Dynamic programming has the benefit of bootstrapping–estimates of state values are updated based on estimates of future states. Dynamic programming makes a guess based on future guesses. Monte Carlo is not a bootstrapping technique. Rather Monte Carlo methods are steeped in simulation and experience.

Temporal Difference methods, formalized by Sutton, combine the bootstrapping of dynamic programming and experience-based learning of Monte Carlo. Sutton and Barto provide an excellent review of the history, benefits, and applications of Temporal Difference (TD) learning. Suffice it to say that TD methods are advantageous over DP in that they do not require a model of the environment, and advantageous over MC methods in that they can be implemented incrementally and online, learning at ever time step rather than after episode completion. This presents a great benefit to control applications, where learning may in fact be interleaved with actually running the policy learned so far. Table 3.1 summaries the relationship between DP and MC, and how TD methods combine aspects of both.

A simple Monte Carlo update rule is given in Equation 3.8

	Dynamic Programming	Monte Carlo	Temporal Difference
Bootstrapping	✓		✓
Modeless		✓	✓
Episodic		✓	✓

Table 3.1: Dynamic programming, Monte Carlo, and temporal difference comparison.

Temporal difference methods have the benefits of DP methods in that they bootstrap, and the benefits of MC methods in that they are simulation-based and therefore do not require an explicit transition model.

$$V(s_t) \leftarrow V(s_t) + \alpha[R_t - V(s_t)] \tag{3.8}$$

$$R_t = r_{t+1} + r_{t+2} + \gamma^2 r_{t+3} + ... + \gamma^{T-1} r_T \tag{3.9}$$

Updating the state value function $V(s)$ is equivalent to updating a state-action value function $Q(s, a)$, which are related according to Equation 3.12. Policies can be directly calculated from a Q-function as in Equation 3.13. In it's most general form, a policy is a joint probability distribution on states and actions, $\pi(s, a)$. However, in many MDP-based learning algorithms, the state is considered 1) fully observable and 2) realized at each time step. Therefore, a more apt representation would be a conditional distribution over actions $\pi(a|s)$. For the purposes of control, the most useful structure to store is a state-action lookup table, which is no longer random and denoted as $\pi(s)$ and is computed by the maximization in Equation 3.13.

$$V^*(s) = \max_\pi V^\pi(s) \tag{3.10}$$

$$Q^*(s, a) = \max_\pi Q^\pi(s, a) \tag{3.11}$$

$$V^*(s) = \max_a Q^*(s, a) \tag{3.12}$$

$$\pi^*(s) = \arg\max_a Q^*(s, a) \tag{3.13}$$

Since MC updates are based on actual (simulated) reward trajectories, R_t in Equation 3.9 is that actual reward received starting at time t concluding at T. Contrastingly, TD updates are based on one-step predictions of future returns rather than T-step realizations. A simple one-step TD update rule is given in Equation 3.14.

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \tag{3.14}$$

Equation 3.14 implies that, rather than waiting to observe the actual return R_t at the conclusion of the episode, the update is based on an existing estimate of the expected future

return, as given in Equation 3.2. The update rule in Equation 3.14 is both a bootstrapping and a modeless rule, and provides a basis for what is know as the TD(0) learning algorithm.

All TD learning algorithms can be classified as either *on-policy* or *off-policy*. The distinction refers to the policy that is being followed during the learning process, referred to as the *behavior policy*, π_b. This distinction is to distinguish π_b from the policy that we would like to end up with at the conclusion of the learning, the target policy, $\pi_t \rightarrow \pi^*$. The difference between on-policy and off-policy learning is formally stated as follows:

On-policy A learning algorithm that continually estimates the state-action value function Q^π by following the behavior policy, π_b. The behavior policy is simultaneously and incrementally improved towards the optimal policy, π^*, as learning proceeds.

Off-policy A learning algorithm for which the behavior policy need not be based on the current value function, and the current value function Q directly approximates Q^*. The only requirement placed on π_b is that it visit all state-action pairs infinitely often as $t \rightarrow \infty$.

The following sections describe two popular learning algorithms, on-policy SARSA learning and off-policy Q-learning.

3.4.3.1 SARSA

The acronym SARSA refers to the data sequence—$< s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1} >$—required for learning. The algorithm learns by observing the Markovian transition between state-action pairs, and the associated reward. The central update rule is given in Equation 3.15. Algorithm 3.1 provides a standard implementation of SARSA learning.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{3.15}$$

$$\pi^*(s) = \arg\max_a Q^*(s, a) \tag{3.16}$$

3.4.3.2 Q-learning

The algorithm known as Q-learning was first posed by Watkins as a means to learn optimal value functions completely independent of the behavior policy. The only requirement is that the policy being followed is "sufficiently exploratory" ie. all state-action pairs are visited infinitely often as learning time approaches infinity. The update rule is a slight modification on the SARSA update rule, and is given in Equation 3.17.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{3.17}$$

While separating the behavior policy from the target policy may seem of marginal benefit, the practical implications are great. Q-learning can learn from *any* experience, not just

Algorithm 3.1 SARSA Learning Algorithm

Set number of episodes, N
Set parameters: $P = \{\alpha, \gamma, \epsilon, \}$
function $[Q^*(s,a), \pi^*(s)] =$ LEARN(N, P)
  Initialize: Q, π
  while current episode $< N$ **do**
    Initialize: $t = 0$, s_0
    Select action a_t from $\pi_b(s_t)$ derived from Q
    while not at the goal: $s_t \neq s_{goal}$ **do**
      Take action a_t in s_t and observe r_{t+1} and s_{t+1}
      Select action a_{t+1} from $\pi_b(s_{t+1})$ derived from Q
      Update: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$
      $\pi_b(s) = \arg\max_a Q(s, a)$
      $t = t + 1$
    end while
  end while
end function

specific experience. So long as the behavior policy explores the state space, it can be explored in any fashion. Many online learning applications that involve generating experience from actual physical systems can run in parallel with Q-learning. The algorithm only needs to passively observe a state-action-reward sequence to gain knowledge about the system. An implementation of modeless off-policy Q-learning is provided in Algorithm 3.2.

Algorithm 3.2 Q-Learning Algorithm

Set number of episodes, N
Set parameters: $P = \{\alpha, \gamma, \epsilon, \}$
function $[Q^*(s,a), \pi^*(s)] =$ LEARN(N, P)
  Initialize: Q, π
  while current episode $< N$ **do**
    Initialize: $t = 0$, s_0
    while not at the goal: $s_t \neq s_{goal}$ **do**
      Select action a_t from $\pi_b(s_t)$ derived from Q or other method
      Take action a_t in s_t and observe r_{t+1} and s_{t+1}
      Update: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$
      $t = t + 1$
    end while
  end while
end function

3.5 Learning Examples

The previous section has introduced the Markov Decision Process and basics of reinforcement learning. The main focus has been on temporal difference methods such as Q-learning, which combine the best elements of dynamic programming and Monte Carlo methods. Based on these standard algorithms, many variants exist that attempt to in some way enhance the learning process. It is important to point out that these algorithmic variations all attempt to improve speed, accuracy, or robustness, but all aim at the same optimal value function. The learning variations explored here include:

Eligibility Traces A variable lookahead method that attempts to learn state values in problems where the start state and goal state are separated by many steps.

Exploration A behavior policy technique that trades off greediness with respect to the value function versus random action so as to effectively explore the state space.

Prioritized Sweeping A scheme for visiting and update state values in a guided fashion so as to maximize learning.

These techniques are applied to the canonical grid-based learning problem in Example 2. This problem is popular because, while the agent dynamics are simple and the world is limited, it highlights well the differences between learning algorithms.

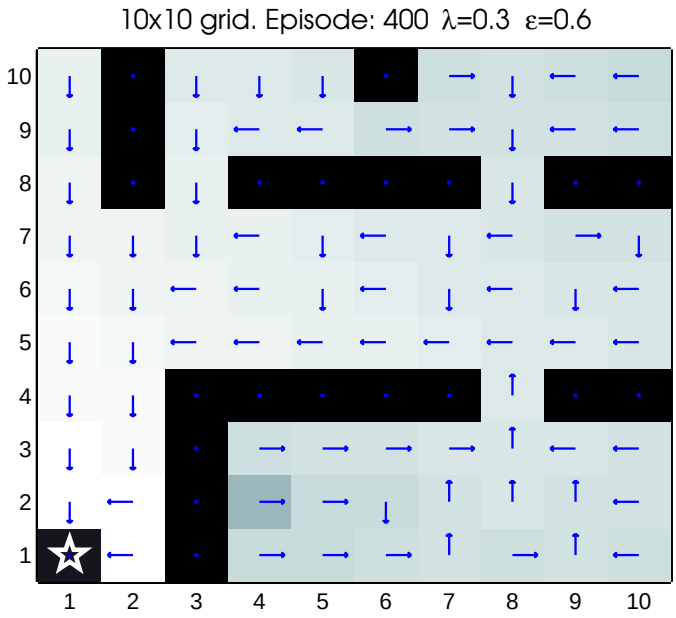
Example 2 (Maze World)**.**

An agent is tasked with navigating in a walled 2D grid world from a start state to a goal state. The goal is to find the optimal policy for navigation through the maze. The problem is modeled as an MDP with the following definition:

Parameter	**Value**	**Description**
w	10	$w \times w$ world size
S	$[i, j] \mid 1 < i, j < w$	state set
$s_{goal} \in S$	$[1, 1]$	goal state
A	$[\uparrow, \rightarrow, \downarrow, \leftarrow,]$	action set
$R(s', a, s)$	$R(s_{goal}, :, :) = +1$	reward function
p_{fail}	0.01	action failure probability
α	0.9	learning rate
γ	0.99	discount factor
ϵ	0.8	probability of following π_b

The reward function gives zero everywhere and +1 at the goal. Agents taking action a_t in state s_t according to the transition model $P(s'|s, a)$ succeed with probability 0.99 and fail with probability 0.01. Walls added to the grid create rooms and narrow passages. The effect of walls is to increase learning time by increasing the length of the paths from start to goal. Learning must also take into account that the shortest path to the goal may involve moving away form the goal for a period of time. The learned policy over successive iterations for the example problem is shown in Solution 2 below. Learning parameters variation is further explored in subsequent examples.

Solution 2.



In the solution, each grid square, representing each of the 100 states, contains several pieces of information. Black squares represent walls, and the black square with the star located at $[1, 1]$ is the goal state. The arrows indicate the policy learned for each state, which is calculated by Equation 3.13. Grid cells are shaded according to the value at each cell(state). Black corresponds to zero value and white corresponds to high value. Value increases with proximity to the goal state since those states are visited most often. The learned policy in some states is suboptimal, as can be seen by arrows not pointing directly at passageways or the goal. This can be solved by longer learning time or tuning parameters.

3.5.1 Learning Metrics and Analysis

Several useful analytical tools give insight into the learning process. Metrics can be roughly divided into those that assess the learning process and those that assess end policy quality. Learning performance and policy quality are not necessarily co-varying and, in fact, often inversely related: An algorithm that takes a longer time to learn for a fixed number of iterations may in fact produce a better policy than a faster algorithm.

Utility The utility of a policy can be calculated as follows in Equation 3.18 by summing the value at each state multiplied by the prior probability of that state. Utility is often calculated online during learning as a measure of how close the policy is to stationary.

$$U^{\pi} = \sum_{s} V^{\pi}(s)\mathbf{P}(s) \tag{3.18}$$

Learning Curve The learning curve shows episode length versus learning iteration. If the behavior policy is at least in part derived from the learned value function, episode length should decrease in time as the $Q \rightarrow Q^*$. Several learning curves are often plotted on the same axis for a varying parameter to compare learning speed and convergence to the optimal policy.

Bellman Error It is often useful to define a Bellman operator, $\mathbf{T}$ that operates on Q as in Equation 3.19. $\mathbf{T}$ is said to be a contraction mapping for which Q^* is a fixed point. The measure of proximity to optimal value function, $Q^*(s, a)$, is the Bellman error, which is calculated state-wise in Equation 3.20. For the purposes of analysis, a scalar Bellman error for each learning step can be calculated as a maximization over all state-action pairs, as in Equation 3.21.

$$Q'(s, a) = \mathbf{T}[Q(s, a)] \tag{3.19}$$

$$B(s, a) = Q'(s, a) - Q(s, a) \tag{3.20}$$

$$B = \max_{s}[\max_{a}[Q'(s, a) - Q(s, a)]] \tag{3.21}$$

Episode Length At the conclusion of learning, the main metric on which to judge a policy is episode length. Episode length is typically calculated by simulating the final learned policy for a variety of initial states and/or goal states if the goal state is movable. For episodic tasks that should all take approximately the same amount to time, it may make sense to average episode length over many test iterations, so as to have a scalar quantity to compare with other policy tests. For non episodic problems, huge state spaces, or problems for which episode length is not uniformly distributed over the state space, several test cases can be used as a benchmark of policy performance.

3.5.2 Eligibility Traces

Learning problems operate on the principle that taking an action in a state produces a successor state along with a reward or a penalty. The received reward or penalty ascribes a certain value to the state-action sequence. Over successive iterations, a state-value function is obtained, which essentially tabulates the value of being in a particular state based on past experience.

Issues arise in learning when the goal state is far from the starting point. In these cases, it may take many steps before reward is applied to the start state or the states along the path. Take for example the maze problem in Example 2. At the outset of learning, when $Q(s, a) = 0$ for all states, the probability of reaching the goal from $s = [4, 1]$ is $\frac{1}{4}^{20}$. Eligibility traces are the means by which temporal gaps between start and goal can be bridged through

temporal credit assignment. Eligibility trace methods may also be seen as a bridge between pure Monte Carlo and Temporal Difference methods. Sutton and Barto provide a review of Eligibility Trace (ET) methods in [66].

MC methods backup the estimate ($V(s_t)$ in Equation 3.23) of the true state-value function ($V^\pi(s_t)$ in Equation 3.22) with the complete return:

$$V^\pi(s_t) = \mathbf{E}_\pi[R_t|s_t] \tag{3.22}$$

$$V(s_t) \leftarrow V(s_t) + \alpha[R_t - V(s_t)] \tag{3.23}$$

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... + \gamma^{T-t-1} r_T \tag{3.24}$$

Equation 3.23 is an estimate of Equation 3.22 because the sample return in Equation 3.24 is used to approximate the real expected return, which is unknown. On the other end of the temporal spectrum lie TD methods, with TD(0) introduced in subsection 3.4.3. TD methods update the value of a state s_t based on a guess of the one-step expected return. The TD update in Equation 3.14 is commonly referred to as TD(0). The return is a one-step estimate of the full return.

$$R_t^{(1)} = r_{t+1} + \gamma V_t(s_{t+1}) \tag{3.25}$$

$$R_t^{(2)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 V_t(s_{t+2}) \tag{3.26}$$

...

$$R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... + \gamma^{n-1} V_t(s_{t+n}) \tag{3.27}$$

In Equation 3.25, the reward is the immediate reward at the next time step and the estimated value of the next state, rather than the full reward used in the MC sample return in Equation 3.24. The corrected two-step return is given in Equation 3.26. Continuing, Equation 3.27 gives what is referred to as the *corrected n-step truncated return*, truncated by n and corrected by the estimated value of the n^{th} state. Clearly as n grows, the sample return takes into account more samples and approaches the MC return. Figure 3.5 shows TD and MC on a spectrum, with the parameter λ controlling the blend between the two methods.

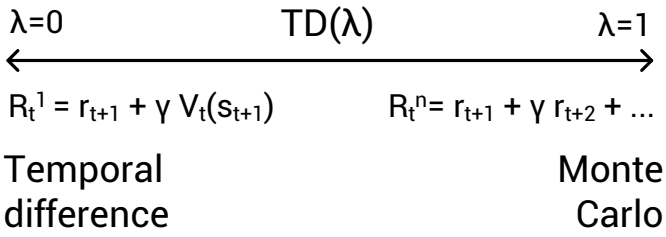


Figure 3.5: Monte Carlo versus temporal difference methods.

As suggested in [66] the return need not be a simple sum of sample returns. With MC on one end of the spectrum, incorporating all rewards from start to goal into each backup, and TD(0) on the other, incorporating the one-step reward plus an estimate of the remainder of the reward, it may be useful to combine these two poles as an average. A commonly-used reward, parameterized by λ, is given in Equation 3.28.

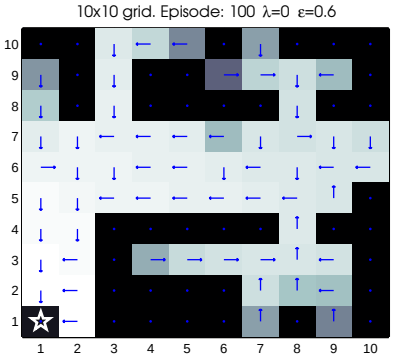
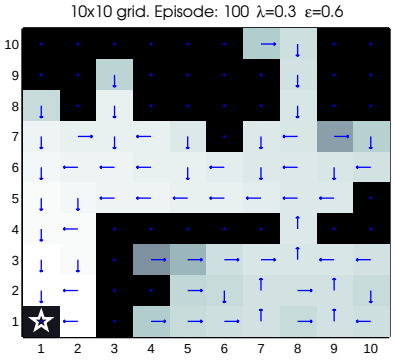
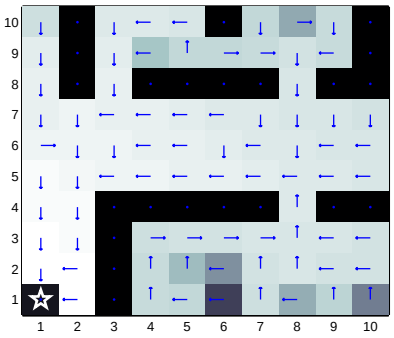
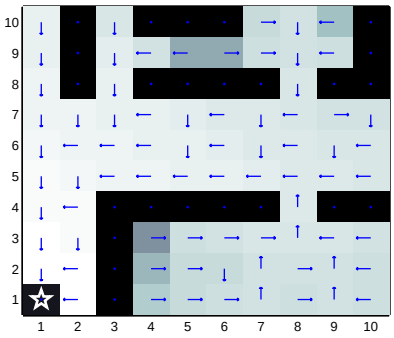
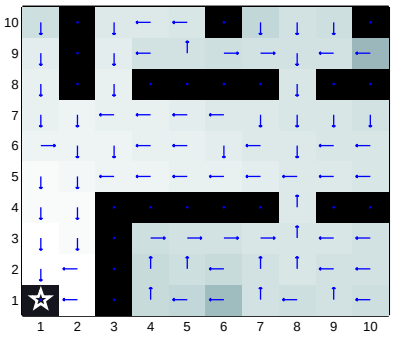
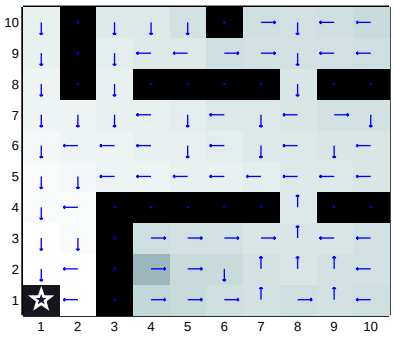
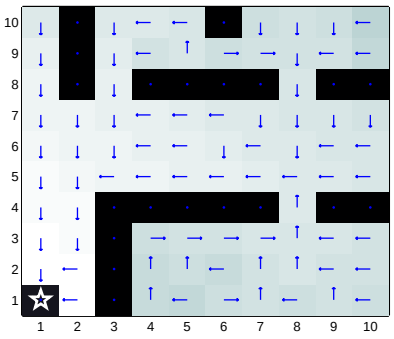
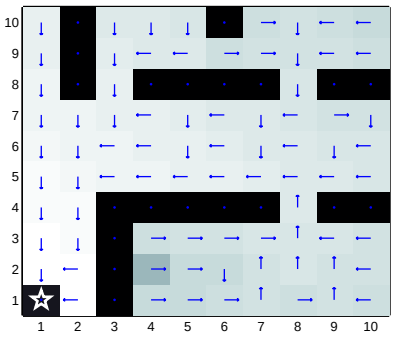
$$R_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}R_t^{(n)} \tag{3.28}$$

The λ-return weights each of the n-step returns in a decreasing manner. For $\lambda = 1$ the return is the MC return as in Equation 3.24. For $\lambda = 0$, the return is the one-step estimate in Equation 3.25. Parameterizing the return by λ produces a new class of off-policy learning algorithms known as TD(λ) methods. The following example demonstrates the effect of eligibility traces on maze exploration problem.

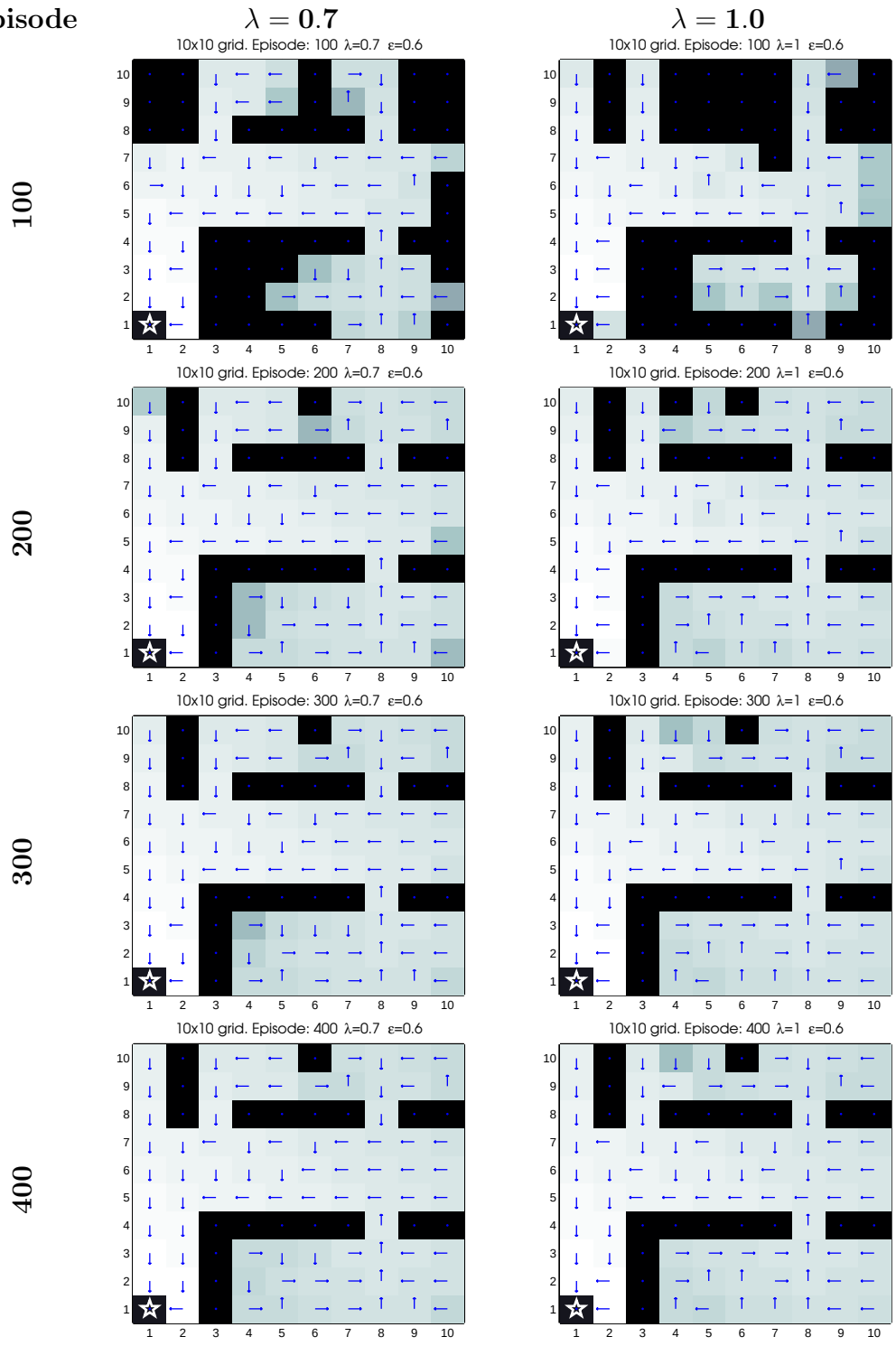
Example 3 (Maze World: Eligibility Traces)**.**

We would like to find the optimal policy for navigating from any start location to the end location in a 10x10 maze world. The underlying MDP, $M =< S, A, P, R >$ is as defined in Example 2. The action set is expanded to include a loiter action. Navigating into a wall results in no progress from the current state (ie. $s' = s$). A positive reward of $+1$ is received for reaching the goal. A additional negative reward of -0.1 is received for loitering in any state.
The optimal policy π^* is solved for over successive learning iterations by Q(λ) off-policy temporal difference learning algorithm (Q(λ)). Policies are learned for $\lambda = [0, 0.3, 0.7, 1]$, which essentially varies the algorithm from pure dynamic programming (TD(0)) to pure Monte Carlo (TD(1)).

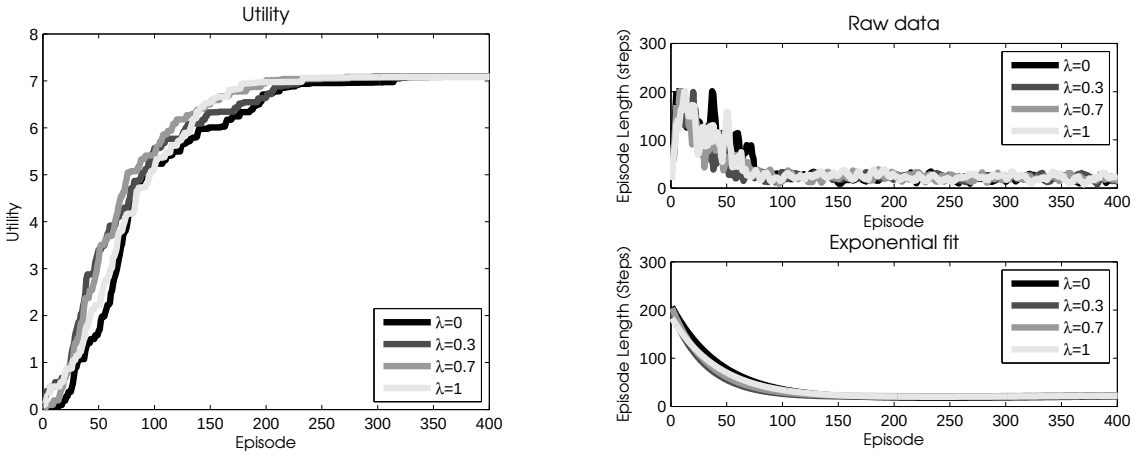
Solution 3 (Maze World: Eligibility Traces).

Episode	$\lambda = 0$	$\lambda = 0.3$
100		
200		
300		
400		

Solution 3 (Maze World: Eligibility Traces).



Solution 3 shows the evolution of the learned policy over successive iterations for varying λ. From the figures it appears that smaller values of λ learn slightly slower. Figure 3.6a shows the utility of the value function over the learning horizon. Several episodes early on in the learning process took more than 1000 steps to complete. However, after the correct policy is learned, particularly at the narrow passageways, learning proceeds rapidly. It does not appear that an appreciable difference exists between the different experiments. Additionally, the learning curves in Figure 3.6b shows that for all values of λ, learning rate is approximately the same.



(a) Utility

(b) Learning Curve

Figure 3.6: Learning results for varying eligibility trace parameter λ.

Some differences exist in the quality of the policy learned for the fixed number of learning episodes. In Figure 3.7, the value of λ that produces the best results in terms of policy quality is $\lambda = 0.3$. The results in [66] show a larger value of λ for a similar maze task. It is difficult to predict whether a large or small value of λ will be optimal for a given problem. Certainly problems that require many steps before arriving at the goal will update many states at once as $\lambda \to 1$. As learning proceeds and more of the states have a non-zero value, $\lambda \to 0$ may be better, as updates occur at each time step rather than at the end of the episode.

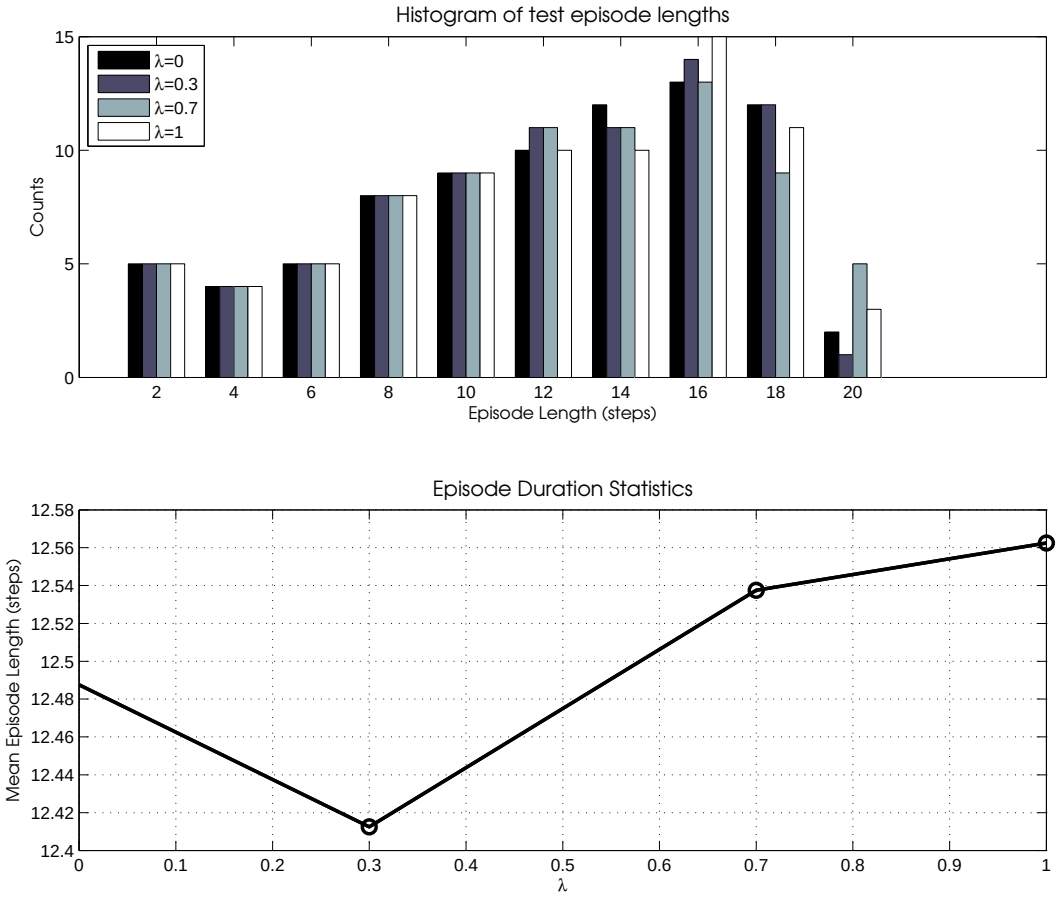


Figure 3.7: Policy test varying eligibility trace weighting λ for 10x10 maze world.

3.5.3 Exploration versus Exploitation

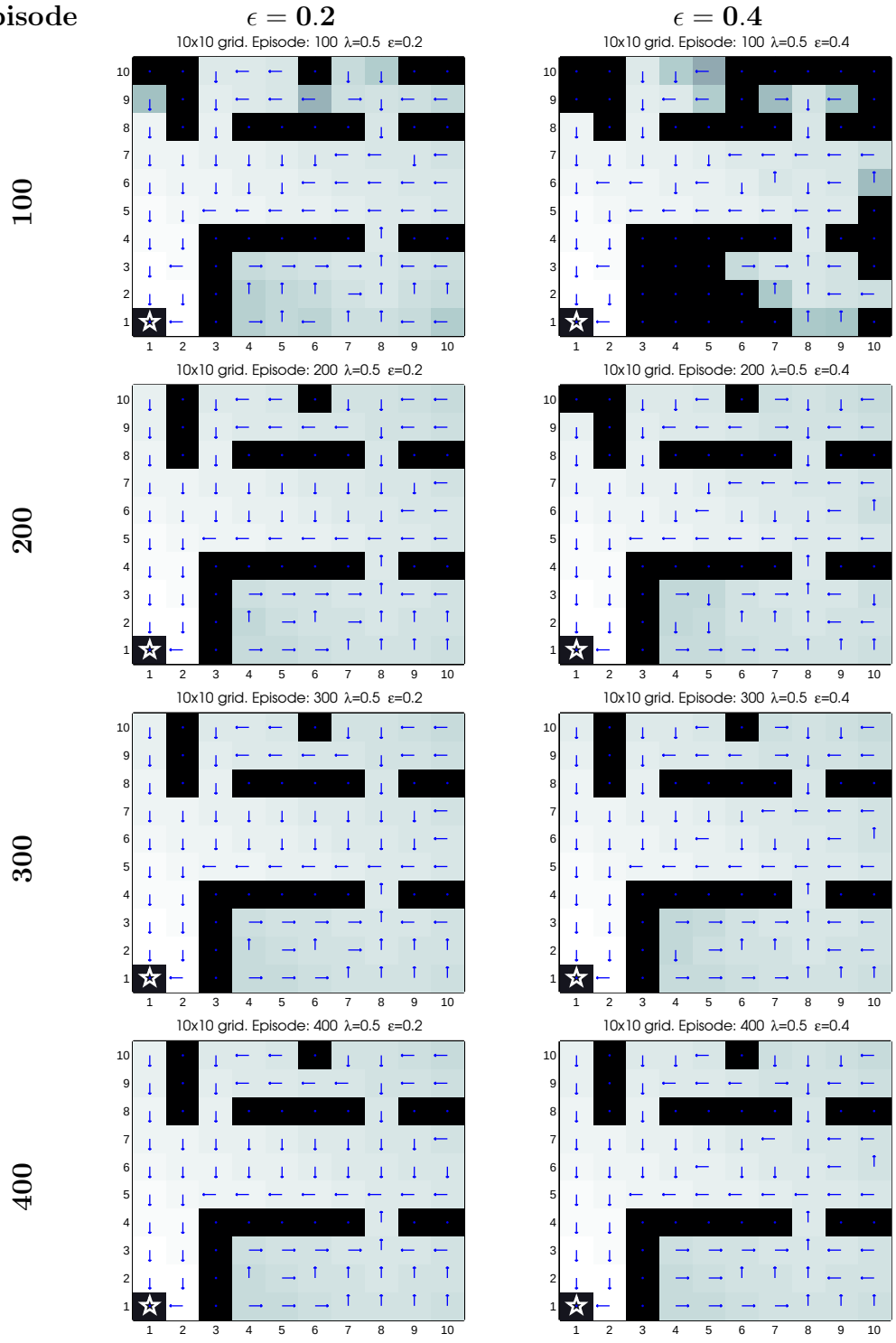
Perhaps the most important tradeoff in any learning algorithm is the exploration v. exploitation tradeoff. The tradeoff refers specifically the the behavior policy, which guides the learning process. An exploitative behavior policy is said to take actions that are "greedy with respect to the value function" ie. calculated as in Equation 3.13. Greedy policies exploit what has been learned over what has yet to be discovered. On the other hand, an exploratory behavior policy will not take the greedy action, valuing exploration over learning more about those states that have already been visited. Particularly in the maze-world problem, where local-minima in the value function are possible, it is necessary to take exploratory actions some percentage of the time.

The parameter governing the exploration versus exploitation is denoted as ϵ, and has the interpretation here of the probability of taking a greedy (exploitative) action. ($1 - \epsilon$ is the probability of taking a random action). For $\epsilon = 1$, the behavior policy is Equation 3.13 with probability 1. For $\epsilon = 0$ a random action is always taken. A test problem is proposed in Example 4. Learning results for varied ϵ are shown in Solution 4.

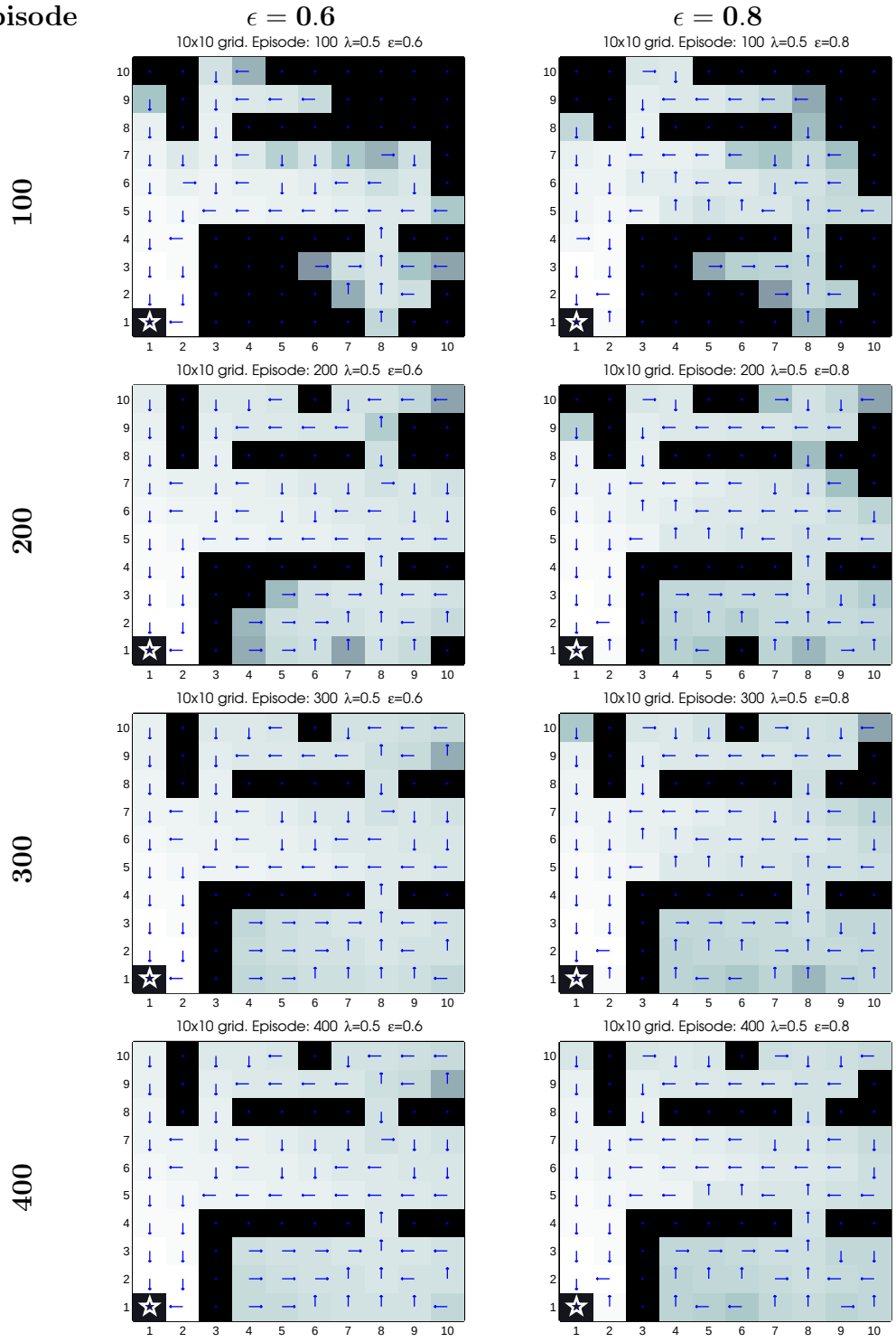
Example 4 (Maze World: Exploration versus Exploitation)**.**

This test case explores the relationship between learning speed, end policy quality and variable exploration probability, ϵ. All properties are as defined in the maze world problems in Example 2 and Example 3 unless otherwise specified.
The optimal policy π^* is solved for over successive learning iterations by Q(λ) with fixed $\lambda = 0.5$. Policies are learned for $\lambda = [0.2, 0.4, 0.6, 0.8]$. Results are given in Solution 4.

Solution 4 (Maze World: Exploration versus Exploitation).

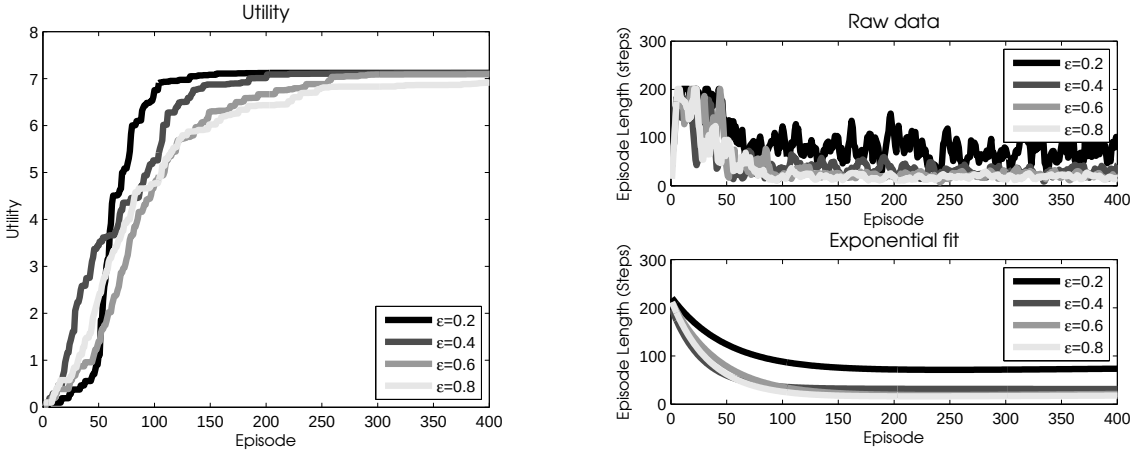


Solution 4 (Maze World: Exploration versus Exploitation).



The learning results in Solution 4 indicate the intuitive trend that a more exploratory behavior policy covers a larger swath of the state space earlier in learning. The utility trend in Figure 3.8a shows that as ϵ decreases, aggregate state value increases. All states have been visited even after 100 episodes for $\epsilon = 0.2$. Surprisingly, it would appear that after 200 episodes, the lower ϵ examples have learned a comparable if not better policy.

The learning curve in Figure 3.8b shows that more exploratory behavior policies take longer to reach the goal. In fact, the steady-state episode length value for $\epsilon = 0.2$ is appreciably higher than other policies. This trend is not indicative of policy quality, and in fact indicates that a nearly random behavior policy can in fact learn the best target policy.



(a) Utility

(b) Learning Curve

Figure 3.8: Learning results for varying ϵ.

Policy test results in Figure 3.9 indicate the trend implied by the learning results: more exploratory behavior policies produce the best ultimate target policy.

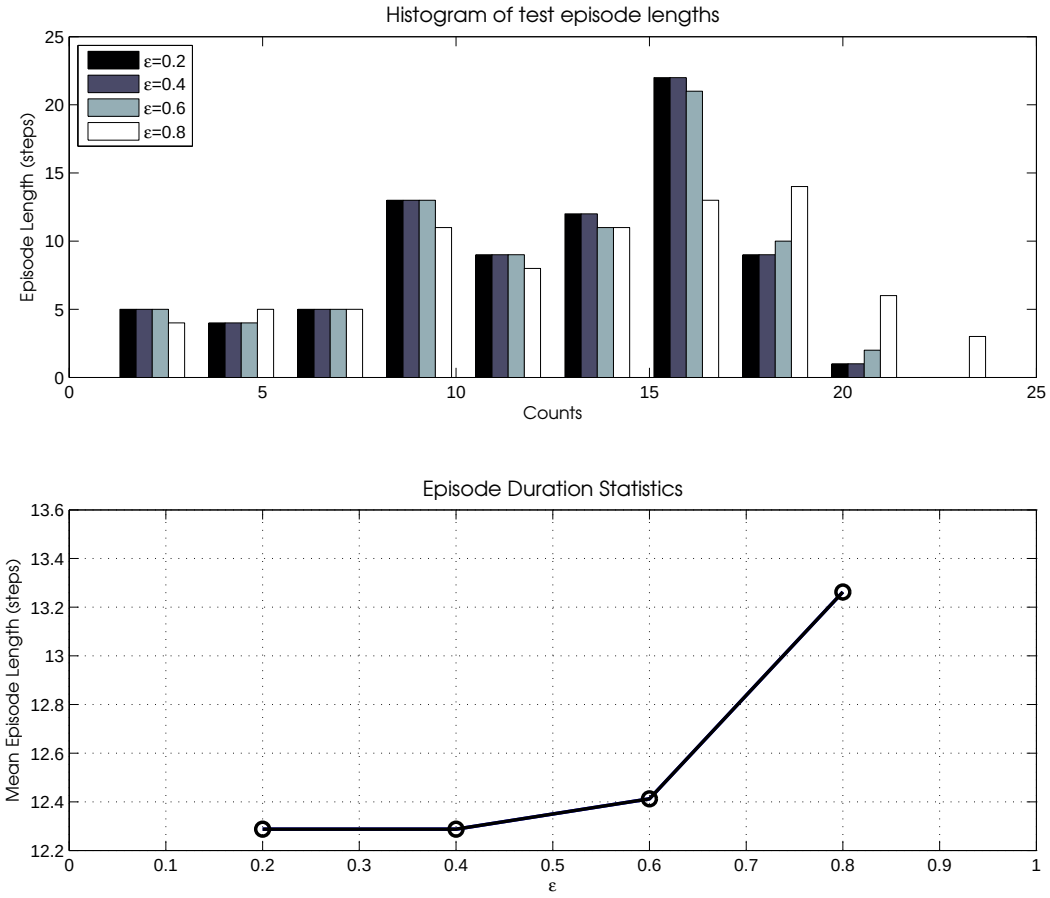


Figure 3.9: Policy test varying exploration ϵ-greedy policy for 10x10 maze world.

3.5.4 Prioritized Sweeping

Prioritized sweeping is an essential technique for learning problems with vast state spaces. Many learning algorithms hinge on the principle that convergence to an optimal value function can be achieved by visiting all states infinitely often. Several factors complicate complete state coverage in practice.

- Visiting all states often enough such that the Bellman error is close to zero requires longer learning times as the size of the state space increases.
- In many problems, certain states are more important to learn than others, such as those that are close to the goal along common state trajectories.

- Learning can achieve faster and better results by visiting states in proper sequence such that value is effectively propagated back from goal to start. In large problems and especially at the early stages of learning, unprioritized state visitation can result in exceedingly long learning times.

A priority queue is an ordered list of states or state-action pairs that stand to gain or loose significant value. Essentially, the priority queue lists the states in order of maximum learning or information gain. Priority is calculated after each state update as follows in Equation 3.29.

$$p = |r(s', a, s) + \gamma \max_a Q(s', a) - Q(s, a)| \tag{3.29}$$

A basic priority queue implementation is given in Algorithm 3.3. Other implementations exist, such as the one in [66]. The algorithms differ in how the queue is maintained and which states are updated. In Algorithm 3.3, only the top state to popped off the queue to initialize the next learning iteration, rather than emptying the entire queue at each update. The queue is then updated at every learning iteration. Since the convergence and optimality of Q(λ) and other learning algorithms are not affected by state visitation, other schemes for effective state visitation can be applied to address problem-specific considerations.

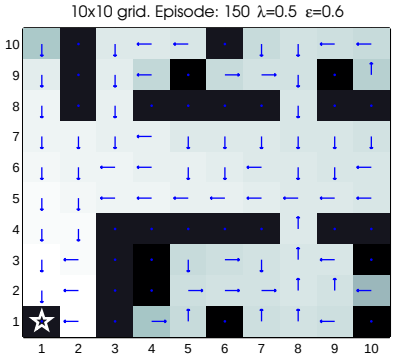
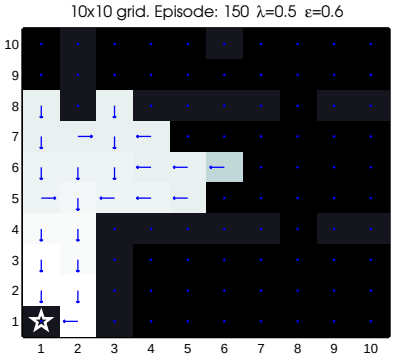
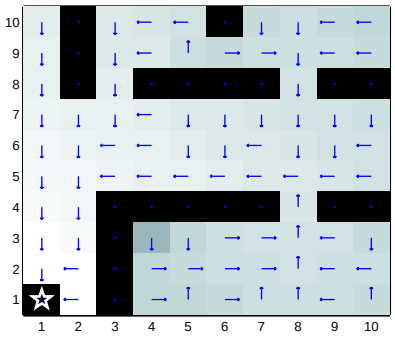
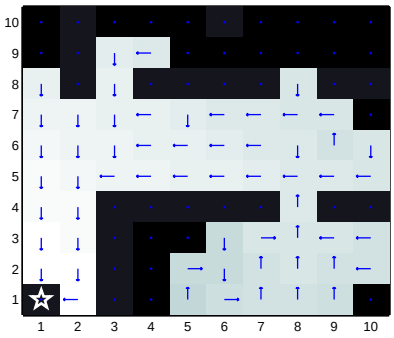
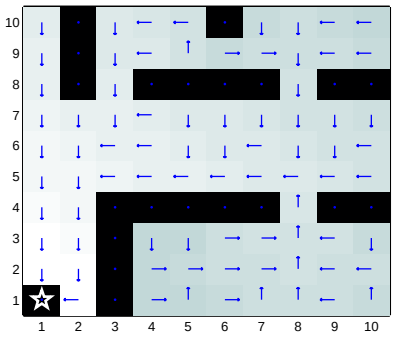
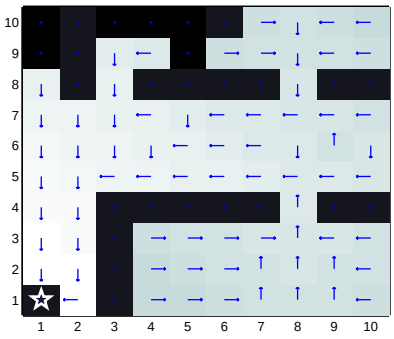
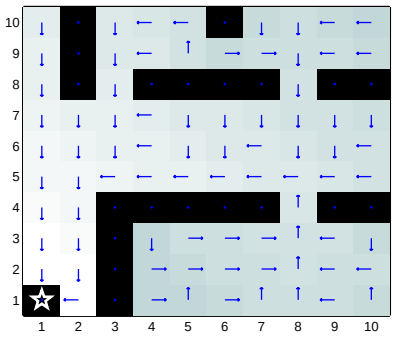
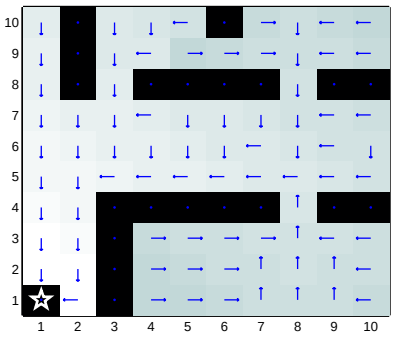
Algorithm 3.3 Basic Prioritized Sweeping

```
Initialize pQ = zeros(nS, nA), p_min = priority minimum
while Learning do
    function INITIALIZE(s_0)
        Pick initial state: s_0 = arg max_s max_a[pQ(s, a)]
        Reset priority: pQ(s_0, :) = 0
        return s_0
    end function
    function UPDATE(pQ, Q')
        Choose action a by any on or off policy method
        Take action a, observe s' and r
        Choose action a' based on s'
        Calculate priority: p = |r + γ max_a Q(s', a) − Q(s, a)|
        if p > p_min then
            pQ(s, a) = priority
        end if
        Update Q(s, a) by any learning algorithm
        return pQ, Q'
    end function
end while
```

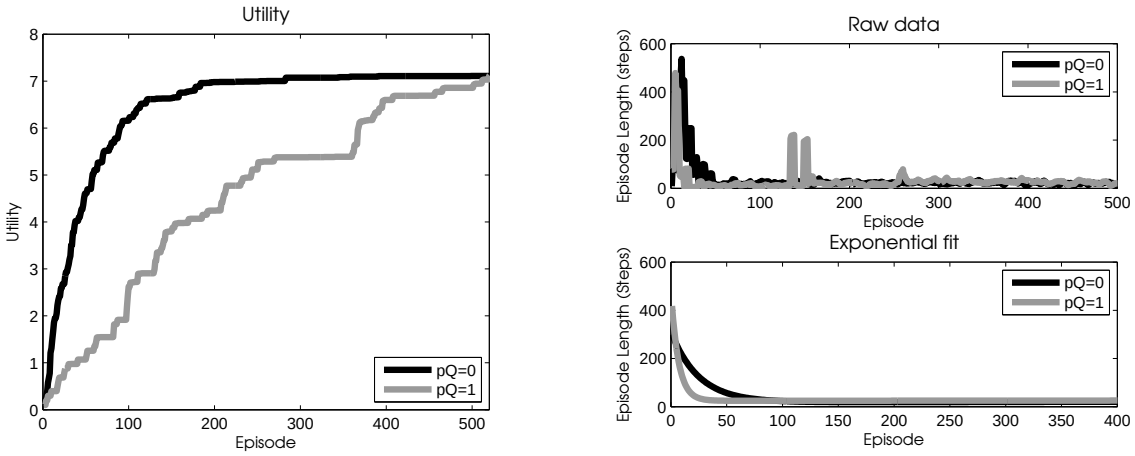
Example 5 (Maze World: Prioritized Sweeping)**.**

In this example, a priority queue is implemented to achieve faster convergence to the optimal policy on the maze introduced in Example 2. The parameters controlling eligibility trace length and exploration are fixed at $\lambda = 0.5$ and $\epsilon = 0.6$ respectively, based on experience gained in Example 3 and Example 4. The priority queue is calculated according to the method described in Algorithm 3.3. Results are gathered for both standard random state initialization and initialization via the priority queue.

Solution 5 (Maze World: Prioritized Sweeping).

Episode	Priority queue off	Priority queue on
150		
300		
450		
750		

Solution 5 highlights how learning progresses under prioritized sweeping. When the queue is used, value propagates back from the goal, as indicated by the grid cell shading. The top and bottom rooms are the last areas explored, as they are farthest from the goal. In this relatively small example problem, it may appear that the priority queue does not add much value, and in fact makes learning progress slower: it takes 750 episodes to explore all the states under the priority queue, whereas the entire space is sufficiently explored after 300 episodes without the queue. Where the priority queue stands out is in policy quality–the learned policy is often correct earlier, whereas the random state space search must visit the same state multiple times before the correct policy is learned. This policy quality effect is also evident in the learning curve in Figure 3.10b.



(a) Utility (b) Learning Curve

Figure 3.10: Learning results for prioritized sweeping.

For $pQ = 0$, the priority queue is off and initial states are selected at random. For $pQ = 1$, the priority queue is on, and initial states are selected according to priority.

Figure 3.10 provides more insight into learning with a priority queue. The utility plot in Figure 3.10a shows that without the priority queue, more states gain value sooner, which is also the trend seen in Solution 5. When using the queue, value seems to be added in a more linear and less exponential fashion, with both methods achieving the same utility at the conclusion of learning.

The benefits of prioritized sweeping are clearly seen in Figure 3.10b. Despite the irregularity in utility growth, the learned policy is better earlier on in learning, and both policies achieve the same optimal after several hundred learning episodes. Prioritized sweeping should be of even more benefit as the state space becomes larger and the learning time increases.

3.6 Conclusions

This chapter introduces the basic principles of reinforcement learning and several popular variants. The learning approach, in contrast to methods from control theory, solves decision problems through experience. Experience comes in the form of multiple simulated examples. The example decision problem introduced in this chapter is a simple 10x10 grid maze navigation task. Though basic in concept, the problem provides a standard backdrop for understanding several popular learning extensions. Eligibility traces are introduced as a bridge between pure Monte Carlo and pure single-step temporal difference learning. The test results show that some blend of the two produces the best learning results on the maze world example. The classic exploration versus exploitation tradeoff is also explored through varying randomness in the behavior policy. Test results show that a high degree of random exploration produces both the best learning and end policy quality. Last, a basic prioritized sweeping method is implemented as a means to guide state space exploration. Particularly in large learning problems, completely unguided exploration of the state space can result in prohibitively long learning times. Initializing states from a priority queue benefits even the 100 state maze problem, and likely will benefit the larger problems introduced in subsequent chapters.

One of the most significant developments in learning research introduced in this chapter is temporal difference learning. Algorithms such as Q-learning and the more general $Q(\lambda)$ combine favorable aspects of dynamic programming and Monte Carlo learning. All of the examples in this chapter are solved using some variant of $Q(\lambda)$ learning. Temporal methods, most of all, do not require a state transition model, a feature of great significance for complex learning problems, such as the GPP that will be introduced in Chapter 4.

Chapter 4

Learning and The Generalized Planning Problem

Machines take me by surprise with great frequency. - Alan Turing

4.1 Introduction

The Generalized Planning Problem was introduced at the conclusion of Chapter 2 by way through the taxi routing problem in Example 1. Chapter 3 describes the process of reinforcement learning and several learning algorithms in the context of a simple maze navigation problem. The is small and simple by design to showcase learning algorithm variations. Applying this experience to a larger problem, this chapter focuses on posing and solving a routing problem as a GPP. Routing, addressed Chapter 2, not typically solved by learning-based methods. In order to do so, subsequent sections address the following questions:

Why is learning a viable solution method to complex spatiotemporal decision problems. (Section 4.2)

How can the GPP be represented? What is the state? How does it scale? (Section 4.4)

What underlying model (Section 4.5) *and learning techniques* (Section 4.7 *can solve the GPP?*

Why should human input be included and how can it be done in a structured manner? (Section 5.1)

*How is a GPP implemented in a larger hybrid human-machine system*Chapter 5.

4.2 Motivation

The benefits of learning as a method for obtaining control policies have been highlighted in Chapter 3. In the context of the GPP, the benefits of and motivation for a learning approach can be summarized as follows:

- **Modeling**: A GPP such as a constrained stochastic routing problem may have high-dimension, nonlinear, and generally difficult to model state transition dynamics. However, simulation is often much easier than explicit modeling. Modeless TD-approaches introduced Chapter 3 can compute optimal policies without an explicit transition model.
- **Randomness**: Learning directly addresses randomness by allowing nearly every problem characteristic to be random so long as they can be simulated. Through experience based learning, randomness is automatically included in learned policies.
- **Temporal Abstraction**: Semi-Markov learning methods previously discussed closely model the reality of scheduling and routing problems: inherently discrete actions (eg. assignments/allocations) take place over (possibly random) time periods, and can be interrupted at any time without consequence to the overall learning problem.
- **Flexibility**: Learning affords the problem designer direct control of high-level behaviors through the reward function.
- **Human Input**: A large body of learning research exists in learning from humans and learning about humans, including supervisory learning and inverse reinforcement learning.

Despite advantages, several classic pitfalls of learning require attention when posing the GPP as a learning problem.

- **Size** The exponential explosion of the joint state-action space is a consideration for any decision problem and associated solver. This so-called curse of dimensionality in the context of the GPP demands judicious choice of the underlying state such that increasingly large problems remain tractable.
- **Tuning**: A blessing and curse of learning is tuning. Parameters such as discount factor, learning rate, eligibility trace weight, and others allow tight control over the learning process. However, tuning these parameters is often opaque and problem-specific.
- Tuning: (might need to do some more research here) $Q*$ is optimal reward function wrt the reward, but was the problem defined appropriately to begin with.

- **Function Approximation**: A survey of current learning literature indicates that many of the meaningful problems are inherently large, and the only hope of solving them lies in function approximation techniques. However, several theoretical holes, specifically in the area of function approximation and TD-learning with eligibility traces, require further research.

4.3 Routing Problems and Learning

This chapter focuses on the intersection of decision and learning. The specific context is multi-agent generalized planning, which includes well-known problems from scheduling, routing, and generalized assignment. These types of problems have gained popularity in the learning community because they present several challenges: characteristically large state and action spaces, state and/or action hierarchy (further discussed in Section 4.6), and uncertainty at many levels of abstraction.

Several researchers have studied advanced learning techniques in the context of single agent assignment. Sutton et al. has applied a temporally abstract SMDP model to a single agent routing problem with uncertain environmental elements [67]. Theocharous, Rohanimanesh, and Mahadevan has proposed a hierarchical POMDP model for single-robot navigation [69, 68].

Adding a layer of complexity, many researchers have studied multi-agent decision making; Mathews, Durrant-Whyte, and Prokopenko provides an excellent overview of decentralized decision making by optimization-based methods [42, 43]. Sugawara and Lesser proposed a learning based framework for cooperative decision making with an emphasis on assessment of network health in a decentralized manner [62]. In both cases, the authors identify the problems with decision making in vast networks, where the number of states and joint actions increases exponentially with the number of decision points.

Directly addressing the task allocation problem, Dolgov and Durfee proposes a reduced "loosely-coupled" MDPs model for problem size reduction. The coupling constraints are the tasks themselves. The solution is based on decoupling and reducing the problem to a set of smaller mixed-integer linear programs [18]. Abdallah and Lesser addresses the same multi-agent task allocation problem, but proposed a learning-based solution based on a SMDP model. [2]. Perhaps the most abstract problem, Midtgaard et al. propose a learning-based solution to a real-time strategy game, modeled as a SMDP Midtgaard et al.

All research in the field of multi-agent decision making must address the problem of problem size reduction. Guestrin, Koller, and Parr addresses learning and the so called curse of dimensionality with a factorized MDP specifically in the multi-agent planning context [28, 29]. Makar, Mahadevan, and Ghavamzadeh applies the MAXQ framework to multi-agent learning by using intrinsic hierarchy to decompose the problem into subtasks [39].

Thought not directly addresses here, some research has focused on how to even evaluate performance of a multi-agent decision engine. Metrics are mainly borrowed or adapted form legacy TSP and VRP literature. Challenges arise in evaluting problems with a high digree of

uncertainty, for which optimal solutions may not exist in the general case, or would be over laborious to calculate. Particularly of interest here is evaluating the performance of hybrid human-machine decision making. Crandall and Cummings has developed several metrics in the context of supervisory control, where the human is the supervisor [14]. In many cases, performance metrics will be problem specific.

4.4 Representation

The question of representation is of central importance to any problem. It determines the underlying state on which a problem is defined. Representation is also a matter of perspective, where multiple perspectives are possible. Points in space can be represented in polar or cartesian coordinates, for example, each with their own benefits and disadvantages in different circumstances. Representation can also be viewed as a parameterization, with certain parameterizations that are more compact than others. The Fourier basis, for example, is a parameterization that is extremely compact for periodic signals. Gaussian distributions are completely parameterized by a mean and a variance.

The central question of this thesis can be stated as follows: *Does their exist a compact representation, parameterization, or basis for the GPP, as described in Definition 6*? Many representations may exist, and the challenge for the problem designer is to find one that is expressive enough to capture important problem details, but not so granular that it becomes intractable for marginal expressive benefit.

4.4.1 Representational Requirements

Formally, representation refers to the state on which a problem is defined. The notion of representation is expanded here to also include actions, and conditions and constraints on both. The specific use case in mind here is a hybrid routing/scheduling problem such as the taxi problem described in Example 1. The representation must therefore be able to capture the following properties and behaviors.

- A notion of agents, tasks, environment and all associated properties therein.
- Deterministic or stochastic agent, task, and environment dynamics.
- Distinct agent task costs attributed to their properties (capacity, for example).
- A discrete or continuous state set.
- A notion of start, intermediate, and goal states.
- A means of deterministically or probabilistically modeling or simulating state transitions.
- A discrete or continuous action set.

- Low-level actions associated with locomotion, sensing, or other primary behavior.
- High-level macro actions such as "allocate" and "deallocate".
- A means of deterministically or probabilistically modeling or simulating the translation between high-level actions and low-level actions.

With inspiration from the rich body of routing, scheduling, and assignment literature, the central representational feature used here is the cost matrix $\mathbf{C}$ as described in Definition 4 and Definition 6. Unlike in standard routing and scheduling problems, however, the notion of cost here is expanded to encapsulate all aspects associated with an agent or set of agents performing a particular task.

The cost for each agent i to perform each task j (c_{ij}) can be aggregated into the cost matrix $\mathbf{C}(x,a)_{ij}$ in Equation 4.1. The cost matrix is a function of the state and action (x,a). The total cost is a sum of three components: the cost associated with just the task regardless agency $J_j(x,a)$, the cost associate with just the agent regardless of task $J_i(x,a)$, and the cost associated with the specific agent-task pairing $J_{i,j}(x,a)$.

$$\mathbf{C}(x,a)_{ij} = J_{ij}(x,a) + J_i(x,a) + J_j(x,a) \tag{4.1}$$

$$J_{ij}(x,a) = S(i,j) + T(i,j) + P(i,j) \tag{4.2}$$

$$J_i(x,a) = S(i) + T(i) + P(i) \tag{4.3}$$

$$J_j(x,a) = S(j) + T(j) + P(j) \tag{4.4}$$

Function	Example
Joint Costs (J_{ij})	
$S(i,j)$	euclidean distance between two points, difficulty score
$T(i,j)$	agent-task rendezvous, agent-task specific deadline
$P(i,j)$	specialization - certain tasks require certain agents
Agent Costs (J_i)	
$S(i)$	absolute agent position ("no fly zone")
$T(i)$	agent-specific deadline
$P(i)$	agent capacity
Task Costs (J_j)	
$S(i)$	absolute task position (high spatial priority tasks)
$T(i)$	task timeout, reoccurring tasks
$P(i)$	fixed pickup cost, priority

Table 4.1: Cost function examples.

The state of a GPP is defined on these spatial, temporal, and parameter dimensions. Problem elements that are typically represented as constraints in a traditional VRP, for example, are folded into the state here.

The joint state action space can be divided into three primary dimensions: spatial, temporal, and constraint. For notational brevity, each element of the cost is expressed as a sum of these three dimensions as in Equations 4.2, 4.3, and 4.4.

4.4.2 Constraint Incorporation

Differing from traditional optimization based approaches, constraints here are folded into the state and captured in the cost function, and its analogue, the reward function. In certain cases it may be unclear whether to incorporate a desired problem dynamic or behavior into the state or the reward. For example, in the taxi problem in Example 1 the initial cost of a taxi, $\$1,000,000$ can be considered as a negative reward in every state, or as part of the "cost state" itself. The approach taken here is to push problem constraints into the state, and learn the optimal policy through successive simulation Figure 4.1 shows how spatial, temporal, and parametric constraints are incorporated directly into the state of the problem.

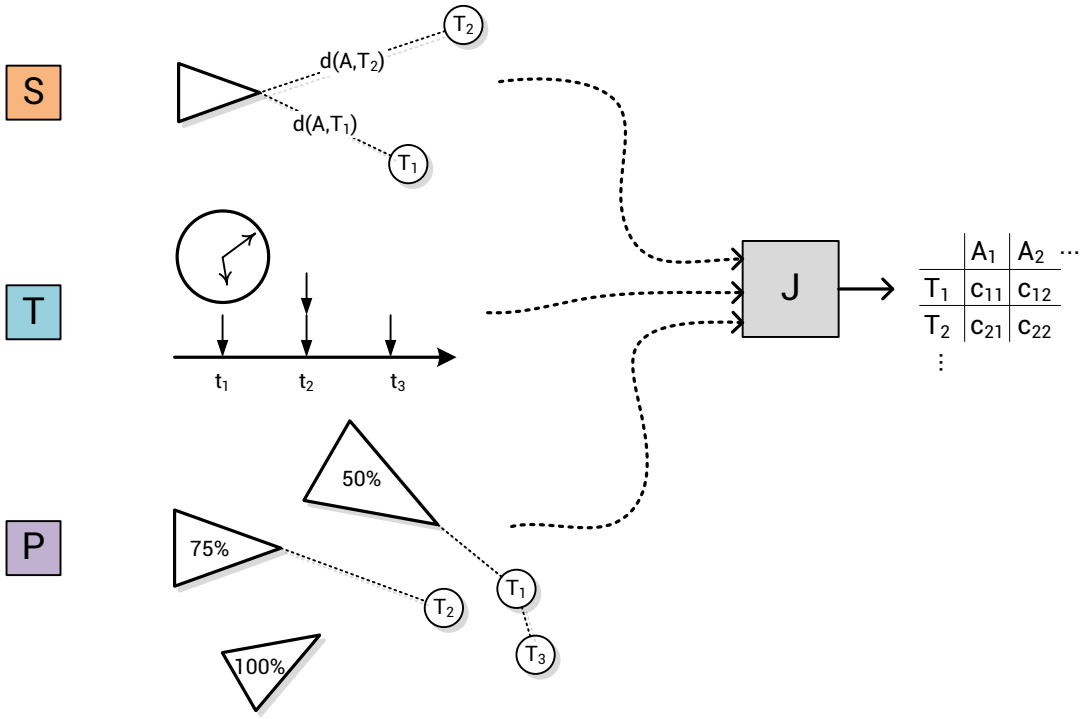


Figure 4.1: Spatial, temporal, and parametric constraint cost-based representation.

Rather than explicitly enumerate all state and action constraints, the proposed method transfers constraints to the state itself. This transfer of complexity allows for great flexibility but places the onus on the problem designer to define appropriate task costs based on the constraints.

Constraint incorporation has several potential benefits over traditional constraint enumeration. By incorporating the constraints directly into the state, certain infeasible states may be identified and disregarded, reducing the size of the learning problem. Additionally soft constraints are possible to express simply by scaling the cost A task deadline, for example, can be expressed as a temporal step function where the cost goes from zero to infinity at a certain time. A soft deadline can then be expressed as a linear or exponential ramp in cost over time.

Constraint incorporation–the folding of state and action constraints directly into the state space itself–promotes an abstracted, highly flexible, cost-based view of the world. Much like in optimization-based methods, the onus is on the problem designer to come up with realistic costs. Complexity is therefore effectively transfered from constraints to the state, and functions on the state such as the reward function, transition model, or simulator if one exists.

Important considerations include the resolution of the cost variable. Since cost takes on all meaning in a problem, it must be granular enough to capture relevant problem dynamics and differentiate between states.

4.5 Modeling

Representation provides a perspective by which a problem can be defined . Modeling ties the perspective to a particular mathematical formalism for solution (Section 4.7). For the GPP

Most of the modeling requirements follow directly from the representation requirements listed in Section 4.4. Based on the requirements, the benefits of learning highlighted in Chapter 3, and the generality of graphical models, Markov decision based models are used here. As defined in Definition 7 MDPs require a state set, action set, transition model, reward function, and additional parameters depending on the learning algorithm. For the GPP, an MDP model can be formulated as follows.

State The state of the GPP is essentially the cost matrix discussed in Section 4.4. At a minimum it must contain agent-task costs, but may also include agent-agent costs, and, in the case of routing, task-task costs, which are required to optimal vehicle routing solutions. [35, 34].

Action Actions are generally agent specific: the actions for a simple robot as in Example 2 are up-down-right-left, whereas the actions for a server that is being scheduled in a JSP would be to take a certain job over another. As such, defining a general action set that will work for all GPPs is essential. The action set chosen here is the joint set of all possible agent-task pairing, and are referred to as "allocations" herein. While exponential in the number of agents (n) or tasks (m), the accessible set can be vastly reduced by eliminating infeasible actions, such as disallowing allocations to completed tasks, and allowing only one agent per task, for example.

Transition Model A transition model is required for any non-temporal difference learning method (subsection 3.4.3). This requirement can present challenges for a GPP where state transitions may be difficult, if not impossible to explicitly model. It is for this reason that modeless approaches are proposed as a solutions means for GPPs such as vehicle routing and scheduling.

Reward Function The most significant and least straightforward modeling element is the reward function. The reward function essentially encapsulates the goal of a problem. The most common reward function for any learning problem is $R(s_g) = +1$ which assigns a value of $+1$ to the goal state, s_g, and zero everywhere else. However, in many learning problems, particularly those with subgoals, long distances between start and goal, and complex dynamics, a simple reward function will not suffice. As such, the GPP reward is a function of the current state, current action, and successor state: $R(s', a, s)$. The primary goal, in all cases, is to accomplish all tasks. Other rewards can be added, Example 6 provides an example of a complete reward function for a learned VRP.

Parameters The standard MDP model includes one parameter, the discount factor (γ), which guarantees convergence for non-episodic tasks. Other parameters, such as learning rate (α), are both learning algorithm and problem specific. Table 4.4 summarizes all the parameters for a learned VRP.

The standard MDP model provides a good basis for evaluating different representations and learning algorithms. However, due the inherent size of the state and action spaces and poor scaling properties of the GPP, alternate methods need to be considered for anything other than toy problems and test cases. The following sections addresses hierarchical models, which present huge representational benefits as well as potential problem size reductions.

4.6 Hierarchical Models for Inference and Decision Making

The examples in the previous section have demonstrated the flexibility of the reinforcement learning framework. High-level tuning parameters provide the problem designer with intuitive means for controlling learning speed and end policy quality. The 10x10 maze world setting is simple and small by design, so as to exhibit parameter effects. However, many problems have vastly more states, more complicated dynamics, and more complicated environments. Often, the process of determining a tractable state space represents the vast majority of effort put into a learning problem.

As an extension to one-layer models such as the MDP, Hierarchical models capture the multi-resolution nature of of perception and action in decision making. The models discussed in the following section each exploit either state hierarchy, action hierarchy, or both to achieve more efficient representations. Several reasons why hierarchy may be useful in RL in particular are as follows. Barto and Mahadevan provides an excellent overview of learning with hierarchical models [4].

- State hierarchy can be used to capture problem structure, as in natural language processing, where sentences, words, phones, and letters each live on different levels of hierarchy/abstraction [46].
- Partial observability can also be viewed as state hierarchy, where the observed state (observation) is some noisy projection projection of the true state.
- State approximation methods in which the observed state is a lower dimension projection of the true state can also be represented as state hierarchy, such as in HMMs and Hierarchical Hidden Markov Models (HHMMs).
- Action hierarchy can be used to represent temporally abstract actions, as in SMDPs [63].

- Incorporating hierarchy in models has been shown to improve inference performance over the equivalent "flat" model in some cases [48].

Several of the most popular graphical models are shown in in Figure 4.2. The most basic Markov process model is shown for reference. Models can broadly be categorized as autonomous, active, partially-observable and fully observable. Autonomous models have no actions, and the state is said to progress forward in time "autonomously". So called "active" models such as the MDP incorporate the notion of actions driving the state forward in time. State observability is categorized as full when the state of interest in available to the user, and partial when user only gets to see observations. Observations are often categorized as noisy projections of the true state in state estimation applications. In these cases, the user only gets to see a sensor reading (eg. laser range scan), and must infer position based on a sensor model, or observation model. However, in higher level application such as those that arise in planning and allocation domains, observations may take on the interpretation of a lower-dimension projection of the "hidden" state. In these cases, the hidden state may in fact be directly measurable, but due to computation and storage constraints, only the observed state is considered as a simplification.

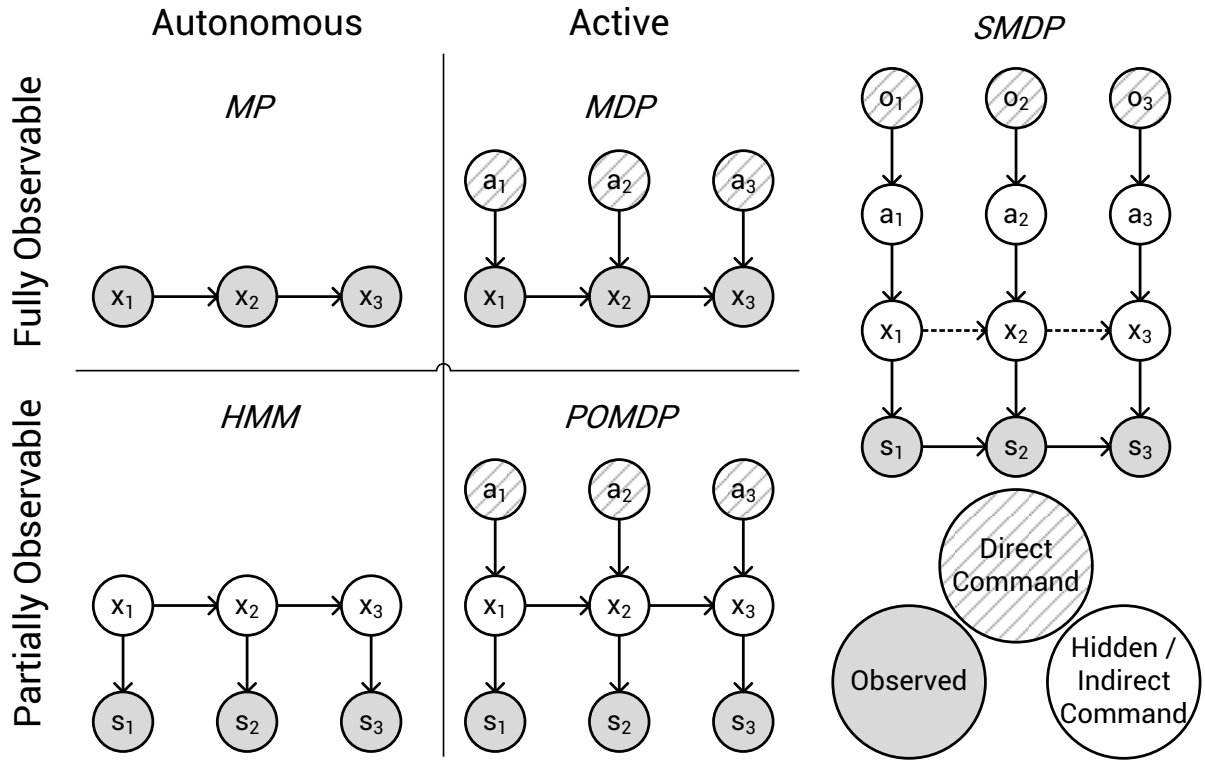


Figure 4.2: Graphical models: autonomous, active, observable, and partially observable.

Shaded nodes indicate observed states, hatched nodes are actions directly commanded, versus un-hatched action nodes, which represent consequent or lover level actions. Unshaded state nodes are unobserved.

4.6.1 Hierarchical Hidden Markov Models (HHMMs) and Dynamic Bayes Nets

A Dynamic Bayes Net (DBN) is a standard Bayes net, expanded to represent sequences. Perhaps the best known DBN is the HMM, shown in Figure 4.2. HMMs have one hidden level, while HHMMs expand on the include multiple hidden levels. As with many graphical models the goal of HHMMs is efficent inference of hidden nodes. Full conditional probability distributions can be defined for a general HHMM and are given in [46]. Closely related to the SMDP in Figure 4.2 is a specialization of HHMMs referred to as hidden semi-Markov model (HSMM) [47]. This HSMM should not be confused with a SMDP, in which actions are semi-Markov, not states. Nevertheless, the HSMM provides some insight into how one would construct a graphical model that would account for both abstract (hierarchical) states and actions.

4.6.2 Abstract Hidden Markov Models

Closely related to HHMMs, we next consider Abstract HMMs [9]. AHMMs arise in applications where we would like to infer the policy an agent is following. The "state" captured in an AHMM is a (possibly hierarchical) policy, which results in an action, is applied to the system state, and results in an outcome/observations. AHMMs are used in setting where, given an observation sequence and observation probabilities, the goal is to infer variables that parameterize the policy.

4.6.3 Dynamic Abstraction Networks (DANs)

The authors of [41] present a small but notable augmentation to HHMMS and AHMMs, which they refer to as DANs. A DAN is a AHMM, which encodes policy abstraction, and a HHMM which encodes state abstraction. The joint representation gains some expressive power over AHMMs by encoding multiple state abstraction levels, the guiding principle being that abstract states are useful for learning abstract policies.

The authors of [41] compare DANs and AHMMs by 1) using expectation maximization to learn model parameters given an observation sequence and appropriate emission probabilities, and 2) applying the learned model to the RL Taxi problem in [17]. The learned policies are then compared. The results show some empirical learning advantage for DANs over AHMMs, though the advantage is not great, and EM takes longer in DANs due to a greater number of parameters.

4.6.4 Hierarchical Abstract Machines (HAMs)

HAMs, first introduced by [52], are yet another way of adding hierarchy and abstraction to traditional flat MDPs. Hierarchy of Abstract Machiness (HAMs) provide hybrid control scheme, where a state machine switches between lower level controllers; in HAMs the low level controller is an MDP and the supervisor is a SMDP. Resulting policies are hierarchies of finite state machines. The theoretical results of Parr and Russell show that the composition of MDP M and HAM H, $H \circ M$, is an MDP, and that optimal policies in $H \circ M$ are also optimal for M [52].

The theorems in [52] essentially say that solving a reduced problem inferred from the HAM hierarchy is equivalent (and more efficient because there are less states and actions) to solving the original MDP. This is the key reason why learning policies for SMDPs can present great efficiency gains over solving the original MDP. It is an intuitive result that introducing high-level (temporally abstract) actions, essentially ignoring lower level dynamics, can make previously intractable problems tractable (It may be impossible to get to the grocery store if you are forced to enumerate a sequence of joint angles for your limbs, as opposed to several well-placed intermediate waypoints).

4.6.5 MAXQ Value Function Decomposition and Task Graphs

MAXQ, developed by [17] also hinges on SMDP option-based planning theory. Unlike HAMs however, MAXQ methods do not restrict the resulting model to a be single SMDP. Rather, complex tasks are decomposed into a set of nested MDPs. Corresponding to the task hierarchy, a decomposed value function can also be defined, and a RL problem can be posed that solves subtasks simultaneously. The MAXQ model is significantly different from those previously presented, as the authors are focused on creating optimal hierarchical policies, rather than using inference techniques to compute distributions of hidden variables in the GM.

4.6.6 Partially Observable Models

POMDPs are notoriously intractable problems to solve. Barto and Mahadevan indicates that hierarchical POMDPs such as those in [68] may possess some expressive benefits over planning with traditional POMDPs [71]. In most cases, POMDPs increases the state space to intractable dimensions and are therefore of limited application in real planning applications without considering simplifications, such as parameterization [8].

4.6.7 Temporal Abstraction and Semi-Markov Decision Processes

Many decision problems are modeled as Markovian: the past and future are conditionally independent given the present, and successor states are produced from origin states by applying an action over exactly one time step. However, many problems have actions that inherently occur over multiple time steps. Even the simplest of day-to day decision problem involve a hierarchy of actions: A high level action (eg. *Walk to the door.*) begets a lower level action (eg. *Move forward one step.*), and possibly still an even lower level action (*eg. Move left foot in front of right.*). Indeed, an infinitely fine resolution of subactions may be considered. The SMDP model, formalized by Sutton, Precup, and Singh, provides a means for expressing hierarchical actions and temporal abstraction in the context of MDPs.

The focus of this thesis is on hybrid scheduling-routing decision problems, termed Generalized Planning Problem. The Semi-Markov Decision Process model has been chosen to represent the GPP for the following reasons.

- SMDPs are a minimal extension of the classic MDP model to include multi-step actions, protracted in time, for a possibly random amount of time. Temporally extended actions underpin many real decision problems.
- Much of the underlying MDP theory applies to SMDPs.
- SMDPs present a potentially huge reduction in problem size by considering macro actions (referred to as options), rather than traditional primitive actions.

- On and off policy learning algorithms that compute optimal policies on MDPs have been extended to compute optimal policies on SMDPs.
- The temporal difference SMDP Q-Learning off-policy temporal difference algorithm (SMDP-Q) algorithm can compute optimal policies on SMDPs without need of a transition model. Complex multi-agent planning problems such as the GPP introduced in Chapter 4 have complex underlying dynamics for which a transition model is difficult if not impossible to define.
- Figure 4.3 illustrates that the general SMDP structure (which is a HHMM in [46]) can be reduced to a one-level state and action hierarchy by considering the complex underlying MDP as hidden internal dynamics. Modeless solutions methods in subsection 4.7.1 solve for optimal policies as long as the internal dynamics are simulateable.

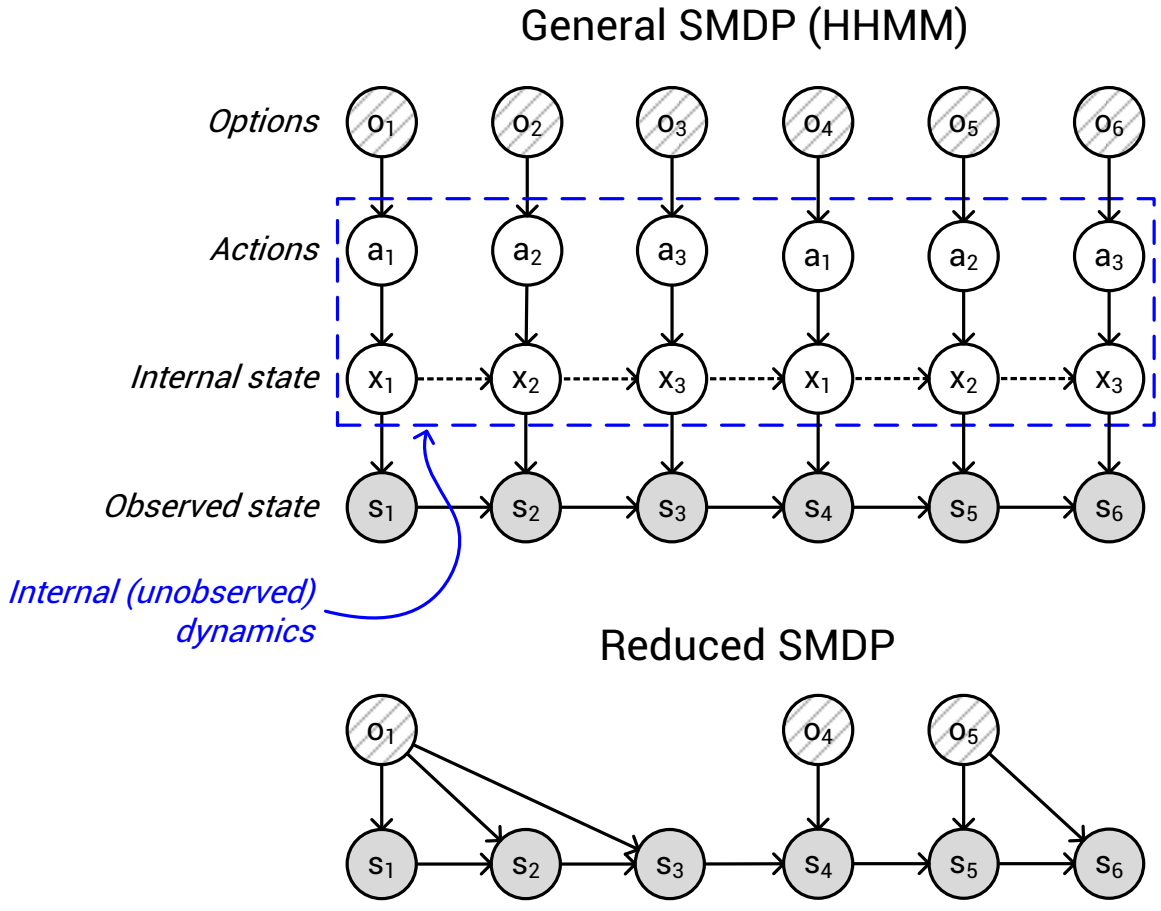


Figure 4.3: SMDP graphical model

The general SMDP model on the left may be conceptualized as the model on the right, with options acting directly on observed (high-level) states $s \in S$.

Name	**Def**	**Note**
Markov action	$a \in A$	lasts exactly one time step
Markov policy	$\pi(s, a) : S \times A \to [0, 1]$	selects actions at every time step
Termination condition	$\beta(s) : S \to [0, 1]$	Markov option termination
Markov option	$a \in A$	terminates in s_{t+1} wp. $\beta(s_{t+1})$
History sequence	$h_{t\tau} \in \Omega$	history from t to τ
Termination condition	$\beta(h) : \Omega \to [0, 1]$	semi-Markov option termination
Semi-Markov option	$o \in O$	terminates in s_τ wp. $\beta(h_{t\tau})$
Policy over actions	$\pi(h, a) : \Omega \times A \to [0, 1]$	policy over histories and actions
Semi-Markov policy	$\mu(s, o) : S \times O \to [0, 1]$	policy over states and options

Table 4.2: Relevant SMDP variables and definitions.

SMDP theory and the concept of options flows natural from first principles of MDPs. A *Markov action* is drawn from a Markov policy–a distribution over states and actions in its most general form. Markovian actions last for exactly one time step by definition. Contrastingly, a *Markov option* transfers the state from s_t to s_{t+1} and terminates there with probability $\beta(s_{t+1})$ or continues as such forward in time.

A Markov option is a generalization of classical Markov actions to allow for multi-step actions. A Markov action is still fully defined by the current state. It would thus be useful to define a setting in which a policy may look more than one step backward in time in order to determine termination. To do so, a history sequence $h_{t,\tau} = s_t, a_t, r_t, ...s_\tau, a_\tau, r_\tau$ is defined to include the full state–action–reward sequence from time t to τ. Following logically, a semi-Markov option is defined as a temporally extended macro action, stating at time t and ending at time τ in state s_τ according to the termination condition $\beta(h_{t\tau})$. Last, a semi-Markov policy $\mu(s, o)$ is defined over options and states and depends on history in the same way as a semi-Markov option. For convenience, Table 4.2 lists all of the relevant variables for defining SMDPs. A formal SMDP descriptions is provided in Definition 8.

Definition 8 (Semi-Markov Decision Process)**.**

A Semi-Markov Decision Process is specified by the tuple $< S, O, P, R, \beta >$. S is a set of states. The reward function for a SMDP reflects the total path reward received when executing option o starting in state s_t and ending in state s_τ. P is a transition function that describes the likelihood that an option terminates in a particular state. Rather than actions, a SMDP consists of a set of semi-Markov options O defined over a traditional MDP. The termination condition β determines whether an option continues or ends in a particular state.
The reward sequence is defined to be the sum of expected future rewards from the time an option begins executing until termination:

$$R(s_{t+k}, o_t, s_t) = \mathbf{E}[r_{t+1} + \gamma r_{t+2} + ... + \gamma^{k-1} r_{t+k}] \tag{4.5}$$

The state transition function is the probability that o terminates in s' after k steps.

$$P(s'|o, s) = \sum_{k=1}^{\infty} p(s', k)\gamma \tag{4.6}$$

Options are formally specified by the tuple $< I, \pi, \beta >$ where π and β are as defined in Table 4.2. $I \subseteq S$ is an initiation set, which can be the entire state space. Semi-Markov options are actions selected from a policy that depends on the history from the start of the execution of the option up to the current time, and terminating according to a termination condition that also depends on the history. A semi-Markov policy is thus a policy that selects options based on history. Key properties of SMDPs are as follows. A full list explanation of properties with proofs can be found in [63].

Semi-Markov options Options are semi-Markov because the policy that selects them and the termination condition are functions of history rather than just the previous state.

Flat Policy A policy over options μ specifies a policy over actions π, referred to as a *flat policy.*

Semi-Markov policies In the same way that options are said to be semi-Markov because they depend on history, policies that depend on history are also semi-Markov. Semi-Markov policies are also non-stationary in that they depend on events back to a particular point in time.

SMDP present several practical and advantages over traditional MDPs:

- A semi-Markov policy is inherently multi-step, allowing for macro action that take place for a random amount of time.
- Since the termination condition β depends on the history sequence $h_{t,T}$, semi-Markov policies allow for time-dependent behavior, such as timeouts, time-based penalties and rewards, as well as soft and hard time windows. MDPs, being one-step temporally myopic, do not consider time-based effects.
- While a Markov policy $\pi(s, a)$ is of dimension $S \times A$ in general, a semi-Markov policy $\mu(s, o)$ is of dimension $S \times O$. Assuming that the number of options is far less than the number of actions, the storage size may be vastly reduced.
- Traditional MDP actions are in actuality one-step options. Therefore, every MDP is representable as a SMDP
- Options are highly versatile structures, and can be changed easily by changing termination conditions, for example, without change to low-level underlying policies.

4.6.7.1 Temporal Abstraction Extensions

SMDPs amount to one of the most significant advances in decision theory, decoupling action duration from time. SMDPs, a superset of MDPs, allow for actions that can take place over one to many time steps, where duration is itself a random quantity. However, one additional extension is needed to fully generalize MDP action duration. Time-dependent Markov Decision Process (TMDP) proposed by Boyan and Littman provides this extension [7]. Recent work in the XMDP model [54] GSMDP [54] and TiMDP model [55] have further solidified the concept of temporal abstraction for Markov Decision Processes. The TMDP model proposed by Boyan and Littman in [7] is in fact equivalent to the XMDP model with the addition of a "wait" (loiter) action [54]. Table 4.3 summaries the differences between existing models in terms of action duration

Model	State transitions	Action duration	Action selection
MDP [66]	stochastic	deterministic, time invariant	$a = policy(s)$
SMDP [63]	stochastic	stochastic, time invariant	$a \sim policy(s, a)$
TMDP [7]	stochastic	stochastic, time varying	$a(t) \sim policy(s, a(t))$

Table 4.3: Temporal abstraction in Markovian decision models

Actions in the MDP case are often deterministically selected by the learned policy, whereas options, which determine low-level actions are drawn according to a distribution over states and options. It should be noted that in [7] actions are considered rather than options–for consistency, the table refers to actions drawn from an abstract policy (markov or semi-Markov).

4.7 Learning

After identifying an appropriate state for the GPP (Section 4.4), and selecting a model (Section 4.5), the final step is calculating an optimal policy through learning. Section 3.4 presents the basic theory of reinforcement learning. In the context of the GPP, two factors weight heavily into the choice of learning algorithm: size and transition model. The issue of size is partially solved through judicious choice of representation: a balance between granularity and generality is selected by condensing certain groups of "similar" states into one state. After a certain point though, a problem can be distilled no further into its principle components, and further reduction would result in loss of information. Hierarchical models and learning algorithms though can help by 1) further reducing the state or action set or 2) factorizing a problem into solvable subproblems [28]. The learning approach taken here is the former.

As identified in Section 4.5, deriving a GPP transition dynamics model may be difficult if not impossible. The probability of moving from one high-level state to the next given an MDP action or SMDP option is ambiguous, and is in itself, the part of the solution of the problem and therefore unknown at the outset. As such, the approach taken here is one of purely modeless temporal differences learning as described in subsection 3.4.3

4.7.1 Semi-Markov Decision Process Learning

As with MDPs, dynamic-programming based iterative methods readily solve for optimal semi-Markov policies on SMDPs. Of particular interest is SMDP Q-learning, which is the off-policy TD analogue to SMDP value iteration [63]. The update rule is as follows in Equation 4.7

$$Q'(s,o) \leftarrow (1-\alpha)Q(s,o) + \alpha[r_{tt+k} + \gamma^k \max_{o'} Q(s',o')] \tag{4.7}$$

$$R(s_{t+k}, o, s_t) = r_t^k + r_{t+1}^{k-1} + ... + r_{t+k} \tag{4.8}$$

The convergence of the state-option value function $Q \rightarrow Q*$ has been proven by Parr in [53]. An implementation of SMDP Q-learning is provided in Algorithm 4.1.

Algorithm 4.1 SMDP-Q Learning Algorithm

```
Set number of episodes, N
Parameters: P = {α, γ, ϵ, λ, pQ, ...}
function [Q*(s,o), μ*(s)] = LEARN(N, P)
    Initialize: Q,π
    while current Episode < N do
        t = 0
        Initialize agents and task positions: s_t ∈ S,
        while not at the goal: s_t ≠ s_goal do
            Select option: o_t ~ μ(o|s_t)
            k = 0, discount = 1, r_p = 0
            while option not complete (ie. not terminal) do
                Determine low level action a_t from o_t
                Step state forward: s_{t+1} = Simulator(s_t, a_t)
                Observer one-step reward: r = R(s_{t+1}, o_t, s_t)
                Update path reward: r_p ← r_p + r × discount
                Update discount: discount ← γ × discount
                k = k + 1
            end while
            function Q' = UPDATE VALUE FUNCTION(Q,r_p)
                Q'(s(t), o) ← r_p + discount × max_o[Q(s(t), o)]
                t = t + k
            end function
        end while
    end while
end function
```

The implementation of SMDP-Q learning in Algorithm 4.1 follows the familiar structure introduced in Algorithm 3.2. Four distinct phases are as follows: 1) Initialize values, 2) Select and option, 3) Carry out the option until terminal, and 4) Update the value function. This implementation presents several challenges. First, it is clear that options must be complete before learning can take place. This hurdle can be overcome by Intra-option learning, the details of which can be found [63]. Perhaps more subtle, two models are required by Algorithm 4.1. In addition to the state transition simulator ($s_{t+1} = f(s_t, a_t)$), an action-

generation model must also exist ($a_t = h(o_t)$). In many cases this model may be refereed to as a "low-level controller", and may be much easier to simulate than to create a-priori probabilistic models.

Of particular interest is variations on the SMDP-Q algorithm. Sutton, Precup, and Singh notes that an extension of eligibility trace methods to SMDPs should be straightforward, although no work in this direction exists to the knowledge of the author. Several researchers have noted functional challenges with combining reward shaping with SMDPs []. A special case time-based reward shaping strategy is proposed in [44] which can preserve policy optimality for SMDPs.

One major deficiency of standard SMDP-Q-Learning is that options must be completed before the visited states are updated. Sutton et al. addresses this problem with so called "intra-option value learning". An example in which options can be interrupted during execution is provided in [67]. Options may in fact be interrupted at every time step, in which case the resulting single-step problem behaves like a classical MDP. Indeed the interesting research areas in SMDP theory lie at the intersection of MDPs and SMDPs: construction options from actions, decomposing options into constituent parts, and interrupting options are all active areas of research in reinforcement learning.

4.7.2 GPP Problem Size and Scaling

The learned solution to a GPP will be either a state-action value function $Q(s, a)$ or a state-option value function $Q(s, o)$. In the depicted case, the four actions with result in a $Q(s, a) \in \mathbb{R}^{nS \times 4}$ versus $Q(s, o) \in \mathbb{R}^{nS \times 2}$ in the SMDP case. Iterating over and storing twice as many state-action pairs may seem insignificant for this case, but quickly scales as the number of primitive actions increases. Figure 4.4 plots the number of state-action pairs for a 2-agent variable task assignment problem. When the number of primitive actions is increased from four, as in Figure 4.19, to 10 and 100, the differences in problem size and scaling become more apparent.

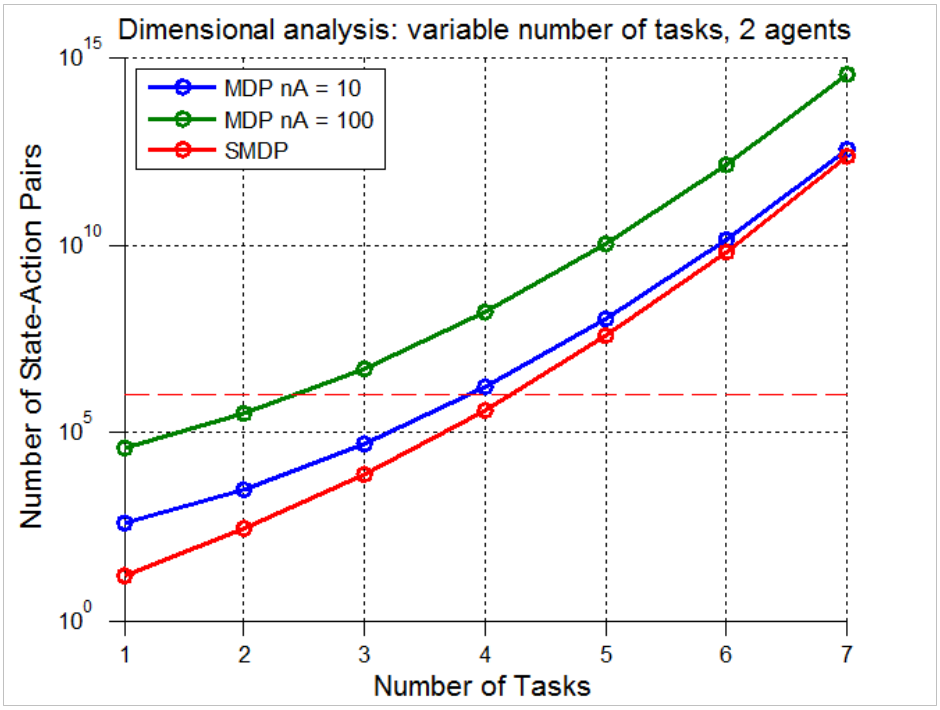


Figure 4.4: Scaling properties of a routing problem represented as an MDP v. SMDP

Represented as an MDP, the number of primitive actions is nA^2 versus $(m+1)^n$) options when represented as an SMDP.

4.8 Learned Vehicle Routing Simulation Setup

A simple vehicle routing task is set up to demonstrate the representational scheme discussed in Section 4.4, the modeling discussed in Section 4.5, and learning algorithms discussed in Section 4.7. The goal is to demonstrate several basic VRP characteristics with the learned solution.

- Multi-agent routing solution
- Task allocation, deallocation, loitering
- Temporally extended actions (options in SMDP model)

Chapter 2 introduced the Generalized Planning Problem as particular class of Assignment Problem with an emphasis on hybrid-routing and scheduling. Example 1 highlights how some routing and scheduling elements may manifest themselves in a real-world problem, with an additional emphasis on stochastic problem elements. Beyond learning solutions to just VRPs or even a more complex VRPPD, we would like to learn solutions to the GPP. Several elements that would have to appear in the solution to the GPP are as follows:

- Task sharing (dual allocation)
- Asynchronous action interruption and termination
- Action outcome uncertainty
- Stochastic action duration (SMDP model)
- Inclusion of human feedback at learn-time or run-time

Section 4.4 first introduced the cost-based GPP representational scheme. Section 4.5 cast this representation as a MDP and SMDP model, with Section 4.7 providing algorithms for learning policies on both models. The GPP representational and modeling scheme combined is depicted in Figure 4.5 for the SMDP model.

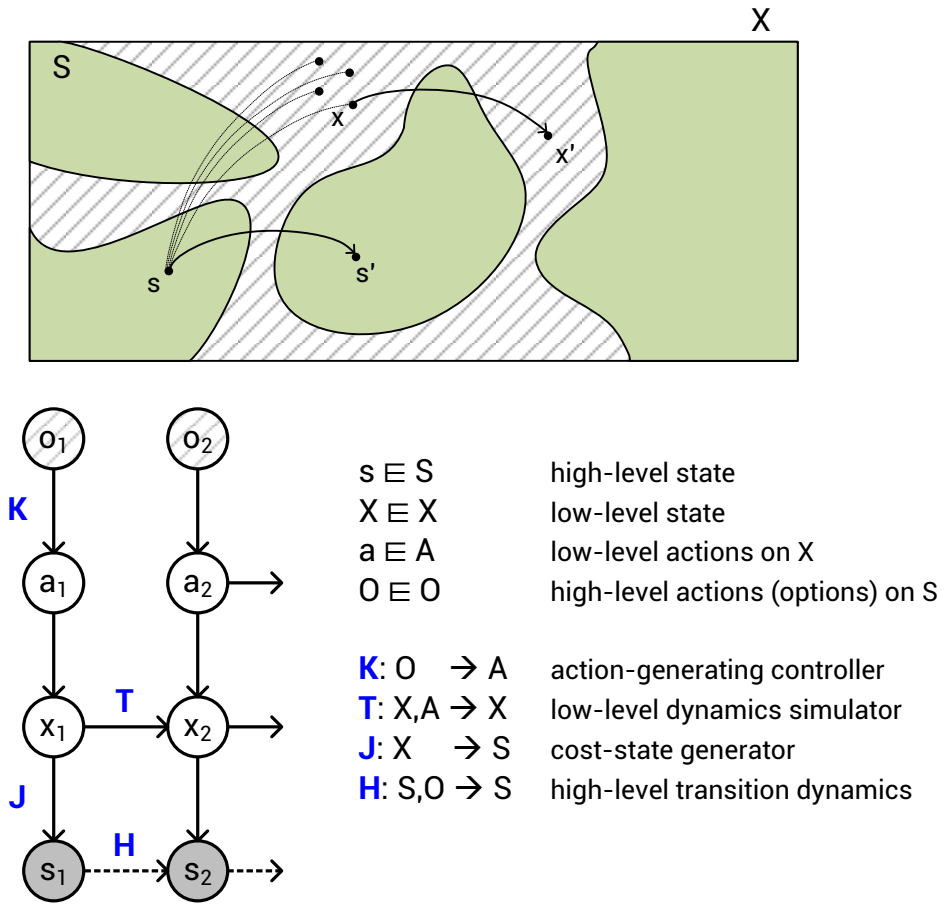


Figure 4.5: GPP representation, modeled as a SMDP

In the depiction, X is the entire state space, including all agent, task, and environmental information. Defining any GPP based on X would produce a hopelessly large state space.

As such, X is projected to a lower dimension space, S, a process that is described in detail in Section 4.4. The lower dimension abstracted space is cost-based, and many x will map to the same s by the map $\mathbf{J}$. The map $\mathbf{J}$, constructed as shown in Figure 4.1 is therefore a *surjective* and *non-injective* map from the domain X to the codomain S (every element of the codomain is maps to more than one element of the domain). It is exactly this surjectiveity that results in a massive reduction in the number of states.

Though the standard GPP formulation does not specify a learning method, modeless temporal difference methods are exclusively used here. Therefore a low-level simulator is required, which is exactly the state transition map $\mathbf{T}$. Additionally, for SMDP learning, the map, or controller, $\mathbf{K}$ is required to generate low-level actions from options. Figure 4.5 shows the high-level transition function $\mathbf{H} : S \times O \to S$, which we assume to be unknown at the time of learning. $\mathbf{H}$ is effectively simulated by translating $O \to A$ with K, applying $a \in A$ to $x \in X$ and producing the successor low-level state $x' \in X$, which is translated to the high-level state S via the surjective map $\mathbf{J}$. In effect, options O are applied directly to the state S as depicted in Figure 4.3.

The Matlab-based test environment shown in Figure 4.6 is used for all subsequent simulations. Agents are modeled as velocity-constrained point-masses. A more expressive vehicle model is possible, although the benefits of including one at this level of abstraction are minimal. Tasks are modeled similar to vehicles and are stationary for subsequent simulations. The environment is also static, although a wind model or other environment dynamics would be easy to implement.

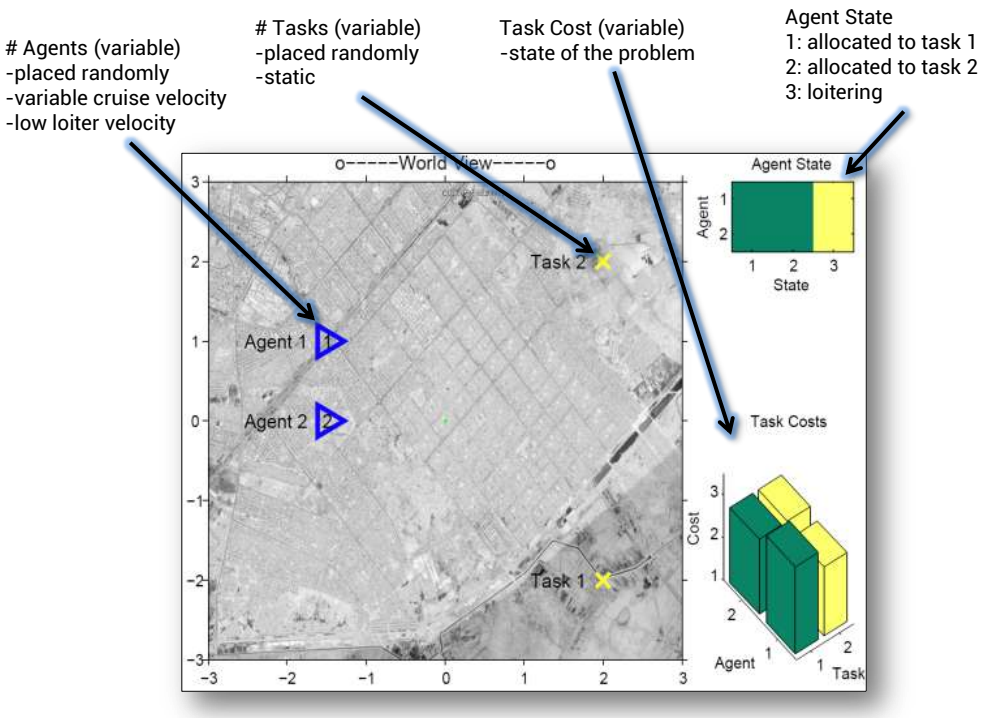


Figure 4.6: Learned VRP simulator overview

The VRP simulator used here has the ability to independently vary the all problem attributes, such as the number and type of agents and tasks, the environment, agent/task dynamics, learning algorithm, and all learning parameters

At the outset of each simulated episode the agents and tasks are initialized to a random and uniformly distributed position over the physical space. Learning is carried out as follows for any learning algorithm (standard value iteration, Q(λ), SARSA(λ) on-policy temporal difference learning algorithm (SARSA(λ)), etc.):

Algorithm 4.2 provides a generic overview of how a TD learning algorithm would solve for optimal GPP polocies. It could easily be modified to work for value iteration, for example, by sampling from a transition model instead of forward simulation.

Algorithm 4.2 GPP Learning Algorithm

```
Set number of episodes, N
Parameters: P = {α, γ, ϵ, λ, pQ, ...}
function [Q*, π*] = LEARN(N, P)
    Initialize: Q,π
    while Current Episode < N do
        t = 0
        Initialize agents and task positions: s_t ∈ S,
        while Not at the goal: s_t ≠ s_goal do
            function a_t = SELECT ACTION(s_t)
                Via ϵ-greedy or other method
                a_t ~ π_b(a_t|s_t)
            end function
            function s_{t+1} = TAKE ACTION(s_t, a_t)
                Step state forward: s_{t+1} = Simulator(s_t, a_t)
                Simulator includes agent motion or environment stochasticity
            end function
            function Q_{t+1} = UPDATE VALUE FUNCTION(Q_t)
                Reward: r = R(s_{t+1}, a_t, s_t)
                Update rule: Q_{t+1} = Update_Rule(Q_t, r)
            end function
            t = t + 1
        end while
    end while
end function
```

4.8.1 Simulator Implementation

As discussed in Section 4.4, the proposed GPP state is essentially the cost matrix of a VRP, where edges generalized costs rather than (just) distances. The simulator therefore must perform the critical action of distilling the abstracted cost state S from the full state X. This translation is largely an implementation challenge and must be performed efficiently, or stored as a lookup table to be accessed at run-time.

A full list of parameters available to the simulator at learn-time and run-time ("Testing") is included in Table 4.4. Ranges are provided for variables where applicable. Several variables are limited by computation and storage constraints and therefore have no listed maximum.

Specific parameter values are provided in subsequent example problems. The simulator is written in Matlab as a set of abstract classes, also listed in Table 4.4. Base classes are provided for agents (**agn**), tasks (**tsk**), and the environment (**env**). A super-learner class takes these elements as arguments. A super-problem class takes the leaner as an argument.

Parameter	Range	Description
Environmental		
m	$[1, M]$	number of tasks
n	$[1, N]$	number of agents
c	$[1, C]$	cost state grid resolution
w	$[1, W]$	$w \times w$ world size
v	$(0, V]$	agent velocity
v_L	$(0, V_L]$	loiter velocity
v_T	$(0, V_T]$	task velocity
p_{fail}	$[0, 1]$	action failure probability
s_0	$[-w, w]$	agent/task initial locations
env$(params)$	$\{static, windy\}$	environment dynamics model
agent$(params)$	$\{line, grid, uav, taxi\}$	agent dynamics model
task$(params)$	$\{static, wandering, \}$	task dynamics model
Learning		
e	$[0, E]$	number learning episodes
t_{max}	$[0, T_{max}]$	learning episode timeout
γ	$(0, 1)$	discount factor
α	$(0, 1]$	constant learning rate
α_d	$(0, 1]$	learning decay rate
ϵ	$[0, 1]$	exploration probability
ϵ_d	$[0, 1]$	exploration decay rate
λ	$[0, 1]$	eligibility trace parameter
pQ	$\{0, 1\}$	priority queue flag (off, on)
pQ_{min}	$[0, 1]$	priority queue threshold
learner$(params)$	$\{Q(\lambda), SARSA(\lambda), SMDPQ\}$	learner object
problem$(params)$	$\{gridMDP, vrpMDP\}$ $\{vrpSMDP, taxiMDP\}$	problem object
Testing		
h	$[0, H]$	number test episodes
t_{max}	$[0, T_{max}]$	test episode timeout
test$(params)$	**problem**$(params)$	test object

Table 4.4: Simulator parameters and associated values for a learned VRP.

Lower-case letters indicate specific values for a particular problem; upper-case letters indicate maximum values, which depend on the computation and storage capabilities available.

In this way, many problems can be created by interchanging agents, tasks, environment, and learner.

The test class (**tst**) is an instantiation of a problem class. It can take a different agent and task model, which is useful for showing policy generality. For example, a policy can be learned for agents with one cruise velocity and tested for agents with a different cruise velocity. Clearly, changing the underlying simulator will have effects on the policy performance, as the policy is only valid for the dynamics on which it is trained. However, in practice, small perturbations to agent and task dynamics do not affect the outcome.

All subsequent simulations have been carried out on an IBM ThinkPad T61p Windows 7 machine with Core 2 Duo Intel processor clocked at $2.4GHz$ and with $2GB$ of RAM.

4.8.2 Policy Properties

The learned policy, as a direct result of decoupling the problem state from a strictly physical space, is general to any agent-task spatial configuration; the policy need not be relearned when changing agent or task location. Additionally, the policy has been shown to generalize well to small changes in agent dynamics such as changing the agent cruise velocity.

4.9 Learned Vehicle Routing Simulation Results

Several simulations have been carried out in similar fashion to those presented in Chapter 3 for the 10x10 maze world in Example 2. The learned VRP-like scenario in Example 6 presents several challenges different than those in the maze world.

- The VRP state space is vast. The relatively simple 2-agent 3-task VRP has on the order of 10^4 states, whereas the 10x10 maze in Example 2 has 10^2 states.
- The VRP goal is often far from the starting location. As a result, it may take many steps before receiving a reward.
- The optimal VRP solution oftentimes involves motion away from the goal (ie. non-greedy actions) before the goal can be achieved.
- Despite the great number of states, many scenarios states are similar in that, despite different spatial arrangement of agents and tasks, the policy is the same.

Example 6 (Learned Routing)**.**

In this vehicle routing scenario, two agents must successfully navigate through three waypoints. The overall mission is said to be complete when all task have been visited at least once. The two agents are homogeneous and have a constant velocity. The three tasks (waypoints) are static. The problem is modeled as an MDP with the following parameters.

Parameter	**Value**	**Description**
m	3	number of tasks
n	2	number of agents
w	$6\,km$	$w \times w$ world size
v	$30\,m/s$	agent velocity
$s \in S$	$\mathbf{C_{ij}}$	state set
s_{goal}	$\mathbf{C_{ij}} = \mathbf{0}\ \forall i, j$	goal state
$a \in A$	$[A_1T_1, A_1T_2, ..., A_2T_1, ..., A_nT_m]$	action set
$R(s', a, s)$	*see below*	reward function
λ	$\{.1, .3, .5, .7, .9\}$	eligibility trace parameter
ϵ	$\{.3, .5, .7, .9\}$	exploration probability
e	5000	number learning episodes
h	100	number test episodes

The reward function is comprised of several components and is structured as follows:

$$R_{total} = R_{goal} + R_{nloiter} + R_{ndualalloc}$$

$$R_{goal} \triangleq R(s_{goal}, a, s) = 1$$

$$R_{nloiter} \triangleq R(s', a_u, s) = -1 \qquad \text{where } a_u = \text{any agent unallocated}$$

$$R_{ndualalloc} \triangleq R(s', a_d, s) = -1 \qquad \text{where } a_d = \text{any dual allocation}$$

Several additional constraints and assumptions are as follows:

C1. Tasks are static (do not move) and all have the same demand.

C2. A task is considered to be done when any agent, whether allocated to it or not, enters a small radius about the task location.

C3. Once a task has been done by any agent, it is considered done for all agents. This is equivalent to the standard VRP "visit once" assumption.

Chapter 3 introduced several learning techniques of particular consequence for solving a learned vehicle routing problem as in Example 6. Eligibility traces (subsection 3.5.2) blend Monte-Carlo full backups with Dynamic Programming backups. The TD(λ) algorithm was shown to be useful in cases where the goal state and reward is far from the start state. The exploration vs. exploitation rate can also be tuned (subsection 3.5.3). In applications with vast state spaces, the sequence of states updated can be prioritized with prioritized sweeping methods(subsection 3.5.4).

The VRP has several defining features that motivate the use of these learning techniques:

- The goal can be far-removed from the start, resulting in long episodes, particularly at the early stages of learning where possible state-action trajectories are vast.
- Many state-action trajectories that lead towards the goal (all tasks done).
- The VRP state space is vast, scaling exponentially with the number of agents, tasks, and the level of discretization of the cost variable. Additionally, many of the states are unlikely/impossible to be visited. Both qualities indicate that some state visitation scheme is necessary.

The solution to Example 6 is presented in the following results. Eligibility trace, exploration v. exploitation methods, and prioritized sweeping methods are applied to jointly study their effects on the solution, and conversely, what the effecacy or lack thereof of the applied method indicates about the underlying problem. All solutions are compared against the near-optimal greedy solution to the 2-agent, 3-task learned VRP. Performance metrics are similar to those used to study Example 2 in Chapter 3.

Utility As defined in Equation 3.18 Utility is a learning performance metric that indicates the speed of a learning algorithm: the acquired value versus learning episode number. The learning curve, also a learning performance metric, shows episode duration as a function of learning episode number, and thus the quality of the policy as a function of learning iteration. As a metric of final policy quality, the policy is simulated over a fixed number of iterations. The differences between policies learned under varying parameters can thus be studied.

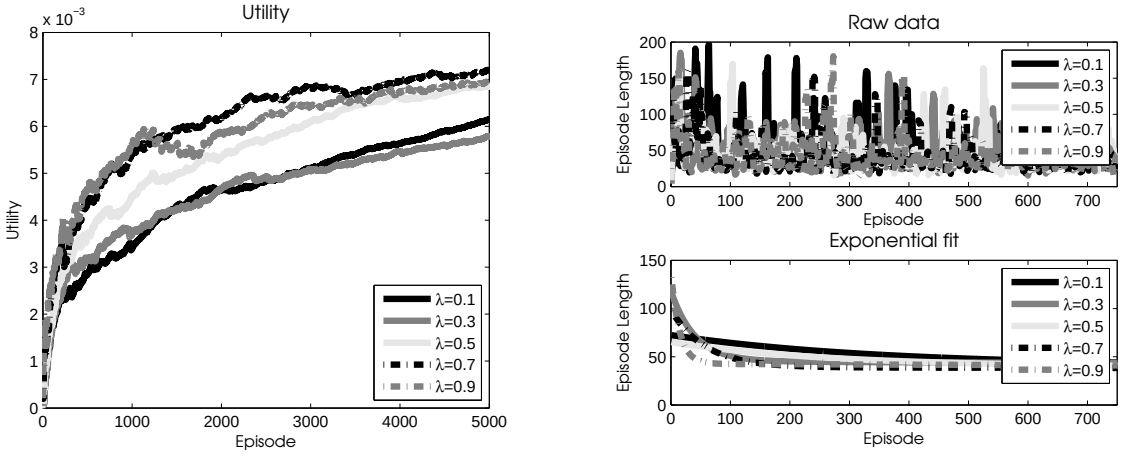
Learning Curve The learning curve shows the trend of episode length versus learning iteration number. In the case of Example 6, "length" is normalized by the step size (*steps/distance per step*) so that simulations with different vehicle velocity can be compared. The horizontal axis is expressed in terms of learning episode number rather than iteration to normalize the data to a specific number of episodes, as is commonly done for episodic learning problems.

Policy Performance Once a policy has been learned, it is simulated over a number of either randomly seeded or predetermined test episodes. In the case of a 10x10 maze world such as Example 2, the number of states is small (100, less the number of "wall

states"), and policy testing takes place over all these possible states. In the case of a VRP as in Example 6, the state space is vast, so a representative subset of states is used to test policies. A timeout is set on each episode, and episodes that timeout are thrown out of the test data. Timeouts occur when imperfections in the learned policies lead to livelock and deadlock situations which would continue indefinitely.

4.9.1 Learned Vehicle Routing with Eligibility Traces

Tests were performed on the two-agent three-task VRP in Example 6 to determine the efficacy of eligibility trace-based learning. The parameter $\lambda = [0, 1]$ was varied over several tests comprised of 5000 episodes. The results indicate a significant positive effect on both the learning performance (Figure 4.7) and end policy quality (Figure 4.8).



(a) Utility

(b) Learning Curve

Figure 4.7: VRP learning performance results for varying λ.

The utility analysis in Figure 4.7a indicates that $\lambda = 0.7$ test gained more value than the other cases, which would be indicative of end policy quality and learning rate. The learning curve in Figure 4.7b also indicates high-lambda learning performs the best, with $\lambda = 0.9$ performing slightly better than $\lambda = 0.7$. It is important to note that the bottom learning curves are based on a fit for clarity, and may not be able to discriminate between two neighboring tests, such as $\lambda = 0.7$ versus $\lambda = 0.9$ due to noise.

The overall trend in both utility and learning curve data both indicate that the policy improves the most over successive learning iterations as λ increases.

To test the resulting policies, 100 randomly-seeded trials were simulated for each value of λ (with the same random seeds used across all tests).

Figure 4.8 shows both a histogram of episode lengths and a plot of episode length as a function of λ. The test results are compared against a hand coded greedy policy, which is

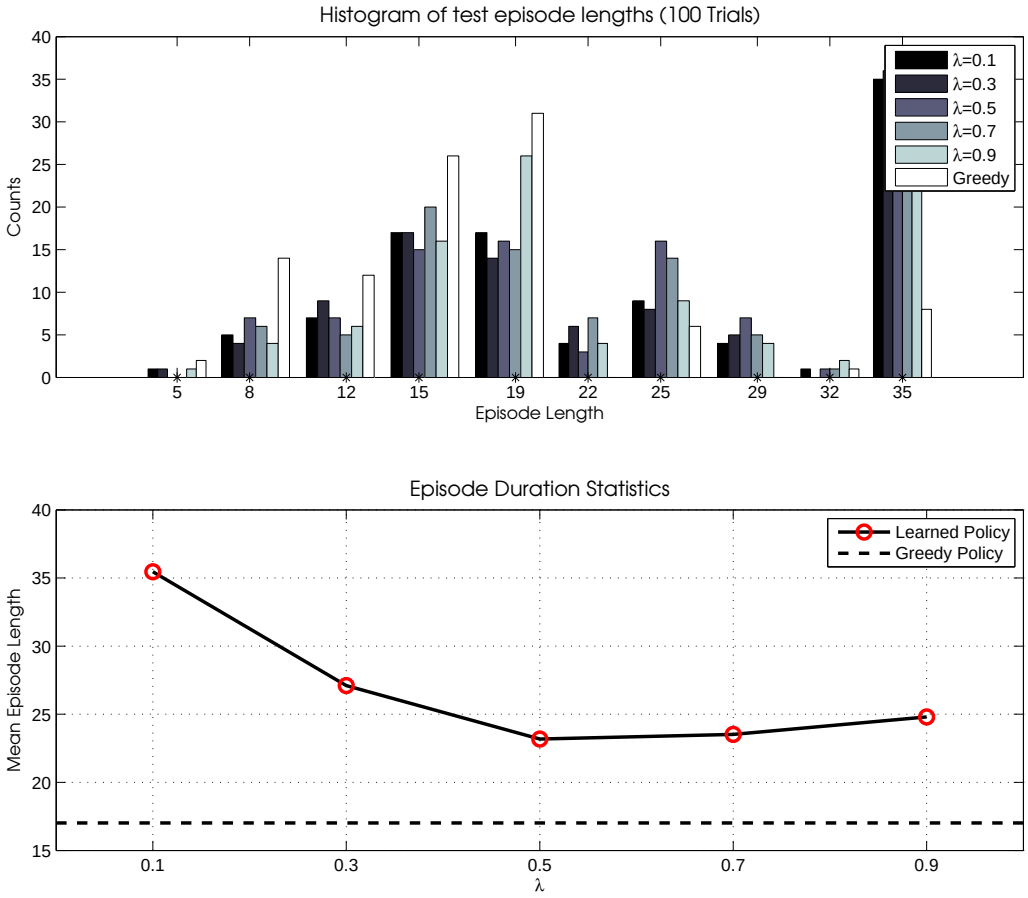


Figure 4.8: Policy test varying eligibility trace weighting λ for a 2-agent 3-task VRP. The exploration parameter ϵ is set at 0.5.

close to the optimal policy for the two-agent three-task VRP studied here. The last bin in the histogram contains all the episodes that took 35 episodes or greater and can be thought of as a grouping of all the worst case performances. These poorly performing trials can potentially be removed or mitigated through longer learning, as it is clear from the utility plot that the optimal value function and policy has not yet been reached after 5000 learning episodes. Predictably, the number of long episodes is small for the greedy policy, as it not susceptible to the policy imperfections that plague the learned solutions.

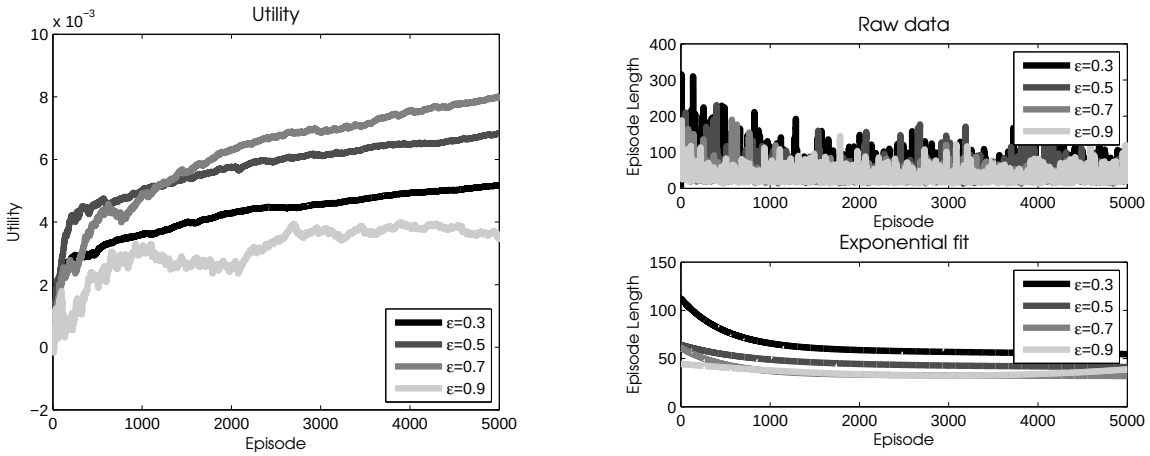
The results in Figure 4.8 indicate that the learned solutions for $\lambda > 0.5$ perform comparably better than small-lookahead $\lambda < 0.5$ policies. This result will likely be even more true

as the number of tasks relative to agents increases, and episodes thus become longer, further separating the start state from the goal state. The greedy policy performs best in all cases, likely because it has no imperfections, as can be seen in the histogram.

4.9.2 Learned Vehicle Routing with Varied Learning Exploration

As was introduced in subsection 3.5.3, $\epsilon - greedy$ off-policy learning can drastically improve learning performance. Early in learning, when little is known about the target value function and policy, exploratory actions are favorable in that they further explore the state space. Later in learning however, after a rough value function has been learned, more exploitative actions that are greedy with respect to the current value function V^{π} are favored.

The parameter ϵ in the following experiments is interpreted as the the probability with which exploitative/greedy actions are taken versus exploratory actions. In the case of $\epsilon = 1$ actions are always chosen according to Equation 3.13. In the case of $\epsilon = 0$, the behavior policy is always random.



(a) Utility

(b) Learning Curve

Figure 4.9: VRP learning performance results for varying ϵ.

Learning performance for varying ϵ is analyzed in Figure 4.9. The trend in the utility data shows that *increasing* ϵ (ie. less random actions) corresponds to better learning performance. However, between $\epsilon = .7$ and $\epsilon = .9$ utility destabilizes and drops off. In all cases, as learning time approaches infinity, the same utility should be reached, since varying the probability of taking exploratory actions does not change the end optimal policy. However, clearly certain high values of ϵ can destabilize the learning procedure. It may be the case that other other parameters, such as learning rate (α) and discount factor (γ) could stabilize the $\epsilon = 0.9$ case.

The learning curve in Figure 4.9b indicates an intuitive trend: that as more random actions are taken, the episode length increases. While utility is a reflection of the target

policy, the learning curve is more indicative of the behavior policy, which is chosen to be some amount of random so as to increase state coverage. So while the "best" learning curve corresponds to the $\epsilon = 0.9$ case, that case may not necessarily correspond to the best target policy.

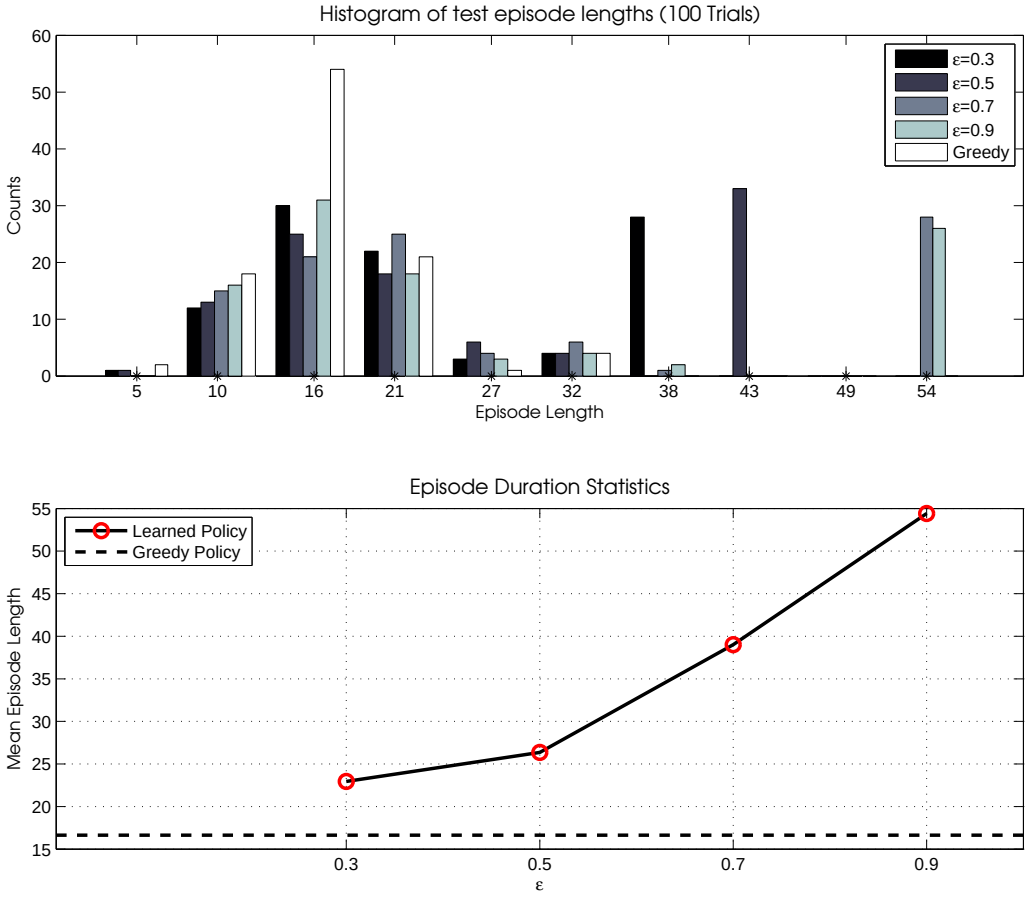


Figure 4.10: Policy test varying eligibility trace weighting ϵ for a 2-agent 3-task VRP. The eligibility trace parameter λ was set at 0.8

Testing the learned policy over successive random trials reveals that small ϵ, more exploratory behavior policies produce the best target policies. This result makes sense particularly in the context of a VRP, where the branching factor of the associated decision tree is high. In complex settings, certain regions of the state space may remain unexplored without introducing random actions. Random action also aid in learning optimal policies that

must be non-greedy. In the maze world problem, for example, certain optimal trajectories require initially moving away from the goal so as to reach it in the long run. VRPs also have this non-greedy solution characteristic: the optimal solution in many cases requires initially not moving towards the closest task, because the greedy action will ultimately lead to sub-optimality.

A common variation on exploratory behavior policies is variable exploration. At early learning iterations, a more exploratory behavior policy ensures the state space is sufficiently surveyed; as learning progresses, the behavior policy should turn increasing "exploitative" ie. greedy with respect to the current value function. The following results show learning and policy performance for linearly increasing ϵ as learning progresses. The behavior policy is governed by Equation 4.9 where e is the current episode number and E is the total number of learning episodes.

$$\epsilon(e) = (\epsilon_{max} - \epsilon_{min}) \times \frac{e}{E} \tag{4.9}$$

For $\epsilon_{max} = \epsilon_{min}$, ϵ remains a constant parameter throughout learning, as in the previous results in Figure 4.10.

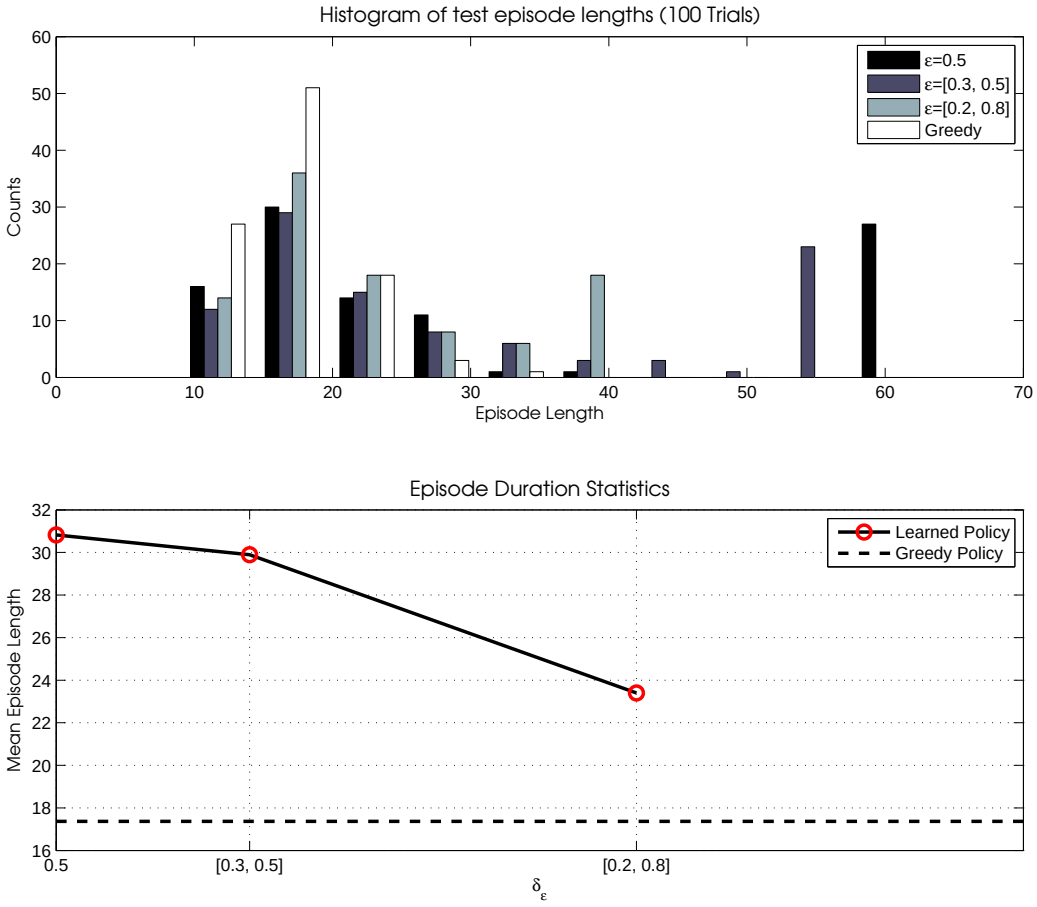


Figure 4.11: Policy test for linearly varying exploration probability

The test consists of three sets of $[\epsilon_{min}, \epsilon_{max}]$: $[0.5, 0.5]$, $[0.3, 0.5]$, $[0.2, 0.8]$

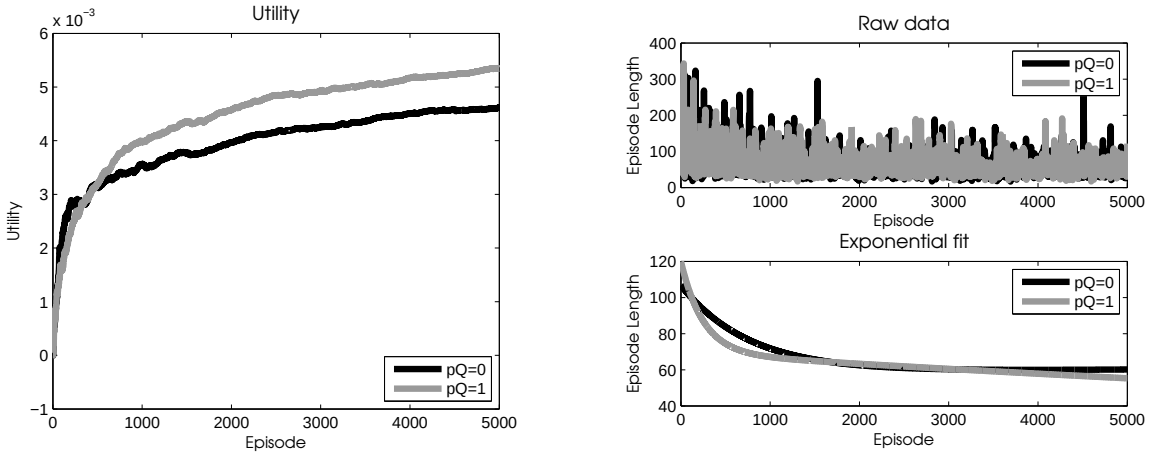
Figure Figure 4.11 shows that policy performance can be greatly improved using a variable-ϵ behavior policy.

4.9.3 Learned Vehicle Routing with Prioritized Sweeping

Prioritized sweeping was introduced in subsection 3.5.4 as an efficient way for exploring a state space. Results on the 100-state maze world (Figure 3.10) indicate that even for relatively small problems, learning performance can be improved through intelligent choice of initial state. The two-agent three-task VRP has $15,625$ states–still a small learning problem by modern standards (computer Go in [61] has $10^{1}70$ states! Function approximation with

10^6 features is used to solve it.)

The prioritized sweeping method described in Algorithm 3.3 is applied to the VRP of Example 6. The learning curve in Figure 4.12b shows that using the priority queue to explore the state space ($pQ = 1$) results in faster learning. Value function utility in Figure 4.12a also seems to improve with the use of a priority queue, and appears to even still be steadily gaining value at the end of 5000 learning episodes. In all cases, other parameter values (λ,ϵ) are held constant.



(a) Utility

(b) Learning Curve

Figure 4.12: VRP learning performance results with prioritized sweeping.

Previously identified parameters $\lambda = 0.8$ and $\epsilon = 0.3$ were used in these learning trials.

The resulting target policy is tested in the same way as previous experiments–with 100 randomly seeded test trials. Figure 4.13 shows that the use of the priority queue drastically reduces the average episode length, from 26 to 20. Referring back to previous policy test results for varying λ (Figure 4.8) and ϵ (Figure 4.10), it is clear that appropriate parameter selection combined with prioritized sweeping produces the best policy.

Figure 4.13 provides further insight into how the use of a priority queue may specifically benefit the learned-VRP. The learned policies in fact outperform the greedy policy in short and medium-length episodes. Problems arise, as in the other experiments, with the longer episodes, indicating policy imperfections for some particular cases. The histogram further discriminates between the two learned-policy cases ($pQ = 0$ and $pQ = 1$). The case in which the priority queue is *not* employed ($pQ = 0$), while comparable for short and medium cases, has poorer "worst case" performance. The histogram show that approximately one third of the non-priority queue episodes took 37 or more steps to complete, while one third of the episodes trained with the priority queue took 32 steps to complete.

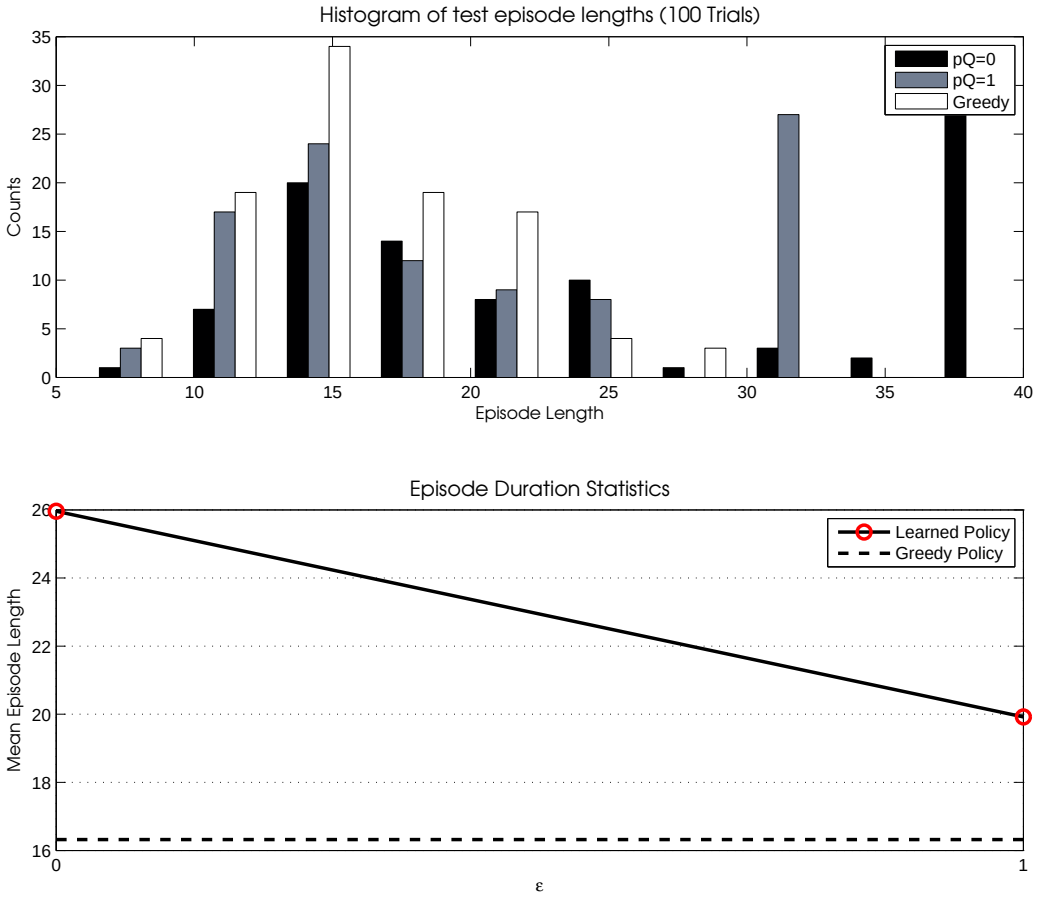


Figure 4.13: Policy test with prioritized sweeping for a 2-agent 3-task VRP.

4.9.4 Reward Shaping

Reward shaping is a powerful technique for adding additional guidance to a learning problem beyond the primary reward function, $R(s', a, s)$. In a standard learning problem, no specific requirements are placed on the reward function, and choosing "correct" reward functions, in the sense that they produce good policies with a minimal learning iterations, is largely a matter of design. In practice, reward functions are often tuned through a sometime laborious trial-and-error process until a desired outcome is achieved. The question then arises, having determined an appropriate reward function, how much can a problem designer change the function and still arrive at the same policy?

Ng, Harada, and Russell provide an answer in a technique known as reward shaping [50].

The theorem of reward shaping essentially says that $R(s', a, s)$ can be augmented, or shaped, with an additional additive function, and the overall problem under the new reward will be optimal, so long as the shaping function has certain properties. Theorem 1 provides further detail:

Theorem 1 (Policy invariance under reward shaping)**.** *Let $M =< S, A, P, R >$ be an MDP according to Definition 7 with optimal value function V^* and policy π^*. Consider the augmented reward function:*

$$R'(s', a, s) = R(s', a, s) + F(s', a, s) \tag{4.10}$$

$$F(s', a, s) \triangleq \gamma\Phi(s') - \Phi(s) \tag{4.11}$$

Let $M' =< S, A, P, R' >$ be a new MDP based on R' rather than R. If there exists a potential-based shaping function $F : S \times A \times S \to \mathbb{R}$ according to Equation 4.11, then every optimal policy in M' will also be optimal in M.

A proof of Theorem 1 is provided in [50]. The proof establishes that the existence of the potential-based shaping function F is a necessary and sufficient condition for policy invariance under reward shaping.Reward shaping theory provides no insight on constructing admissible potential functions Φ, but does provide a method of checking them with Equation 4.11.

4.9.4.1 Reward Shaping and Semi-Markov Decision Processes

Few applications of reward shaping applied to SMDP can be found in the literature. Ng, Harada, and Russell propose a modified shaping function, given in Equation 4.12, where τ is the option execution time and β is the SMDP discount rate.

$$F(s', a, s, \tau) \triangleq e^{-\beta\tau}\Phi(s') - \Phi(s) \tag{4.12}$$

The proposed extension of reward shaping presents several practical challenges to modeless SMDP learning used here. First, option execution is carried out by a simulator and therefore execution time is unknown until the option is completed. $F(s', a, s, \tau)$ must therefore be calculated online. Additionally, although unimplemented here, SMDP learning with option interruption combined with the proposed shaping function may present problems since the execution time τ can change at any point due to interruption.

The authors of [44] propose an alternate to the time-based shaping function in Equation 4.14 in the context of a multi-agent strategy game. The additional negative reward, $\tau(o)$, is the number of steps to episode completion (they use $\tau(a)$, but o is used here for consistency to denote semi-Markov options rather than Markov actions). By removing the state from the potential function and under certain conditions, they guarantee consistency under SMDP-Q learning. In general, applying reward shaping techniques to temporally-abstract Markov models such as those described in subsection 4.6.7 is an open area of research.

$$F(s', o, s) = \tau(o) \tag{4.13}$$

$$\tag{4.14}$$

4.9.4.2 Learned Vehicle Routing Under Reward Shaping

The concept of rewarding subtask accomplishment has been proposed in [50]. In the given example, similar to the grid maze world in Example 2, an agent must navigate through several waypoints before reaching the ultimate goal waypoint. The reward function, R gives -1 everywhere except for the goal. The shaping function F is then introduced, which gives a reward proportional to the Manhattan distance to the subtasks.

A similar approach is proposed for the VRP in Example 6. The proposal shaping function F in Equation 4.16, rewards accomplishing individual tasks, while the original reward R ($+1$ for accomplishing *all* of the tasks remains unchanged. Since the goal of a routing problem is to determine the order in which agents should visit tasks, the shaping function rewards individual task accomplishment equally, rather than specifying an a-priori ordering as in [50].

$$F(s_d, a, s_{d-1}) = 0.5 \tag{4.15}$$

$$R' = R + F \tag{4.16}$$

States $s_d \in S$ are those in which d tasks have been completed and $m - d$ remain to be done. States $s_{d-1} \in S$ are states in which $d-1$ tasks have been done. The shaping function therefore rewards intermediate task completion equally: specifically in the three-task setting, it rewards the transition between zero tasks done and one task done, and one task to two tasks, and so on. It should be noted that a full *successor state-action-state* valued reward function is required to capture subtask completion; a *state-action* valued reward function is not expressive enough to reward the event of a task transitioning from undone to done.

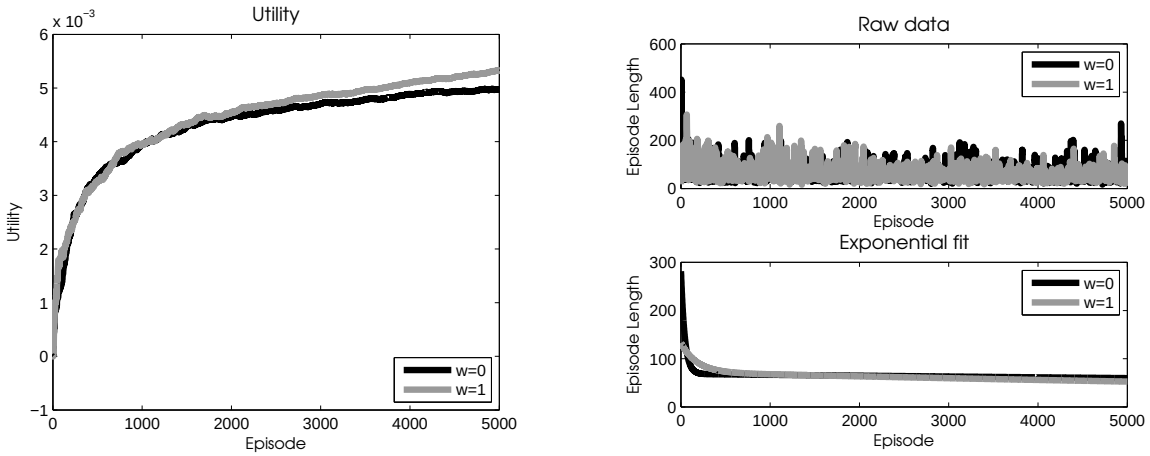
Learning performance is compared for the unshaped and shaped cases below in Figure 4.14a and Figure 4.14b. Previously identified parameter are used along with the priority queue for increased learning speed. Learning performance under reward shaping seems to be comparable to the unshaped case.

The policy test results in Figure 4.15 indicate that the inclusion of the shaping function F produces worse results relative to the unshaped case. The lack of improvement in the learned policy under reward shaping may be explained by one of two possibilities:

1. The shaping function F is not potential based and therefore there is no guarantee that policies optimal in the original mdp M will also be valid in M'.

2. The shaping is valid in most cases, but also a poor choice for solving a VRP

It seems likely that, while rewarding proximity to tasks in the single agent case in [50] is a valid choice, the choice is invalid for a multi-agent VRP. Indeed the reason why VRPs are a challenge to solve is that the best short term path (the greedy solution) is not the optimal



(a) Utility

(b) Learning Curve

Figure 4.14: VRP learning performance under reward shaping.

Previously identified parameters $\lambda = 0.8$, $\epsilon = 0.3$ and $pQ = 1$ were used in these learning trials.

strategy in all cases. By this logic, the proposed shaping function in Equation 4.16, despite being valid in some scenarios, does not satisfy the condition for a valid shaping function.

Reward shaping theory provides a necessary and sufficient conditions for policy optimality under an augmented reward function. A particular shaping function being inadmissible, however, does not necessarily mean it won't produce good results. The success of heuristic methods in TSP and VRP indicate that other inadmissible shaping functions may exist that work well in practice.

4.9.5 Learned Vehicle Routing: Seeding

Thus far, varying λ, ϵ, priority queues and reward shaping techniques have steadily improved the solution of the learned vehicle routing problem. As has been demonstrated by the varied exploration-exploitation tests, some degree of randomness in the behavior policy is absolutely essential for convergence in the off-policy temporal difference learning setting. Despite improvements gained from tuning various parameters, the learned policy has, in all cases, consistently underperformed relative to the greedy policy. For the small two-agent three-task VRP considered, the greedy policy should be close to optimal in all but a few cases. The cause of the performance gap as seen in the policy performance histograms, can largely be attributed to a small number of fringe cases that tend to live-lock or deadlock and eventually timeout. For this reason, before increasing learning time so as to refine those cases that are already performing comparably to the greedy policy, it is important to root out the worst-case test scenarios.

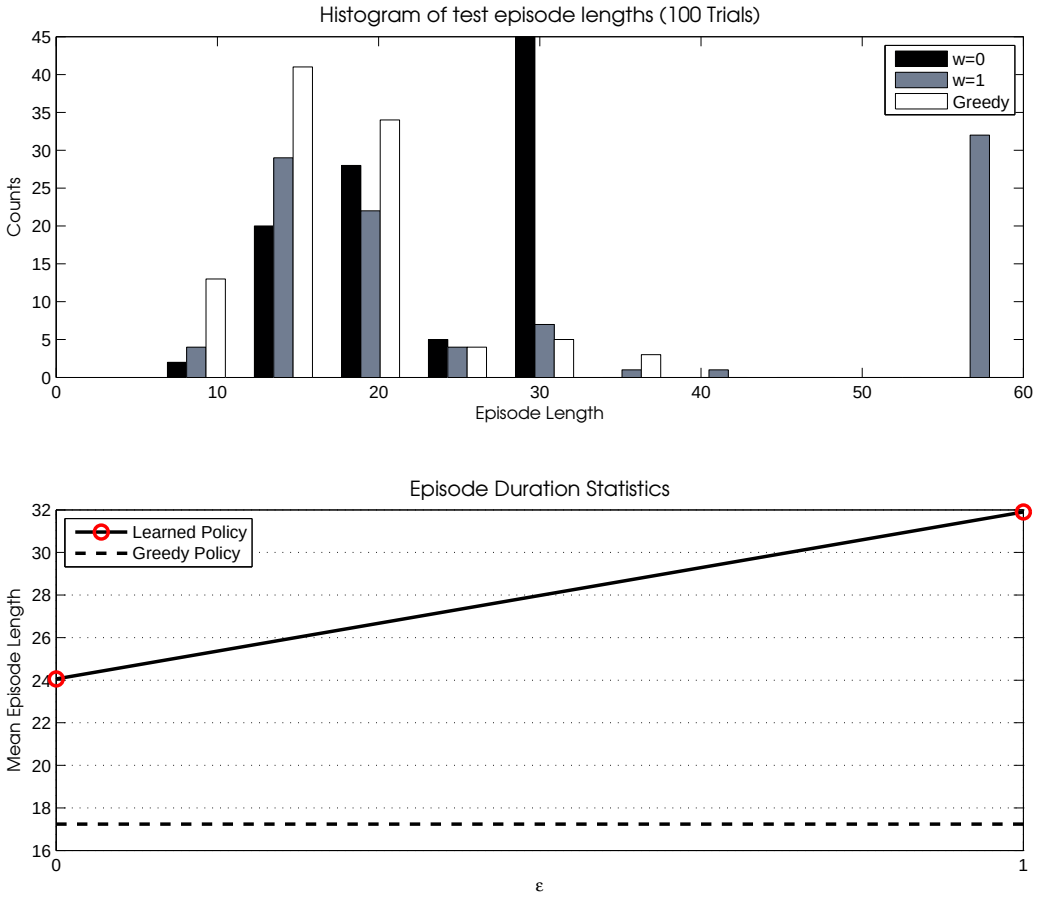


Figure 4.15: Policy test with reward shaping for a 2-agent 3-task VRP.

The only requirement placed on the behavior policy is that it be "sufficiently exploratory." Therefore more complex policies beyond ϵ-greedy may be considered. Surely the best policy to choose to guide learning would be the optimal policy, but having the optimal policy would invalidate the purpose of learning in the first place. The next closest thing would be the greedy VRP policy. This policy is easy to compute and small to store ($O(nS)$ integers). In addition to the greedy policy, denoted by π_g, some exploration and some exploration is still required to achieve the true optimal VRP solution. Following on the previously introduced variable-ϵ behavior policy, a parameterized 3-part hybrid behavior policy, π_3, is proposed.

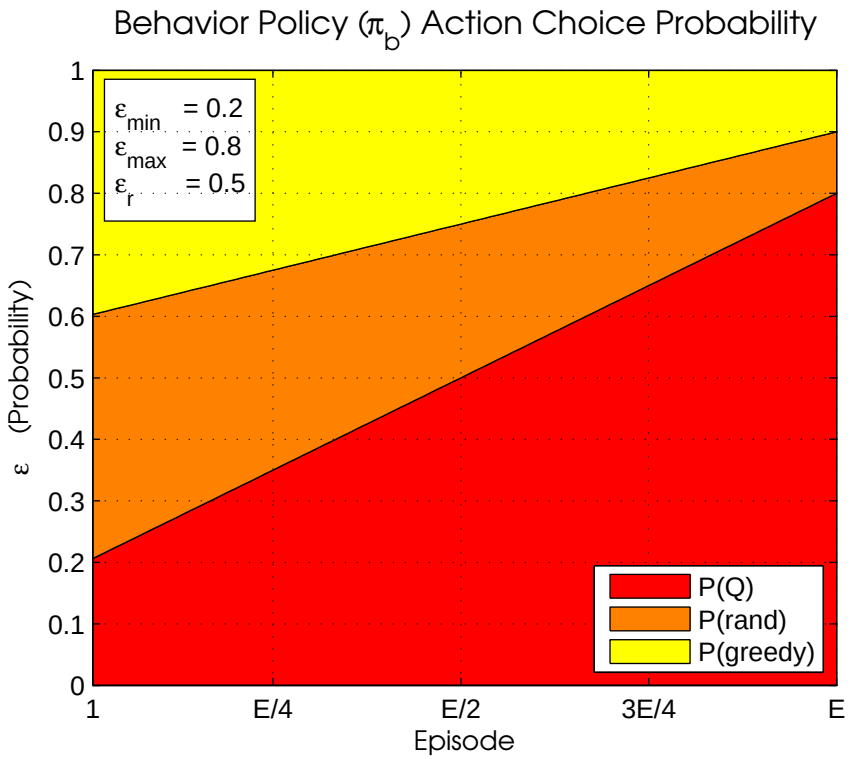


Figure 4.16: A π_3 policy, blending, exploration, exploitation, and greedy VRP behavior policies.

In this example, $[\epsilon_{min}, \epsilon_{max}] = [0.2, 0.8]$ and $\eta = .5$. P(Q) refers to the probability that an exploitative action is taken, P(rand) is the probability a random action is taken, and P(G) is the probability a greedy action is taken

$$\mathbf{P}(Q) = (\epsilon_{max} - \epsilon_{min}) \times \frac{e}{E} \tag{4.17}$$

$$\mathbf{P}(R) = (1 - P(Q)) \times (1 - \eta) \tag{4.18}$$

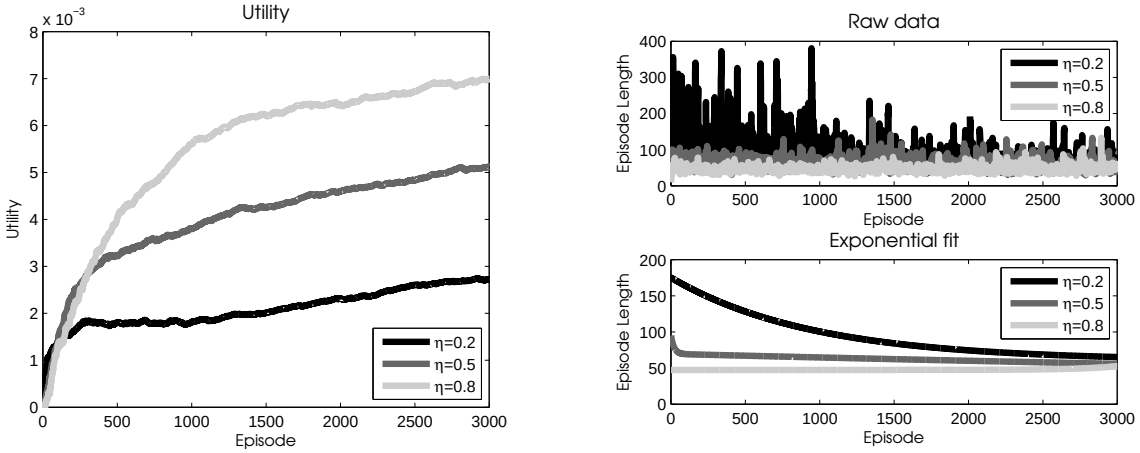
$$\mathbf{P}(G) = (1 - P(Q)) \times \eta \tag{4.19}$$

Equation 4.19 describes the probability of taking each class of action in the behavior policy. The linear policy shifting scheme is parameterized by ϵ_{min}, ϵ_{max},, and η. Exploitative action are chosen with increasing probability over the learning horizon (E), from ϵ_{min} to ϵ_{max},. Exploratory behavior is therefore chosen with probability $1 - \epsilon$. In this case, the greedy policy is used with probability η and a random action is chosen with probability $1 - \eta$.

A behavior policy is required to be "sufficiently exploratory" so as to guarantee convergence of $Q_b^\pi \rightarrow Q^*$ [66]. As time (episodes) approaches infinity, the policy depicted in

Figure 4.16 is sufficiently exploratory. The area under each segment, which can be thought of as cumulative probability, approaches infinity as time approaches infinity.

Learning results show vast improvement using the hybrid π_3 behavior policy. The utility in Figure 4.17a indicates that increasing η increases the utility of a given policy, indicating that following the greedy policy more often than the random policy reaches the goal more often. A small amount of randomness (decreasing over the learning horizon) is still required so as to avoid falling in local minima. Additionally, following the greedy policy in all cases could result in learning the greedy solution rather than the optimal solution, which are similar in cases where the number of agents relative to the number of tasks is small.



(a) Utility

(b) Learning Curve

Figure 4.17: VRP learning performance under greedy policy seeding.

Previously identified parameters $\lambda = 0.8$ and $pQ = 1$ were used in these learning trials.

The learning curve in Figure 4.17b also indicates that increasing η results in faster learning, since the greedy solution is close to the optimal solution. In fact, it would appear that later in learning, when the behavior policy becomes increasingly exploitative, selecting actions based on the learned value function rather than random or greedy options, the variance of the learning curve increases. A possible explanation is that the value function has not yet learned optimal behavior, which could be remedied by either increasing learning time or decreasing $epsilon_{max}$, the upper bound on the probability with which actions are drawn from the value function.

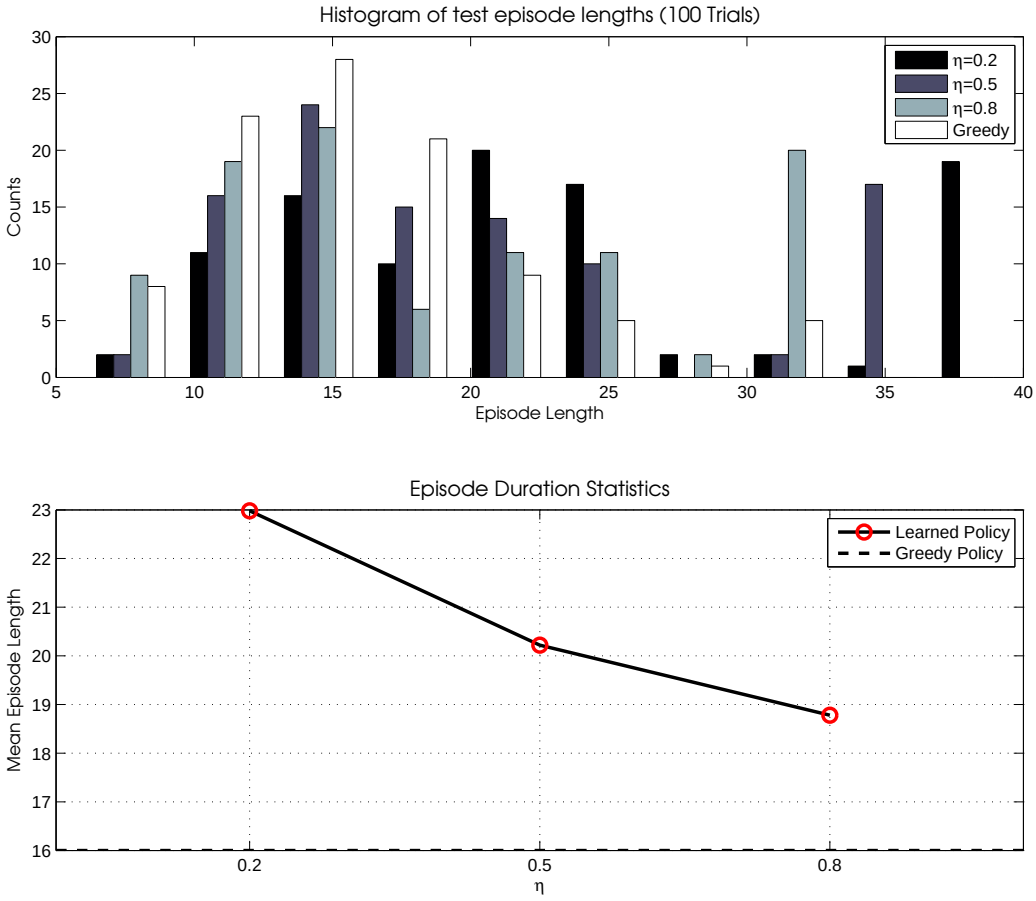


Figure 4.18: Policy test with π_3 policy for a 2-agent 3-task VRP.

The parameters $\lambda = 0.8$, $\epsilon = [.2, .8]$, and $pQ = 0$ were used in these learning trials.

4.9.6 Semi-Markov Vehicle Routing

Section 4.6.7 introduced the Semi-Markov Decision Process modeling formalism for temporal abstraction. SMDPs have the advantages of both providing a general representational framework over MDPs, but also typically reducing problem size. This latter benefit comes as a result of defining a set of macro actions, referred to as options. Most problems require less options than actions and only need a simulator, or controller that can translate between options and actions. Semi-Markov policies are then learned over options as given in Definition 8.

A central claim of this thesis is that routing problems and potentially many other GPPs

are well represented within the SMDP framework. The reasoning is as follows:

- The highest level action in routing or scheduling–allocation–is fundamentally temporally extended.
- The already vast state-action space of a routing problem can be greatly reduced by considering options over actions.
- SMDPs natively support action interruption, which is vital in problems with uncertainty.

In Figure 4.19 the evolution of an allocation problem is represented as an MDP and SMDP. At time step 1, the vehicle can choose from four primary actions, similar to the maze world problem in Example 2. After choosing action a_1 the vehicle advances forward in spaces, choosing again from the four primitive actions until reaching the goal. Consider however, the high-level macro action o_1. At time 1, o_1 advances the state directly to the waypoint at time step 4.

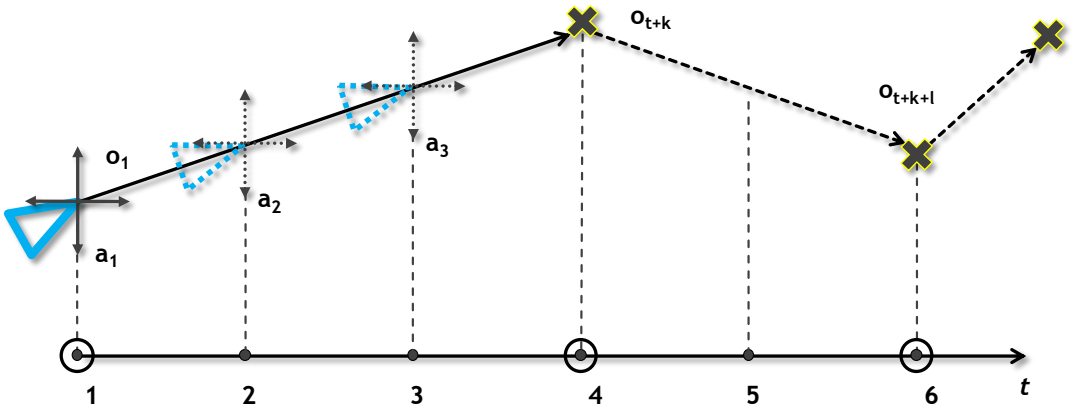


Figure 4.19: SMDP vehicle routing

The vehicle in this setting can choose between 4 primitive actions or 2 options at every time step. The reduction in the number of actions results in a reduction in storage and computation.

In similar fashion to previous sections, learning experiments have been carried out to determine the efficacy of SMDP-learning for a vehicle routing problem. The problem setup is exactly as in Example 6. Q-learning is implemented as in Algorithm 3.2 and SMDP Q-Learning is implemented as in Algorithm 4.1. Figure 4.20 shows learning results, with SMDP-Q performing the best of the tested algorithms. It should be noted that action-interruption as described in [63] and [67]. Is not implemented here. For small numbers of tasks relative to agents and completely deterministic agent, task, and environment dynamics, switching allocations mid-task would not produce optimal behavior. While it is possible

that a switched behavior policy would produce better learning results since it would be more exploratory, the target policy is not switched, and therefore should not be learned.

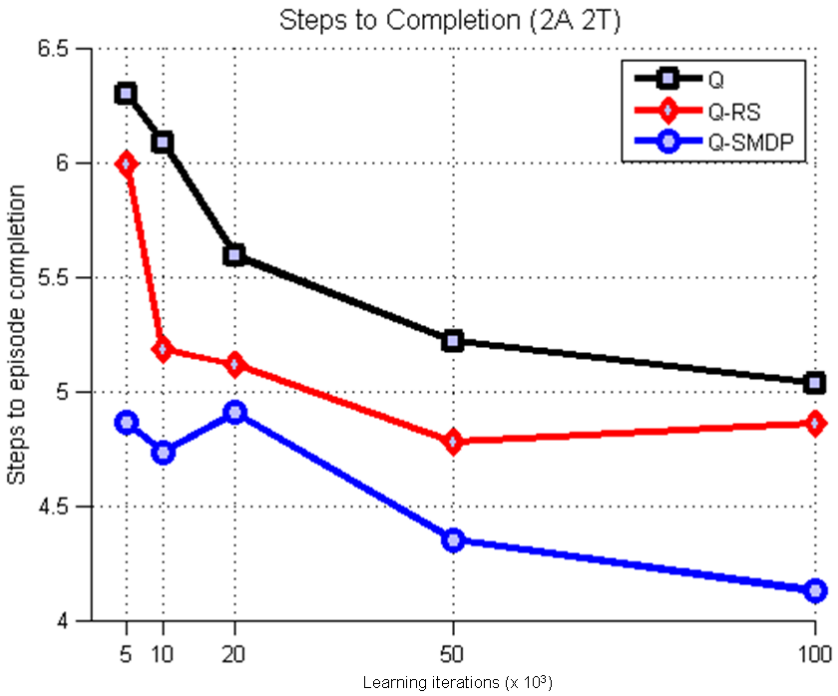


Figure 4.20: SMDP-Q learning curve for a 2-agent 2-task VRP.

Results show learning progress as a function of mean episode length for a MDP representation (Q-learning), MDP representation with a shaping function that rewards subgoal accomplishment as outlined in [50], (Q-RS learning), and an SMDP representation (Q-SMDP).

To determine the end policy quality produced by the SMDP framework versus the MDP framework, the resulting policies were trained and tested for varying numbers of tasks. Figure 4.21 shows results for a 2-agent multi-task scenarios, averaged over multiple episodes. The greedy policy lower bounds the performance of the policies, measured as normalized episode length. Unsurprisingly, all policies perform the same for 2-agent 1-task tests, while the Q and SMDP-Q policies perform the same and are bested by the greedy (optimal) policy for the 2-agent 2-task scenario. Policy performance roughly increases linearly, with the SMDP-Q policy slightly outperforming the Q policy as the load on the system increases.

Due to computation and storage constraints, the problem could not be extended to more agents and tasks. It is expected that at some point, as the greedy policy becomes further from optimal, on average, for a large number of agents and tasks, the SMDP-Q policy would outperform it. The Q policy should in theory converge to the same optimal policy. However, as the problem increases in size, computation limitations dictate that for the same number

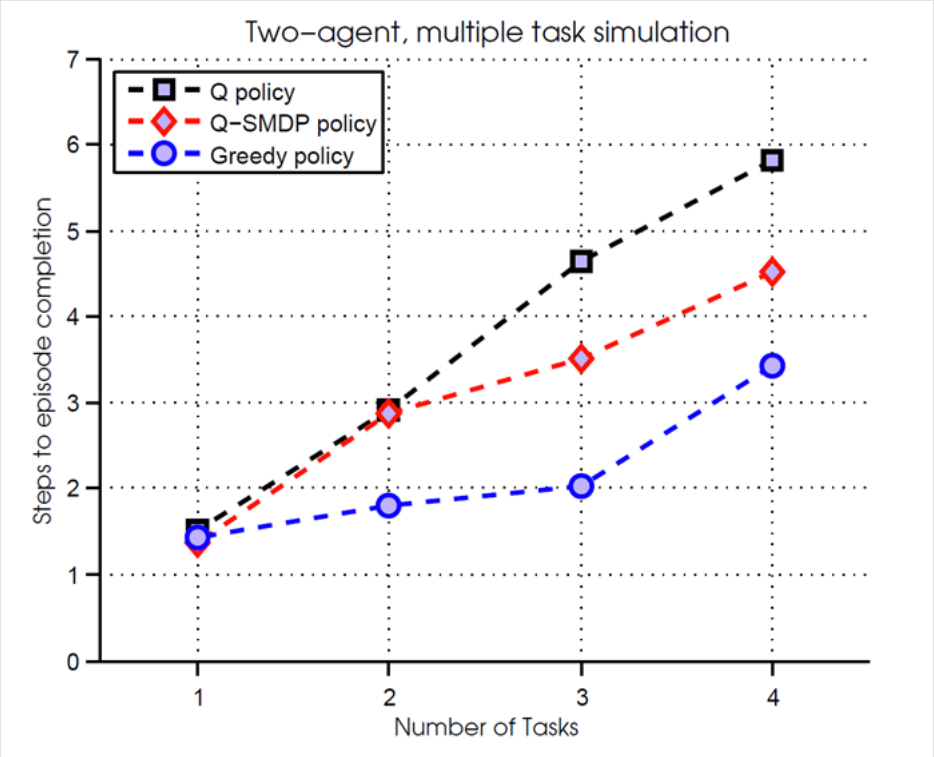


Figure 4.21: SMDP-Q policy test for a 2-agent multiple task VRP

Results from the SMDP representation are compared against an MDP representation and hand-coded greedy policy. Reward shaping is omitted here for simplicity. The greedy policy is optimal for $n \leq m$ and close to optimal from small differences.

of learning episodes, the SMDP-Q policy should be closer to optimal. On average, SMDP-Q learning will update the less numerous state-option pairs than Q-learning will update the more numerous state-action pairs. Storage limitations, a harsher limit, dictate that past a certain size, a problem can not be held in memory, favoring intrinsically more compact representation. Table 4.5 summarizes the performance of each policy as well as the number of states and actions considered in each problem.

Tasks	Agents	States	Actions	Options	Q	SMDP-Q	Greedy
1	2	36	16	4	1.51	1.37	1.43
2	2	729	16	9	2.91	2.86	1.79
3	2	11664	16	16	4.64	3.51	2.02
4	2	164025	16	25	5.81	4.52	3.44

Table 4.5: Multi-agent multi-task simulation results

In all tests, the number of agents is held constant at 2 while the load on the system–the number of tasks–is increased. Episode length is then compared for the three learning algorithms: Q, SMDP-Q, and greedy.

4.9.7 Learning Summary

Previous section have explored the effect of learning parameters on learning performance and end policy performance. Results have shown that learning speed, utility, and policy performance are not proportional or even co-varying, and in some cases, negatively correlated: A small ϵ policy (highly exploitative) for example, will have poor learning performance and utility, but will produce a good target policy.

Parameters were identified by performing simple line search, holding all other parameters constant. Identified optimal values were carried forth to subsequent single variable tests. The assumption is that learning parameters such as λ, ϵ, and η are sufficiently independent. A more advanced analysis could be performed by co-varying parameters or bootstrapping over multiple iterations off of previously identified values. The experimentally determined values not only improve learning and policy performance, but also give insight into underlying problem structure. Table 4.6 summaries the identified values and implications.

	Value	Result
Eligibility Trace	$\lambda = 0.7$	The general trend indicted that both learning performance and policy performance improve for increasing lambda
Exploration	$\epsilon = 0.5$	Lower values of ϵ (more exploration) produced the best policy, while learning performance was best at higher ϵ.
Seeding	$\eta = 0.8$	Following a near-optimal greedy policy, particularly early in learning, results in a drastic reduction in learning time and improved policy performance.
Prioritized Sweeping	*on*	The simple priority queue algorithm reduces both learning time and improves policy performance.
Reward Shaping	*subgoals*	The subgoals shaping function does not improve learned policy performance.

Table 4.6: Learned VRP parameter identification summary

4.10 Conclusion

Having laid the prototypical groundwork for learning in Chapter 3, this chapter is the first foray into applying learning to a non-trivial decision problem. That problem is described as a Generalized Planning Problem, which is essentially a Generalized Assignment Problem, but with an emphasis on blending the routing and scheduling application domains. A formulaic approach is taken to develop effective GPP solutions. The first step is representation, which defines the state and action spaces of a general GPP. The representation chosen is abstract and cost based, with a cornerstone feature of incorporating constraints directly into the state as costs. This design decision comes at the price of granularity–two states that both "cost" 5 may in fact be different–but at the great benefit of abstraction: any problem that can be reduced to agents and tasks and pairings therein is representable under this GPP framework.

Following representation is modeling–the casting of the GPP into a specific modeling formalism. The models chosen here are Markovian graphical models. MDPs are an extremely general modeling formalism, incorporating notions of states, action, stochastic transitions, and a great body of literature exist for computing optimal policies on problems cast as Markov Decision Processs. However, as with many learning problems, the curse of dimensionality precludes all but the smallest of examples. Especially true for the GPP, the joint state and action spaces grow exponentially with the number of agents, tasks, or resolution of the cost state variable. To remedy the explosion of states, and in addition to judicious choice of state representation, hierarchical models are explored. The Semi-Markov Decision

Process model in particular not only reduces the action space by considering a hierarchy of actions, it allows for temporal abstraction, and is a generalization of the MDP model. SMDPs introduce the options framework–macro temporally extended actions–which have both intuitive and practical benefits for GPPs such as the VRP.

Learning optimal policies on MDPs and SMDPs is a well-studied area of research. Modeless temporal difference algorithms exist for both. Both Q-learning and SMDP-Q are implemented on an example VRP, cast as a GPP. The cost state is simple distance between all tasks and agents. Constraints, such as preventing dual allocations, and disallowing loitering are explored through selective rewards and penalties in the reward function.

Learning results are presented for a 2-agent 3-task VRP in Example 6. Following from Chapter 3, learning is optimized for several tuning parameters: eligibility traces, exploration, prioritized sweeping, and reward shaping. Additionally, a new state space exploration technique referred to as η-search is introduced. The technique attempts to expedite learning by incorporating guidance from the greedy VRP solution, which is suboptimal, but easy to compute. The parameter η controls the blend of exploration (random), exploitation (greedy with respect to the value function), and sub-optimal greedy VRP actions. The results show a significant improvement in learning speed when implementing an η-behavior policy during learning. And for large learning problems, increased learning speed often results in increased performance, and more relevant states are visited more often for a constant learning duration.

The final learning results are for semi-Markov vehicle routing. Though other work has applied SMDP theory to robotics decision problems, to the knowledge of the author, this is the first application of SMDPs directly to the canonical VRP. Results show a significant increase in learning performance and end policy quality when considering semi Markov-options over primitive actions. Beyond the apparent computational and storage benefits, the SMDP is conceptually a well-suited model for the GPP. GPPs are intrinsically temporally abstract–the fundamental action and agent can take, allocating itself to a task, takes place over multiple time steps for a random and indeterminate amount of time.

The purpose of this chapter has been to show the expressive power of the learned GPP. This has borne out in simulations of a canonical NP-hard optimization problem, the VRP. The final chapter addressed the incorporation of human feedback to aid in learning solutions to an ever more challenging stochastic GPP.

Chapter 5

Human Guidance

5.1 Introduction

Human beings are decision making engines. From low-level automatic activities, such as walking, to high-level reasoning, our brains readily solve complex perception and decision problems in real time. On the other end of the spectrum, computers and robots have proven particularly useful in dull, dirty, and dangerous domains: inspecting nuclear disaster sites, solving huge state-space games, and generally accomplishing tasks difficult for humans. As such, human input can and should be leveraged wherever possible to learn faster and better solutions to complex decision problems.

The Generalized Planning Problem architecture, introduced in at the conclusion of Chapter 2, is primarily associated with decision problems at higher levels of abstraction. Although it can be applied to lower-level, more classical control problems, those problems are, by definition, better defined than more abstract allocation, planning, and decision making tasks. Figure 5.1 depicts this spectrum as a control system diagram, from assignment, to planning, to lower-level problems such as motion control.

Section 5.2 further motivates the reasons for incorporating human guidance into the GPP. Section 5.4 introduces the specific reward shaping-based architecture used herein as a conduit for human guidance. To showcase both the expressive power of the GPP, the benefits of human-guided learning, and the specific implementation used here, Section 5.5 provides an example problem as a testbed. The complex decision problem is essentially a taxi pickup and delivery problem which incorporates several stochastic elements as well as human guidance. Section 5.6 presents the results from learning, and Section 5.8 draws several conclusion from these results.

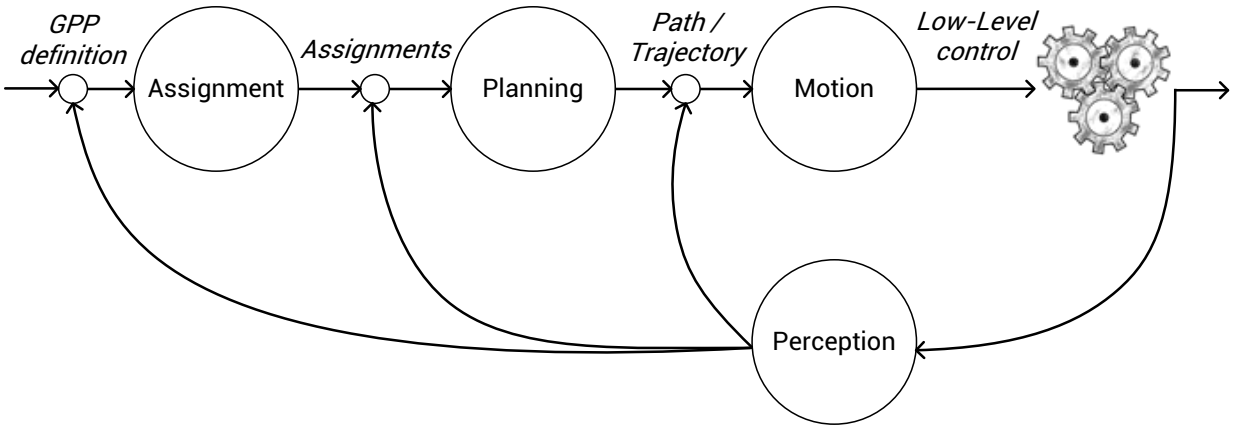


Figure 5.1: Multi-agent control system.

5.2 Motivation

Many hard decision problems display common features. Vast state spaces, ambiguous objectives, and uncertainty of any kind can render a problem practically unsolvable. However, humans display great aptitude at solving some of the most difficult decision problems. One reason for this is our innate ability to infer meaning from context. A car driving down a neighborhood road at noon on a Saturday poses less danger than a car on the highway on a rainy night. Context is how we distill our past experiences to make inferences about current situations. Context, being an inherently vague concept, is challenging to encode in an algorithm.

The second more nebulous uniquely human skill is our ability to think abstractly about groups and similar situations rather than just on a per example basis. Classification is one relevant example that both humans and computers attempt to solve. In classification, a visual processing system (human or computer) attempts to apply labels to objects, effectively grouping them into classes. The challenge comes in understanding objects at all scales, shapes, sizes, and permutations. Humans, for example, could label all of the chairs in a room without error, and even all of the objects that would provide suitable seating. A computer-classification system can not, however, think in this manner, and can only classify chairs based on examples of chairs it has been given, let alone answer the even higher-level question of suitable seating surfaces.

The two human skills that often far surpass their computer-based counterparts are contextualization and abstraction. Human aptitude at solving hard decision problems by applying these abilities is a chief motivating factor human-guided learning. Formal definitions are as follows, as well as a depiction in Figure 5.2.

Contextualization *The ability to think about or provide information about the situation in*

which something happens. Humans use the context of a situation–the surroundings–to infer meaning. Analogous computer contextualization includes function approximation, which systematically infers a functional context from existing data.

Abstraction *To consider as a general quality or characteristic apart from specific objects or instances*. The human ability to label an entity as part of a group–*chairs*–place that group in a hierarchy–*recliners in chairs in seating*–then reason about that object as part of a group is unmatched by computers. An analogous computer abstraction procedure is classification, which clusters data according to some similarity metric.

What does human intuition have to say about a GPP such as a routing problem? This question is primarily answered in the results section, where human guidance is directly incorporated into the learning via a human reward signal. Suffice it to say that understanding context of a GPP and being able to abstract away low-level detains can certainly augment existing optimal VRP solvers.

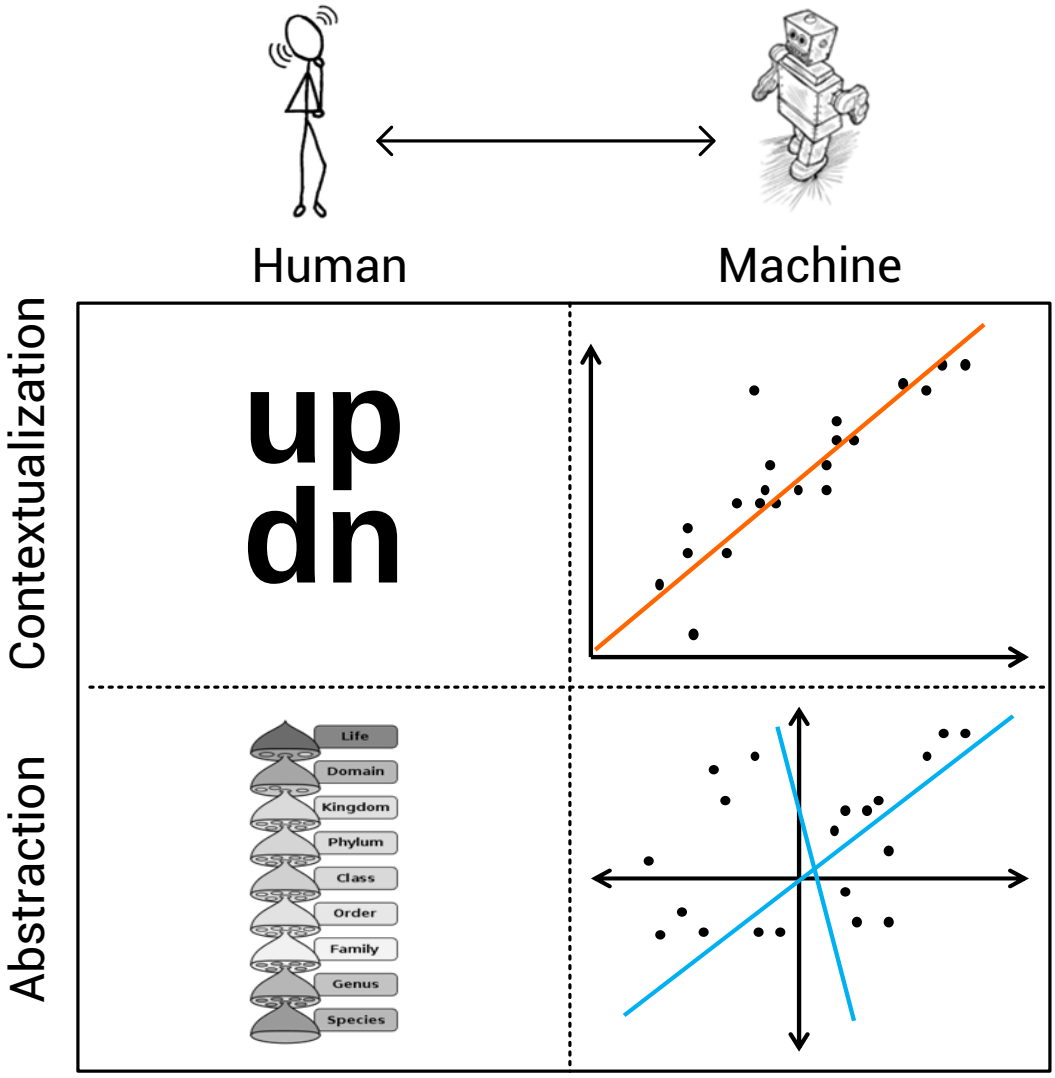


Figure 5.2: Human - machine attribute comparison.

On the upper right is a pictorial representation of function approximation, which is analogous to human contextualization. On the lower right is a pictorial representation of classification, which is analogous to human abstraction.

5.3 Previous work

Advances in robotics and computer science have only served to underscore the difference between humans and robots, namely that robots tend to exceed at lower level structured tasks, while humans are unmatched at higher level decision making, particularly when problems involve unstructured data, soft constraints, and ambiguous objectives.

Human-Robot Interaction (HRI) is a broad field of research, touching upon human factors, autonomous systems, adjustable autonomy, and human in the loop learning [30]. The authors of [70] provide a simple example of incorporating human feedback via a reward signal for learning how to perform a sequential cake baking task. Other work focuses on large scale problems in which robots learn a human model for the purpose of determining online when to query a human operator [31] [32]. Work specific only to multi-agent problems of any domain has studied broadly applicable performance metrics [14].

A recent advancement in the field of reinforcement learning has been apprenticeship learning. Apprenticeship learning involves an expert (human) who teaches the apprentice (learning algorithm) by way of example. Through a learning mechanism, often inverse reinforcement learning, the reward function of the expert is learned, which can then subsequently be used to learn a policy for control [1].

5.4 Human Feedback Architecture

How and when should human guidance, of any kind, be included in a decision making process? Any decision making process will have three distinct phases during which a human operator can add advice, as shown in Figure 5.3. In the setup phase, the problem is defined and relevant parameters are initialized. The next phase, labeled "Learn" in Figure 5.3 solves the problem, and the final phase, "Run", implements the solution. For many problems, setup transitions directly to solving, which transitions directly to running. In the learning framework, learning solutions and implementing them online are often interleaved, with a (sub-optimal) policy available to the problem designer at any point in time; this is known as the *just-in-time* algorithm property.

A human is likely to be involved in the initial setup phase of a learning problem, but not in feedback. During the learning phase, a human can add advice based on learning performance feedback (learning curve, or utility, for example). And while running the policy, a human can make corrections based on policy performance. The two paradigms of human advice are therefore precisely defined below with associated definitions

- **Operator**: A human who sets up a GPP, including original placement of tasks, agents, and specification of know problem dynmaics and constraints therein.
- **Learn Time**: The learning phase of a problem during which policy improvement occurs and $V^{\pi} \rightarrow V^{*}$.
- **Run Time**: The phase during which a policy is used to determine the actions of a system. Run time can refer to simulation or real world action.

***

- **Learn-time human feedback:** To include human feedback to improve learning time while preserving policy optimality.

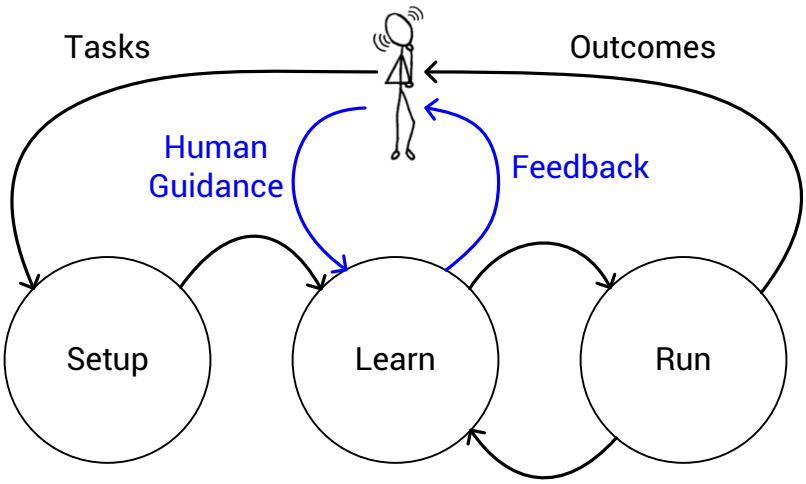


Figure 5.3: Human feedback learning architecture.

In the setup phase initial variables and objectives are defined. In the learning phase, a simulator or motion model propagates the state forward it time; the system receives reward and an update rule (Q, SARSA, value iteration, policy iteration, etc.) incorporates the reward into a value function. A policy may be drawn from the learning algorithm at any point and tested online. A human operator observing the autonomous system evolve may choose to input advice in the form of an exogenous reward signal, resulting in online learn-time human feedback.

- **Run-time adjustable autonomy:** To create human-assisted systems, where the operator can choose any value between manual and autonomous control.

As has been highlighted in subsection 3.4.3 of Chapter 3, TD-algorithms, which are based on bootstrapping, essentially interleave learning and running. If learning and running are interleaved as shown in Figure 5.3 then human guidance can be added just during the learning phase. This proposed human feedback architecture is agnostic to the specific guidance signal and algorithmic details of how the signal is incorporated. Section 5.5 proposes an intuitive conduit for incorporating human feedback based on reward shaping.

5.5 Human-in-the-loop Learning

Consider the scenario depicted in Figure 5.3 in the context of a vehicle routing problem GPP. After the human operator has defined a set of tasks, a goal (complete all the tasks) and other problem parameters, an (offline) learning algorithm solves for the optimal allocation of agents to tasks. That solution is then implemented online, or the learning phase itself may have occurred in an online fashion of the learning algorithm can run in real-time. The operator

observes the outcome of the problem–namely agents completing tasks in a certain order–and may at that point define a new problem.

However, two run-time issues may occur that would motivate the human operator to make online changes to the learned policy. First, the operator may be dissatisfied with the solution because they defined the problem poorly in the first place. This is the case of operator error. Second, unmodeled agent dynamics, random environmental events, or uncertainty in task location, may necessitate a run-time change to the problem definition in order to compensate for unpredictable occurrences.

Whether due to operator problem definition error or unforeseen problem dynamics, the outcome is the same: the operator wants to make changes to the learned solution. Learning algorithms discussed up to this point leave the operator with two options: redefine the problem and solve from scratch, or make ad-hoc corrections to the solution, losing all optimality guarantees.

The third and alternate solution proposed here is to leverage the theory of reward shaping as introduced in subsection 4.9.4 to incorporate human feedback in a structured fashion.

5.5.1 Incorporating Human Feedback via Reward Shaping

The theory of reward shaping allows for construction of reward functions as a sum of individual rewards. The proposed total reward R is comprised of an intrinsic reward, R_I, and a human-inputed reward, R_H. The intrinsic reward is the original reward that gives positive reward at the goal state(s) and negative elsewhere. The human reward is simply ascertained by having a human observer assign a value to a full state transition sequence, defined by the tuple $< s, a, s' >$. The full reward function is given in Equation 5.1.

$$R(s', a, s) = R_I(s', a, s) + R_H(s', a, s) \tag{5.1}$$

$$R_H(s', a, s) \triangleq \gamma\Phi(s') - \Phi(s) \tag{5.2}$$

The shaping function, R_H, is admissible if and only if it can be written as a potential based shaping function as in Equation 5.2. This guarantee can never be met so long as the human is permitted to give any and possibly faulty guidance. If an admissible $\Phi(s)$ were known ($V*(s)$ is suggested in [50]), then R_H could be tested with Equation 5.2 as a criteria. However, in that case, a reward function based on $\Phi(s)$ should just be used rather than R_H, unless R_H could be shown to be a more effective reward than the one based on the potential function.

Ng, Harada, and Russell note that in fact *any* policy is optimal under R if it is potential-based. Therefore potential based shaping reward functions have no policy preference, they do not prefer one policy over another, which is the reasoning behind why all policies optimal in the original MDP are optimal in the shaped MDP. The human reward need only be potential based to be admissible, leaving the optimal policy due to the intrinsic reward unchanged, and speeding up learning in the process.

5.5.2 Adjustable Autonomy

In many real-world autonomous systems, human operators primarily provide supervisory oversight over the operation of the system. If the unexpected occurs, they are left with either allowing the system to continue to execute, or shutting it down, taking over manual control. With the addition of human-based learn-time guidance via reward shaping as described in the previous section and depicted in Figure 5.3 a human operator is allowed to make learn-time changes to the learned value function. Changes are incorporated as an exogenous human reward signal using the reward shaping architecture. However, changes will not be incorporated immediately, as they will depend on learning parameters, including learning rate, discount factor, and others.

A useful addition would be a framework for hybrid human-machine continuously variably autonomous systems. Such a system would seamlessly transition between full manual operation, full autonomous operation, and everywhere in between. The following modified shaping function is therefore proposed in Equation 5.3 and depicted in Figure 5.4 based on a simple weighted sum of intrinsic and shaping reward functions.

$$R(s', a, s) = (1 - w)R_I(s', a, s) + wR_H(s', a, s) \tag{5.3}$$

$$w \in [0, 1]$$

The proposed weighted sum combination of rewards preserves policy optimality. The original MDP, $M =< S, A, P, R_I >$, is reward function scale invariant: Policies under R_I are optimal under $(1 - w)R_I$. Additionally, the only requirement for the shaping function is that it be potential-based, which is also scale invariant, in that if R_H is potential based, so too is wR_H.

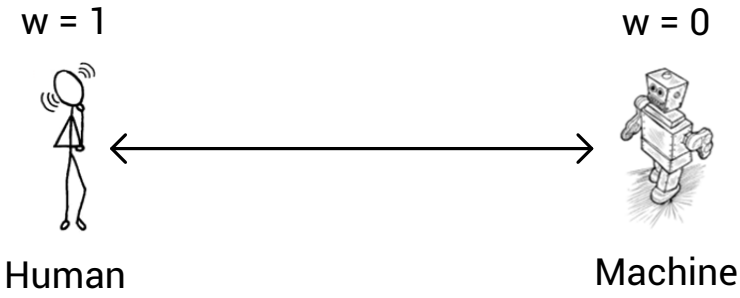


Figure 5.4: Adjustable autonomy via reward shaping.

The parameter w variably weights the intrinsic reward (standard reward function) with the human reward (shaping function)

5.6 Variably Autonomous Planning

The previous section proposes a method for incorporating weighted human feedback as a reward signal is a learning problem. In this section, an example GPP is introduced to showcase human-in-the loop adjustable autonomy. The problem is based on a well know routing problem variant known the VRP with Pickup and Delivery. The problem characteristics can be described in brief as follows:

- Multiple agents seek to accomplish a set of pickup and delivery tasks.
- Tasks consist of a pickup site and a delivery site.
- Agents have a capacity and tasks have a "weight" (how many units of agent capacity they require).
- The goal, like in vehicle routing, it to complete all tasks with minimum aggregate distance traveled (or time for constant velocity vehicles).

The VRPPD is adds complexity to the standard VRP which ultimately results in larger state and action spaces and more constraints. Stochastic VRPPD are perhaps some of the most complex routing problems. Of the surveyed current literature, VRPPDs incorporating stochastic elements tend to allow only one random problem parameters, while the rest are deterministic. A complex spatiotemporal routing problem proposed is proposed in Example 7 with multiple stochastic elements.

Example 7 (Taxi Routing)**.**

Multiple taxis are tasked with completing a set of pickup and delivery tasks. At the outset of the problem pickup locations are known along with estimates of the distance to the destination. After a task has been picked up by an agent, the true destination is revealed, and the taxi proceeds directly to the destination. The goal state is after all tasks have been picked up and delivered. Problem parameters are listed below or in Example 6 and defined in Chapter 4 if not specified.

Parameter	**Value**	**Description**
m	3	number of pickup/destination tasks
n	2	number of agents
w	$5\,km$	$w \times w$ world size
v	$15\,m/s$	agent velocity
$R(s', a, s)$	*see below*	reward function
p_fail	0.05	agent action failure probability
λ	0.7	eligibility trace parameter
ϵ	[0.3.08]	exploration probability
η	0.8	seeding parameter
w	$[.1, .3, .5, .7]$	shaping weight
P_j	$U(-w, w)$	pickup locations - uniform over the space
D_j	$N(P, \Sigma)$	destinations - normally distributed about P
E_j	$N(P, \Sigma)$	estimated destinations - same distribution as D

The overall reward is constructed as a weighted sum as in Equation 5.3. The intrinsic reward includes the static reward for reaching the goal, $R_g = 3$, and a temporal penalty, $R_t = -0.01$, for every second of travel. The human reward is as follows

$$\Phi(s) = -F(\mathbf{J}(s)) \tag{5.4}$$

$$R_H = F[\mathbf{J}(s)] - \gamma F[\mathbf{J}(s')] \tag{5.5}$$

$$F(\mathbf{J}) \triangleq \|\mathbf{J}\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|J_{ij}|^2} \tag{5.6}$$

Several additional constraints and assumptions imposed by the simulator are described in Appendix A.

The human reward function described in Example 7 is simulated based on observed human behavior. After several qualitative experiments in routing-like scenarios, it appears that the human operators value state transitions that bring all agents closer to tasks on average. Unfortunately, this strategy is suboptimal for the VRPPD; many scenarios arise in which agents need to move away from the closest task in order to accomplish all tasks in minimum time. Nevertheless, a norm-based human reward function is selected, which awards value proportional to the *decrease* in aggregate agent-task distances. The Frobenius norm in Equation 5.6 therefor selected, since the state is represented as a non-square matrix: $J \in \Re^{n \times m}$.

It is easy to construct counter-examples in which Equation 5.5 is not potential-based. However, overall the cost of the problem (even if cost is just distance-based) should be decrescent, as the cost-state is explicitly designed to do so. Additionally, in practice, the human guided reward will be inputed by a human operator rather than a simulator; therefore a realistic human simulator should be suboptimal to reflect the fact that the human reward signal will not always obey the conditions for optimality under reward shaping.

5.7 Learned Stochastic Pickup and Delivery Results

The problem described in Example 7 is a stochastic VRP with Pickup and Delivery. It however departs from the standard formulation in that all constraints are soft and reflected as costs in the state. All goals are aggregated in the reward function, and all other problem dynamics are simulated. The cost (state) and simulator each have stochastic elements. The learned policy represents the confluence of effects from the cost, reward, and simulator.

As in Example 2 in Chapter 3 and Example 6 in Chapter 4 learning performance is judged based on utility and learning curves, and end policy quality is judged by successive simulation. It is important to note that, unlike in previous examples, there is no true baseline optimum by which to judge the learned policies. If the problem at hand were a standard VRPPD, a third-party solver could compute optimal baseline policies. However, no standard solution method exists for the switched-stochasticity introduced by the random destination location, which is realized only upon pickup. In lieu of an optimal baseline comparison, a handcoded greedy policy provides some fixed point for comparison.

In order to demonstrate the representational power of the GPP method as applied to this problem two important cases are compared side-by-side. The metrics for comparison in both cases are cumulative distance–the sum distance of all vehicle routes–and episode length: the total time for episode completion. The episode length criteria is similar to the makespan criteria in the JSP–the time between state and finish of a job, made up of tasks.

Noloiter This case is inspired by vehicles such as fixed-wing aircraft, which can not loiter, ie. their velocity is constant. In this case, the episode length cumulative distance metrics will be the same, as cumulative distance traveled must increase with time.

	Example 6: Routing	Example 7: Taxi
Eligibility Trace (λ)	0.7	0.7
Exploration (ϵ)	0.5	[0.3, .08]
Seeding (η)	0.8	0.8
Prioritized Sweeping	*on*	*on*
Reward Shaping	*subgoals*	*human reward*

Table 5.1: Learned VRP and taxi problem parameter identification summary

Minor discrepancies exist between the two problems. A variable exploration probability seemed to work better for the more stochastic taxi problem. The same handcoded greedy solution is also used for both problems, and seeding parameter η is kept the same.

Loiter This case is inspired by vehicles such as ground vehicles that can come to a stop and loiter at a location for an indefinite amount to time. In this case, cumulative distance is the appropriate metric for discerning policy quality, as it is the only way to capture the learned loitering behavior.

5.7.1 Learning Parameters

In all cases, learning parameters $(\alpha, \epsilon, \eta, \lambda)$ are set based on a similar tuning process the the one used in the routing problem in Example 6 in Chapter 4. The parameters listed in Table 4.6 are used as starting points. In general, the optimal parameters for the taxi problem (VRPPD) differ very little from the parameters for the routing problem VRP). Minor discrepancies in learning parameters are a testament to the generality of the cost-based representation and GPP framework. Learning parameters are listed below in Table 5.1 for both problems.

5.7.2 Learning Algorithm

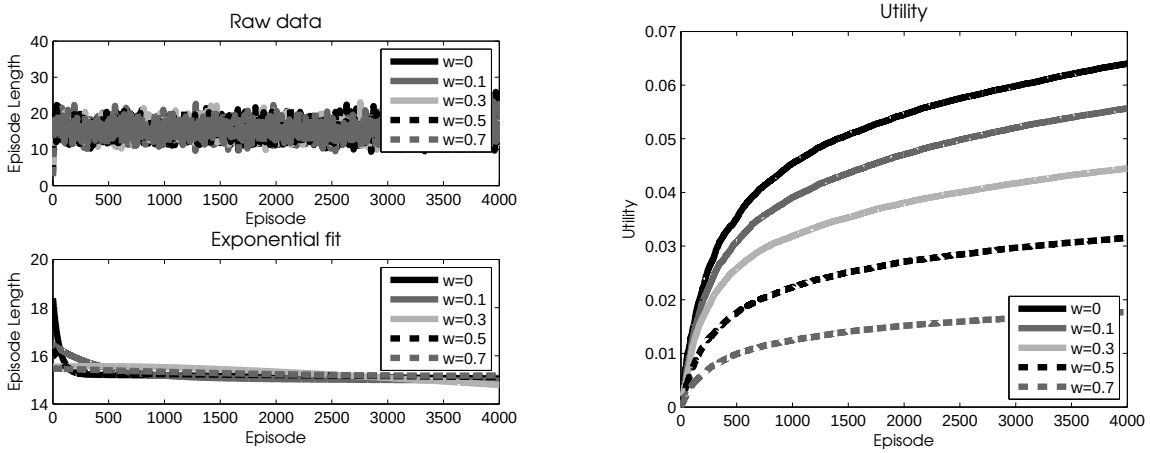
As demonstrated at the conclusion of Chapter 4, the SMDP-Q learning algorithm can improve both learning time and end policy quality over standard Q(λ) learning. The purpose of this experiment is to explore the benefits of variably weighted human-guided reward shaping. As such, learning is performed by Q(λ) rather than SMDP-Q.

5.7.3 Learning Performance Under Human Guidance

Experiments have been carried out to both determine the efficacy of human-guided reward, specifically for the simulated human reward in Equation 5.5, and the appropriate the weighting, w, of the intrinsic reward versus the human reward. This section compares the learning

performance for varying w, and the following section compares end policy performance, which, as explained in Example 6 are not always directly related points of comparison.

Figure 5.5 shows learning results for the noloiter case: ie. agents are aircraft-like, and must always move with constant velocity. The trend reflected in the utility and learning curves to some degree is that increasing w decreases utility over the learning horizon, and may in some cases slow learning for high w. However, this trend is to be expected, particularly for the utility, since wR_H is negative everywhere, and increasingly so for increasing w. The final value of the utility is therefore somewhat meaningless, since the underlying reward function is changing. The general trend in Figure 5.5b is qualitatively the same for all w, which indicates that the learning problem is not fundamentally changing (ie. utility smoothly increases and appears to approach a steady state value in all cases).

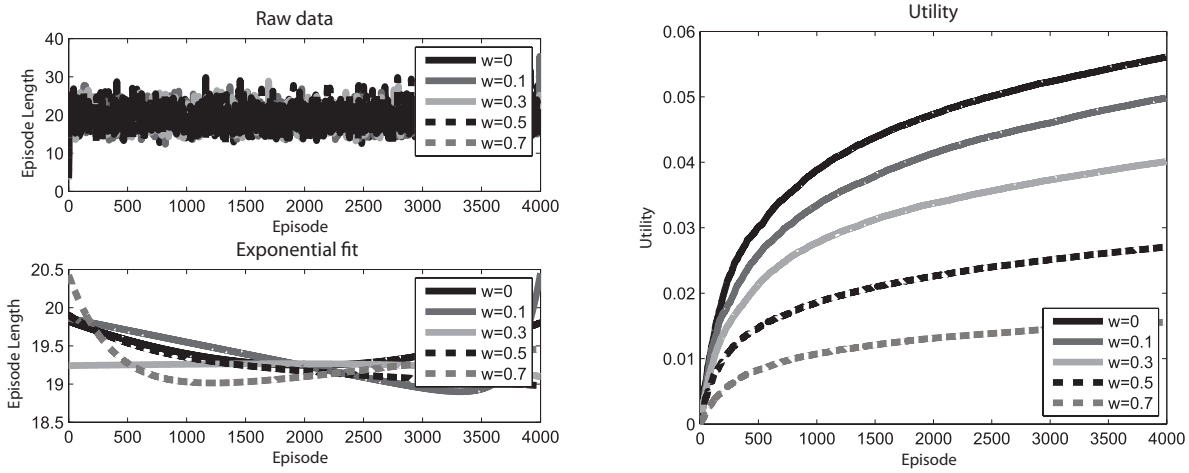


(a) Learning Curve - Noloiter

(b) Utility - Noloiter

Figure 5.5: Taxi-routing learning curves for the **noloiter** case, varying w.

Figure 5.6 shows learning progress for the loiter case: ie. the agents are ground vehicle-like, and can essentially park. Despite this significant change to the underlying agent dynamics, it appears that the value-function utility perspective in Figure 5.6b remains the same in terms of trend. Several of the fitted learning curves in Figure 5.6a seem to vary widely, which is likely due to outliers in the data. In both the noloiter and loiter cases, the learning curves do not provide much information. The learning curves are consistent with the results in Example 6 for low ϵ (highly exploratory behavior policy). Here, ϵ begins at 0.3 and increases linearly to 0.8 over the learning horizon (4000 episodes). Therefore is expected that at least early in learning curves would be high-variance for all weighting w, since random actions are being taken more often than actions greedy with respect to the current value function.



(a) Learning Curve - Loiter

(b) Utility - Loiter

Figure 5.6: Taxi-routing learning curves for the **loiter** case, varying w.

5.7.4 Policy Performance Under Human Guidance

The previous section reveals that the value function is approaching a fixed point for all w. However, due to the stochasticity in the problem (random agent motion, uncertain destinations, etc.), as well as a high degree of exploratory learning, few trends are discernible in the learning curves.

This section explores policy performance, which, is the primary point of analysis for Example 7. Noloiter cases are explored first, followed by the more relevant loiter cases, since real taxis would have the ability to park in low work-load scenarios. The two previously described performance metrics–episode length cumulative distance–are compared as well. Cumulative distance, as opposed to episode length, should be able to discriminate between the cases in which loitering is allowed, since an agent that can loiter with zero velocity will not add to the overall tour length.

In all cases, tests are performed by generating 100 random initial states. Each policy ($w = [.1, .3, .5, .7]$) is then simulated from each initial state to the goal state as runtime statics (episode length, cumulative distance) are recorded. Outliers beyond a 2σ bound are rejected, and a timeout is set to address rare but possible deadlock and livelock scenarios. The outliers can clearly be seen in the histograms. All statistics are normalized and dimensionless so that problems with different parameters—agent velocity, world size, etc.—can still be compared.

Figure 5.7 shows the policy test results for the noloiter case. The first obvious result is that the data episode length data in Figure 5.7a and Figure 5.7b are off by a constant factor, since agent velocities are constant and agents can not loiter. The shape of the curves reveals the following interesting result: for no shaping ($w = 0$, $R = R_I$), performance is slightly worse than the baseline (greedy policy). For $w = 0.1$, which essentially weights the simulated

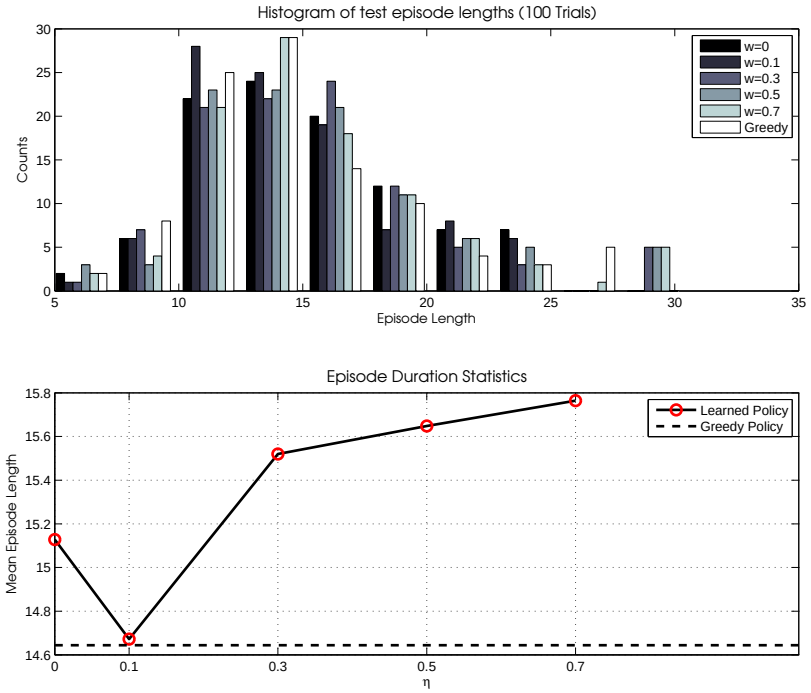
human reward by 10% and the intrinsic reward by 90%, performance is approximately equal to the baseline. This increase in performance is remarkable considering that 1) the shaping function R_H is *not* potential-based, and 2) the problem has a high degree of stochasticity, particularly the destination locations.

The second noteworthy result for the noloiter case is that for $w > .1$, performance immediately degrades and appears to settle at a suboptimal level near $w = 0.7$. Possible explanations are as follows:

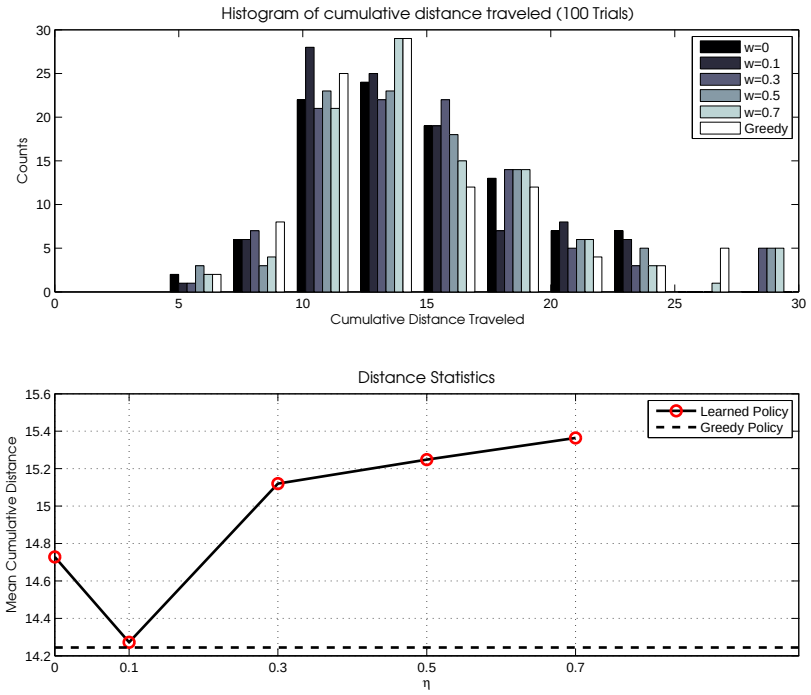
- *Choice of shaping function*: Since R_H does not satisfy the necessary and sufficient reward shaping conditions for policy invariance under reward shaping, the learned policy, while monetarily improved for $w = 0.1$, is in general not improved by the human guidance.

- *Poor numerical conditioning*: As R_H is increasingly scaled up relative to R_I, the goal reward (+3) must also be increased to counteract the negative effect from the human-based reward.

- *Stochasticity* While stochasticity is the same for all problem scenarios, w directly scales R_H which is a function of the cost-state, which is itself stochastic. As w increases, the randomness is amplified more relative to the true state.



(a) Episode length policy test for the noloiter case



(b) Cumulative distance policy test for the noloiter case

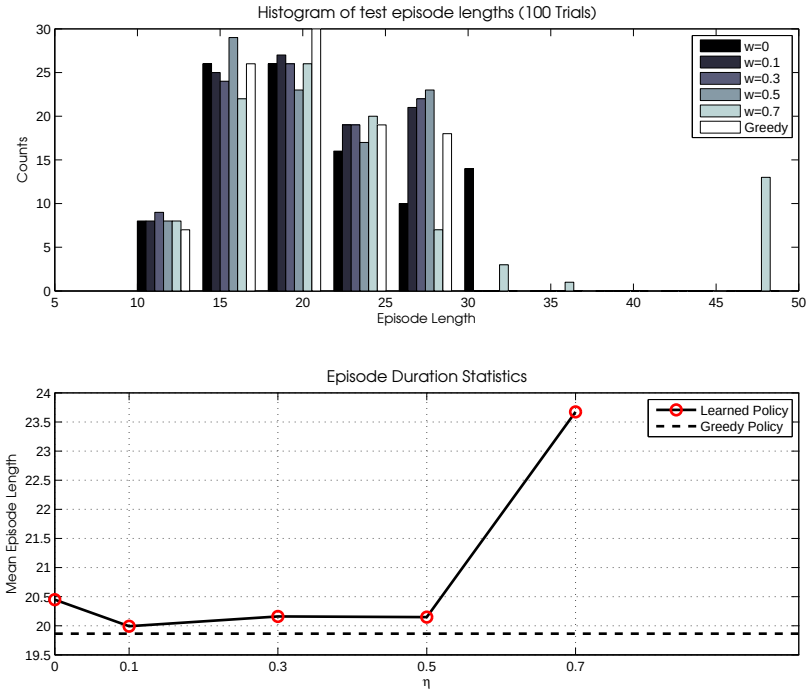
Figure 5.7: Taxi-routing policy tests for the **noloiter** case, varying w.

Figure 5.8 shows results for the loiter case, where agents are essentially allowed to park, and therefor learn policies with this behavior. It is important to note that changing the underlying simulator would not simple be enough to encourage loitering behavior. But by the same token, loitering is not explicitly encouraged in the reward function. The behavior is essentially undefined. However, the reward function does not encourage motion wither, and the initial policy is to loiter. Therefore, while agents my learn to move around without any purpose due to random actions during the learning process, there is no benefit to doing so, so the target policy tends towards favoring loiter actions wherever possible.

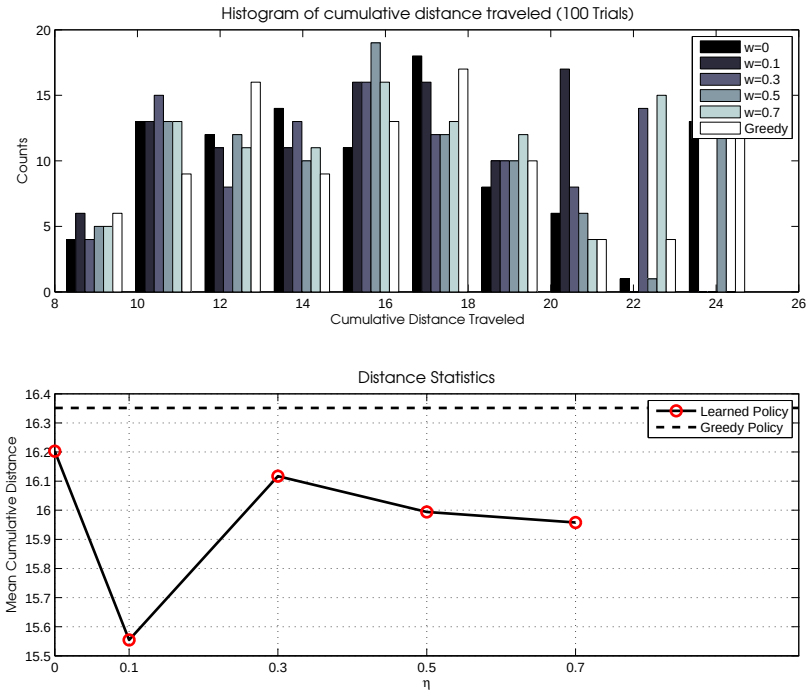
Figure 5.8a shows data collected on overall episode length over the 100 trials for varying weight w. A slight improvement in the policy is seen for $w = 0.1$, with little change to policy performance thereafter ($w = [0.3, 0.5]$). The first policy that weights R_H more relative to R_I ($w = 0.7$), degrades in performance drastically. In this case, the human reward seems to provide marginal benefit up to the point of overtaking the intrinsic reward, after which the suboptimal human reward degrades the quality of the policy.

Figure 5.8b shows the same test, but from a perspective of distance rather than time. As anticipated, allowing the agents to loiter results in a significant increase in policy quality. This is the first policy that out performs the greedy baseline policy, which does not allow loitering. The result is somewhat obvious in that agents that are allowed to park should cover less distance than agents that are not. Less obvious though is the fact that the agents in fact learned this behavior through a combination of intrinsic and simulated human reward. The policy performance follows a similar trend to that seen in Figure 5.7: $w = 0.1$ is by far the best policy, with a degradation in performance thereafter.

This final result validates that the chosen human reward stimulant indeed improves performance with the right weighting, as it did for the noloiter case.



(a) Episode length policy test for the loiter case



(b) Cumulative distance policy test for the loiter case

Figure 5.8: Taxi-routing policy tests for the **loiter** case, varying w.

5.8 Conclusions

Human guidance in learning is conceptually an idea worth exploring, since humans have shown great promise at solving decision problems. The innate human cognitive abilities, abstraction and contextualize allow us to automatically recognize salient problem features, reason about these features in a broad sense, and make decisions based on this reasoning. Automated intelligence essentially tries to encode human ability as classification, function approximation, and many other well-studied techniques. Figure 5.9 captures the overall architecture of hybrid human-machine decision making within a multi-agent control system.

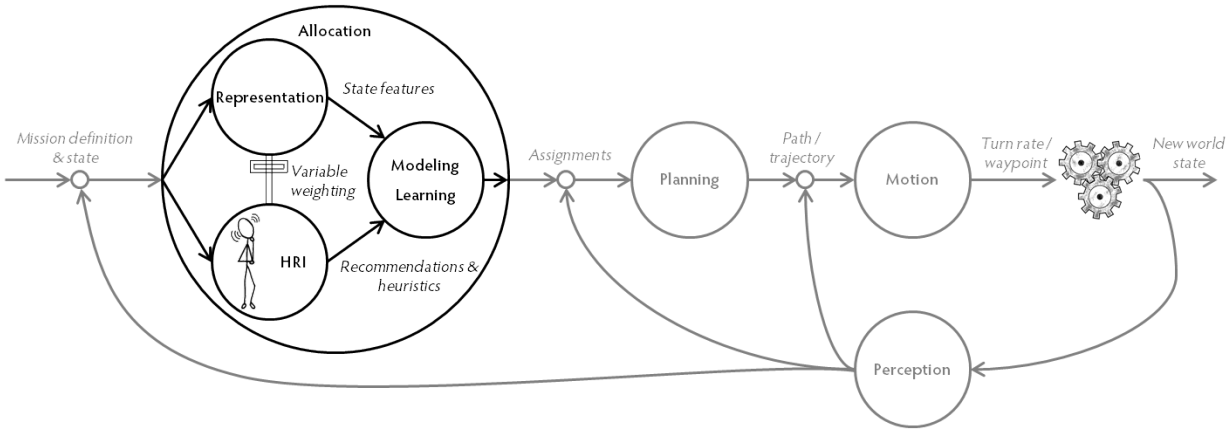


Figure 5.9: Multi-agent control system.

The approach taken here first recognizes the innate ability humans have at solving routing and scheduling-like decision problems, which have broadly been referred to as Generalized Planning Problems. It makes no assumption on the form of the input other than providing humans with a conduit for valuing certain problem configurations over another. The theoretical underpinning for this human-guided evaluation is reward shaping. This perspective and method is, to the knowledge of the author, the first time that human guidance is incorporated in such framework that can provide optimality guarantees. The approach assumes nothing other than that the problem can be modeled as a GPP as described in Definition 6.

Reward shaping allows for the combination of two reward signals, referred to here as the intrinsic reward (R_I, standard reward) and the human reward (R_H, shaping reward). A modification of the standard reward shaping formulation inversely weights these two rewards, allowing for a range of learn-time possibilities, from full automated ($R = R_I$) to fully manual/human-based ($R = R_H$), and everywhere in between. This augmentation to human-based reward shaping is an implementation of true adjustable autonomy.

The testbed to showcase human-guided learning and adjustable autonomy is a stochastic VRPPD, which is essentially a taxi routing problem. Several problem elements are set as

random to show the expressive power of the chosen cost-based representation over more standard VRP-derived representations. Additionally to motivate the modeless learning-based approach itself, underlying agent dynamics are changed at the simulator level, and new policies are automatically learned, without any other changes to the problem setup.

The results in Section 5.7 show two cases: the noloiter case in which the agents are not allowed to park, and the loiter case, inspired by actual ground vehicles such as taxis, are allowed to park. Tests are carried out at several weighting values, although an infinite number are possible since w, the adjustable autonomy variable, is continuous.

The results show both that the adjustable autonomy concept can work, and in fact produces the best end policies for $w = 0.1$. One conclusion is that the choice of human simulator is not a good one, but it is purposely chosen to be suboptimal, as a real human would be. The overall results show that for all values of w reasonable results are obtained, which is an encouraging validation of the setup and procedure for incorporating human feedback in an intuitive, meaningful, scalable, and theoretically sound manner.

Chapter 6

Conclusions

This thesis has presented a new methodology for solving complex decision problems. The following sections summarize contributions and pose future research directions.

6.1 Summary of Contributions

The contributions contained herein lie in several disparate fields of research. All pertain to posing, modeling, and solving hybrid-routing scheduling problems with human-in-the-loop guidance. As such, contributions are divided into the following three categories: *representation*, *modeling and learning*, and *human-machine interaction*.

6.1.1 Representation - The Generalized Planning Problem

This thesis introduces a new framework for posing assignment problems such as canonical routing and scheduling problems, referred to as the Generalized Planning Problem framework. A GPP is completely defined by the problem attributes: agents, tasks, and costs. The representation is fundamentally cost based in that every agent-task pairing (allocation) is completely defined by a cost. The job of the task designer is to come up with meaningful costs that account for spatial problem elements (eg. distance), temporal problem elements (eg. processing time), and other problem elements or constraints that may affect the overall fitness of a particular agent accomplishing a particular task. One of the chief representation benefits of the GPP framework is the ability to include multiple stochastic problem elements as well as hard and soft constraints directly into the state.

The GPP is introduced in Chapter 2. Chapter 4 poses a VRP as a GPP, and Chapter 5 poses a complex pickup and delivery problem with multiple stochastic elements as a GPP. The breadth of examples in this thesis shows is a testament to the representational power and generality of the GPP framework. The GPP framework does not assume a particular solution method, although it has been designed with learning-based solution methods in mind.

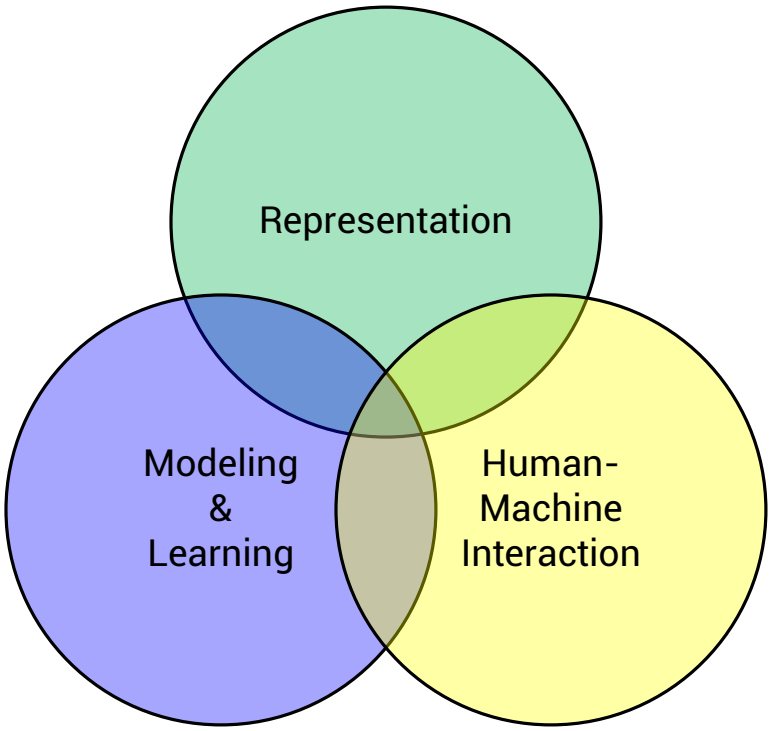


Figure 6.1: Summary of thesis contributions fields.

6.1.2 Modeling and Learning

This thesis introduces a learning-based approach to solving Generalized Planning Problems. Several advantages of learning make it a particularly compelling choice for solving GPPs.

- A standard learning problem definition involves a state and action set definition, transition model, reward function, and choice of algorithm with associated parameters. Constraints are simply encoded in the reward function or transition dynamics.
- Many learning algorithms incorporate a notion of randomness through a transition model or simulator.
- A simulator may be used in place of an explicit probabilistic transition model in problems for which a transition model would be difficult or impossible to define.
- Learning is an experience-based solution method. By simulating state dynamics over many episodes, in conjunction with a reward function that values certain state-action configuration over others, an optimal policy can be learned for all states.
- Many learning algorithms include intuitive tuning parameters that control both the speed of the learning process and the quality of the end policy.

This thesis presents an MDP-based model for the the GPP. A significant limitation of MDPs however, is that actions occur over exactly one time step. Many real problems, however, are more realistically modeled with temporally extended actions that occur over multiple time steps. As such, a significant innovation presented in this thesis is a method of temporal abstraction for the GPP via the SMDP model. SMDPs extend the standard MDP model with the inclusion of macro actions, referred to as options, which occur over multiple time steps. The incorporation of options allows for a more expressive representation, a drastically reduced state-action space, and more effective learning as a result. To the knowledge of the author, this work is the first example of modeling a hybrid routing-scheduling decision problem as a SMDP.

6.1.3 Human-Machine Interaction

A core insight presented in this thesis is that in many real world examples, humans interact with autonomous systems. However, hybrid human-autonomous decision making is poorly understood, and in many cases the human acts either as the primary operator or as a passive observer. Ideally though, systems would be "adjustable autonomous", intelligently combining human advice with autonomous decision making, producing a hybrid solution.

This thesis adds to the body of knowledge in variably autonomous human-machine systems by providing a intuitive and structured conduit for including human feedback into learned GPP solutions. The underlying theory of reward shaping is leveraged to combine a human reward function with an implicit a-priori reward function. The human reward function may be checked against conditions to determine its admissibility, guarantying policy optimality even under human-guided solutions.

Human feedback is included as a weighted sum of human reward and a-priori reward. This approach effectively implements adjustable autonomy while providing conditions for guaranteeing policy optimality. The resultant policy is allowed to vary between fully autonomous and fully manual (human control) and everywhere in between. This variation can occur both at learn-time and at online at run-time.

6.1.4 Stochastic Pickup and Delivery

Chapter 5 presents an example problem to showcase the contributions of this thesis. The hybrid routing-scheduling problem is most closely related to a VRPPD, in which vehicles must both pickup and deliver passengers or goods. A common real-world VRPPD is taxi routing. The example problem includes several stochastic elements: both the destination of each pickup task and the vehicle dynamics are treated as random quantities. Additionally, task destinations are revealed upon pickup, making the problem switched-stochastic: random variables are realized and replaced with the actual quantities at random times. The example also includes a soft temporal constraint which increasingly penalizes loitering as each episode progresses. Traditional solution methods are ill-equipped to deal with such a large-scale decision problem with multiple random elements and soft constraints.

This taxi routing example problem is modeled as a GPP. Reward shaping is leveraged to include human guidance and expedite the learning process while improving policy quality. Although no baseline optimal solution exists for this kind of problem, an hand-coded near-optimal greedy solution provides an indication of learning performance and policy performance.

6.2 Future Research Prospectus

6.2.1 Implementation

Several avenues of future research are as follows.

6.2.2 Unified Assignment Problem Representation

The space of assignment problems is vast, including applications from server scheduling to air traffic control. The utility in unifying these disparate domains under one representational framework is threefold: common heuristics, solution generalization, and software reuse.

Common heuristics: Defining a common representational framework for the space of assignment problems implicitly sets a common level of abstraction. As a direct result, human input in the form of heuristics has the opportunity to generalize across several problems that are all defined in the same way. As useful heuristics are typically hard to find and problem specific, defining problems broadly allows for heuristics that may also be applied broadly.

Solution generalization: A direct result of a generalized state representation is a correspondingly general solution. By considering high-level features of the state rather than low-level state dynamics, the eventual solution may be perturbed along low-level dimensions without changes to the higher level policy.

Software reuse: An increasingly popular software design paradigm is Component Based Software Design (CBSD), which essentially imposes modularity in the form of individually executable components for the purpose of portability and reuse. A practical benefit of a common representational framework is that the software tools written to solve for the optimal policy of a vehicle routing problem may also be used to solve a different assignment problem with minimal changes to the underlying code.

The GPP framework lays the foundation for future representational breadth. Currently, knowledge gained in one GPP is not transferable to another. For example, sub-policies learned in a VRP-like problem can not be leveraged to solve a similar VRPPD. Essentially, no precise notion of problem and policy similarity exists. Some of the difficulty in solution generalization lies in implementation details and reusable software. Other challenges lie in

the theory of policy generalization—how a policy learned in one domain applies in another. Future research in unified representation and policy generalization will allow for reuse of learned policies in similar problems.

6.2.3 Semi-Markov Learning

Much of the modeling and learning algorithms used in thesis are well-studied by the machine learning community. The SMDP model is a relatively recent innovation, and several future directions of learning research are yet to be explored for the multi-agent routing problems presented in this thesis.

Action Interruption In any problem where the environment is random, and agent needs to ability the current course of action for more valuable action. Sutton et al. has demonstrated the utility of asynchronous action interruption for a stochastic single-vehicle routing problem. It would surely be well-applied to the multi-agent case.

Reward Shaping Example problems in Chapter 4 have shown how reward shaping may be combined with the SMDP model. However, several complications exist, a particularly for interrupted actions and other SMDP variants. Considering the benefits of reward shaping as a method of incorporating human feedback, research into reward shaping for SMDPs would be a necessary area of research for extending the results of the taxi routing example problem in Chapter 5 to the SMDP domain

Function Approximation Function approximation is a necessary technique for solving large decision problems, and the combination of SMDP learning and function approximation techniques is the subject of current learning research. The subsequent section address function approximation in general for the GPP.

6.2.4 Function Approximation

Vehicle routing problems are a class of NP-hard decision problems with few solution methods (optimization or search-based) that both scale and generalize well. Especially in the multi-agent setting, problems become intractably large as time and space constraints are added on top of the standard formulation. The problem designer is thus left with approximation techniques.

Several function approximation techniques exist for solving large reinforcement learning problems. Promising methods include radial basis functions [66], Kanerva coding [64], and Gaussian processes (GPs) [16]. Kanerva coding has worked well in applications where a large amount of the training data is irrelevant or erroneous . GPs have show particular promise recently in dealing with high dimension functions where little data is available.

Indeed, multi-agent assignment problems can be intractably large, with no hope of solution without approximation. This reality motivates research into function approximation,

since many of the relevant assignment problems—air traffic control, power grid scheduling, FedEx, and similar—are fundamentally large scale.

6.2.5 Human Heuristics

Humans are particularly adept at developing fast solutions to complex decision problems. Particularly true when state spaces are vast (consider all the states involves in just walking down the street), our brains readily extract pertinent features for the task at hand (feature identification), determine the state (classification), and make reliable decisions (control). Specifically, humans readily leverage problem context and abstraction to render difficult decision problems (chess, strategy games, image classification) tractable. The following questions are related to future work in human decision modeling:

- To what extent do humans use heuristics to solve spatiotemporal problems?
- Are the heuristics based on a small or large set of features, what are the features, and can they be learned?
- Are the features / heuristics transferable from one problem domain to another (hyper-features)?
- How close to optimal can an average human get in solving specific decision problems (ex: vehicle routing)?
- How does human performance degrade as a function of problem size?
- What data structures are useful for capturing human heuristics?
- Can human heuristics be learned effectively in a reinforcement learning scenario?
- What class of graphical models, if any, can be leveraged to model human decision making?

6.3 Summary

Both contributions and future work of this thesis address solving large-scale decision problems for which no other adequate model or solution method exists. Much of the future work pertains to improving solution quality by decreasing problem size and increasing the expressive power of the GPP framework. This thesis makes significant strides towards solving meaningful real-world decision problems by efficient representation and leveraging human guidance.

Appendix A

Example Problem Solutions

A.1 Example 1: NYC Taxi Routing

A.1.1 Setup

Let:

$$p_T = p(v < v_T|t) \qquad \text{Probability of traffic given time of day}$$

$$p_M = p(v \geq v_T|t) \qquad \text{Probability of moving given time of day}$$

$$c_{ij}|v_T = \frac{d(i,j)r_t}{v_T} \qquad \text{Cost in traffic}$$

$$c_{ij}|v = d(i,j)r_d \qquad \text{Cost while moving}$$

A.1.2 Before Pickup ($t < t_p$)

The expected cost before pickup at **P** can be expressed as follows:

$$\begin{aligned} f^{ij} &= \mathbf{E}[c_{iP} + c_{PD}] \\ &= \mathbf{E}[c_{iP}] + \mathbf{E}[c_{PD}] && c_{iP} \perp c_{Pj} = c_{PD} \\ &= \mathbf{E}[c_{iP}|v_T]\ p_T + \mathbf{E}[c_{iP}|v]\ p_M \\ &+ \mathbf{E}[c_{PD}|v_T]\ p_T + \mathbf{E}[c_{PD}|v]\ p_M && \text{Total expectation} \end{aligned}$$

The velocity v is sampled from $p(v|T)$ and is no longer random. Velocity is updated (resampled) and an interval H. During each period between sampling the velocity is assumed to be constant.

$$f^{ij} = c_{iP} + \mathbf{E}[c_{PD}|v_T]\ p_T + \mathbf{E}[c_{PD}|v]p_M$$

$$= c_{iP} + \mathbf{E}[\frac{d(P,D)}{v_T}]\ r_t\ p_T + \mathbf{E}[d(P,D)]\ r_d\ p_M$$

A.1.3 After Pickup ($t \geq t_p$)

After pickup, the random distance variable $d(P,D)$ is realized, and velocity can be sampled from its distribution. The cost is therefore expressed as follows:

$$f^{ij} = \frac{d(P,D) \times r_t}{v_T} + d(P,D) \times r_d$$

A.2 Learned Taxi Routing

A.2.1 Simulator detail

The taxi routing problem described in Example 1 is essentially a stochastic VRPPD. It combines multiple stochastic elements with several dynamic constraints, that are either imposed by selectively rewarding certain states, or not allowing those states to occur in the first place, via the simulator. The following lists several problem dynamics, constraints, and assumptions all off which are imposed by the low-level dynamic simulator.

C1. All pickup locations are know by all agents.

C2. Agents can only be allocated to one task at a time.

C3. An agent proceeds directly to the destination after pickup.

C4. A task can only be completed by the agent that makes the pickup.

C5. A destination D_j is revealed to all agents upon pickup at P_j.

C6. A task done by one agent is considered done over all.

C7. Once a task is done it stays done until episode completion.

A.2.2 Human reward shaping optimality

Bibliography

[1] Pieter Abbeel and Andrew Y. Ng. "Apprenticeship Learning via Inverse Reinforcement Learning". In: *In Proceedings of the Twenty-first International Conference on Machine Learning*. ACM Press, 2004.

[2] Sherief Abdallah and Victor Lesser. "Modeling task allocation using a decision theoretic model". In: *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*. The Netherlands: ACM, 2005, pp. 719–726. ISBN: 1-59593-093-0.

[3] Karol Adamiecki. In: *Przegl?d Techniczny* ().

[4] Andrew G. Barto and Sridhar Mahadevan. "Recent Advances in Hierarchical Reinforcement Learning". In: 13 (2003).

[5] J. Christopher Beck, Patrick Prosser, and Evgeny Selensky. "Vehicle Routing and Job Shop Scheduling: What's the difference?" In: *Proc. of the 13th International Conference on Automated Planning and Scheduling*. 2003, pp. 267–276.

[6] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[7] Justin A. Boyan and Michael L. Littman. "Exact Solutions to Time-Dependent MDPs". In: *in Advances in Neural Information Processing Systems*. MIT Press, 2000, pp. 1026–1032.

[8] Alex Brooks et al. "Parametric POMDPs for planning in continuous state spaces." In: *Robotics and Autonomous Systems* 54.11 (2006), pp. 887–897.

[9] Hung Hai Bui, Svetha Venkatesh, and Geoff A. W. West. "On the Recognition of Abstract Markov Policies." In: *AAAI/IAAI*. AAAI Press / The MIT Press, 2000.

[10] Rainer E. Burkard, Mauro Dell'Amico, and Silvano Martello. *Assignment Problems*. SIAM, 2009, pp. I–XX, 1–382. ISBN: 978-0-89871-663-4.

[11] Rainer E. Burkard, Mauro Dell'Amico, and Silvano Martello. *Assignment Problems*. SIAM, 2009.

[12] G Clarke and J W Wright. "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points". In: *Operations Research* 12.4 (1964), pp. 568–581.

[13] Richard Conway, William L. Maxwell, and Louis W. Miller. *Theory of scheduling / [by] Richard W. Conway, William L. Maxwell [and] Louis W. Miller*. Addison-Wesley Pub. Co, Reading, Mass., 1967.

[14] Jacob W. Crandall and M. L. Cummings. "Developing performance metrics for the supervisory control of multiple robots". In: *HRI '07: Proceedings of the ACM/IEEE international conference on Human-robot interaction*. Arlington, Virginia, USA: ACM, 2007, pp. 33–40. ISBN: 978-1-59593-617-2.

[15] G. B. Dantzig and J. H. Ramser. "The Truck Dispatching Problem". In: *Management Science* 6.1 (1959), pp. 80–91.

[16] Marc Peter Deisenroth, Carl Edward Rasmussen, and Jan Peters. "Gaussian process dynamic programming". In: *Neurocomput.* 72.7-9 (Mar. 2009), pp. 1508–1524. ISSN: 0925-2312.

[17] Thomas G. Dietterich. "Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition". In: *Journal of Artificial Intelligence Research* 13 (2000), pp. 227–303.

[18] Dmitri A. Dolgov and Edmund H. Durfee. "Optimal Resource Allocation and Policy Formulation in Loosely-Coupled Markov Decision Processes". In: *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS)*. 2004, pp. 315–324.

[19] Moshe Dror. "Modeling vehicle routing with uncertain demands as a stochastic program: Properties of the corresponding solution". In: *European Journal of Operational Research* 64.3 (1993), pp. 432 –441.

[20] Samuel Eilon et al. *Distribution management: mathematical modelling and practical analysis*. Griffin, 1971.

[21] Merrill M. Flood. "The Traveling-Salesman Problem". In: *Operations Research* 4.1 (1956), pp. 61–75.

[22] Michael R. Garey and David S. Johnson. *Computers and Intractability A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979. ISBN: 0716710455.

[23] Jared M Garvey et al. "An Autonomous Unmanned Aerial Vehicle System for Sensing and Tracking". In: *AIAA InfotechAerospace*. 2010, pp. 1–8.

[24] Michel Gendreau, Gilbert Laporte, and Ren Sguin. "Stochastic vehicle routing". In: *European Journal of Operational Research* 88.1 (1996), pp. 3 –12.

[25] Brian P. Gerkey and Maja J. Matari? "A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems". In: *The International Journal of Robotics Research* 23.9 (2004), pp. 939–954.

[26] B. L. Golden and A. A. Assad. *Vehicle Routing: Methods and Studies*. North-Holland, 1988.

[27] Ben Grocholsky. "Information-Theoretic Control of Multiple Sensor Platforms". Doctoral Thesis. The University of Sydney, 2002.

[28] Carlos Guestrin, Daphne Koller, and Ronald Parr. "Multiagent Planning with Factored MDPs". In: *In NIPS-14*. The MIT Press, 2001, pp. 1523–1530.

[29] Carlos Guestrin et al. "Efficient Solution Algorithms for Factored MDPs". In: *Journal of Artificial Intelligence Research (JAIR)* 19 (2003), pp. 399–468.

[30] T. Kaupp. *Human-Robot Collaboration – A Probabilistic Approach*. VDM Publishing, 2009.

[31] Tobias Kaupp and Alexei Makarenko. "Measuring human-robot team effectiveness to determine an appropriate autonomy level". In: *ICRA*. 2008, pp. 2146–2151.

[32] Tobias Kaupp et al. "Shared environment representation for a human-robot team performing information fusion". In: *J. Field Robot.* 24.11-12 (2007), pp. 911–942.

[33] H. W. Kuhn. "The Hungarian method for the assignment problem". In: *Naval Research Logistic Quarterly* 2 (1955), pp. 83–97.

[34] G Laporte and Y Nobret. "Extact Algorithms for the Vehicle Routing Problem". In: *Surveys in Combinatorial Optimization* (1987), pp. 147–184.

[35] Gilbert Laporte. "The vehicle routing problem: An overview of exact and approximate algorithms". In: *European Journal of Operational Research* 59.3 (1992), pp. 345 –358.

[36] Allan Larsen. "The Dynamic Vehicle Routing Problem". Doctoral Thesis. Technical University of Denmark, 2001.

[37] Allan Larsen, Oli B G Madsen, and Marius M Solomon. "Classification Of Dynamic Vehicle Routing Systems". In: *Dynamic Fleet Management*. Vol. 38. Springer US, 2007, pp. 19–40.

[38] Eugene Lawler. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. New York: Wiley, 1985.

[39] Rajbala Makar, Sridhar Mahadevan, and Mohammad Ghavamzadeh. "Hierarchical multi-agent reinforcement learning". In: *Proceedings of the fifth international conference on Autonomous agents*. AGENTS '01. 2001, pp. 246–253.

[40] Alexi A. Makarenko. "A Decentralized Architecture for Active Sensor Networks". Doctoral Thesis. The University of Sydney, 2004.

[41] Victoria Manfredi and Sridhar Mahadevan. *Hierarchical Reinforcement Learning Using Graphical Models*. 2005.

[42] George M. Mathews, Hugh Durrant-Whyte, and Mikhail Prokopenko. "Decentralised decision making in heterogeneous teams using anonymous optimisation". In: *Robot. Auton. Syst.* 57.3 (2009), pp. 310–320. ISSN: 0921-8890.

[43] George Matthews. "Asynchronous Decision Making for Decentralised Autonomous Systems". Doctoral Thesis. The University of Sydney, 2008.

[44] Martin Midtgaard et al. "Time-Based Reward Shaping in Real-Time Strategy Games". In: *Agents and Data Mining Interaction*. Vol. 5980. Lecture Notes in Computer Science. 2010, pp. 115–125.

[45] Gaspard Monge. "Mémoire sur la théorie des déblais et de remblais". In: *Histoire de l'Académie Royale des Sciences de Paris, avec les Mémoires de Mathématique et de Physique pour la méme Année*. 1781, pp. 666–704.

[46] Kevin P. Murphy. "Dynamic bayesian networks: Representation, inference and learning". Doctoral Thesis. University of California Berkeley, 2002.

[47] Kevin P. Murphy. *Hidden semi-markov models (hsmms*. Tech. rep. 2002.

[48] Kevin P. Murphy and Mark A. Paskin. "Linear Time Inference in Hierarchical HMMs". In: *In Proceedings of Neural Information Processing Systems*. 2001.

[49] Muhammad Nawaz, E Emory Enscore Jr, and Inyong Ham. "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem". In: *Omega* 11.1 (1983), pp. 91 –95.

[50] Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. "Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping". In: *ICML '99: Proceedings of the Sixteenth International Conference on Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 278–287.

[51] Reza Olfati-saber, J. Alex Fax, and Richard M. Murray. "Consensus and cooperation in networked multi-agent systems". In: *Proceedings of the IEEE*. 2007, p. 2007.

[52] Ronald Parr and Stuart Russell. "Reinforcement Learning with Hierarchies of Machines". In: *Advances in Neural Information Processing Systems 10*. MIT Press, 1997, pp. 1043–1049.

[53] Ronald Edward Parr. "Hierarchical Control and Learning for Markov Decision Processes". Doctoral Thesis. University of California Berkeley, 1998.

[54] Emmanuel Rachelson, Chemin de Borde Rouge, and Patrick Fabiani. "Extending the Bellman equation for MDPs to continuous actions and continuous time in the discounted case". In: (2008).

[55] Emmanuel Rachelson, Patrick Fabiani, and Frdrick Garcia. "TiMDPpoly: An Improved Method for Solving Time-Dependent MDPs". In: *International Conference on Tools with Artificial Intelligence*. 2009, pp. 796–799.

[56] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 2nd edition. Prentice-Hall, Englewood Cliffs, NJ, 2003.

[57] A. Ryan et al. "A distributed control system for unmanned aerial vehicles executing collaborative sensing missions using multi-step plans". In: *American Controls Conference*. 2007.

[58] A. Ryan et al. "An Overview of Emerging Results in Cooperative UAV Control". In: *Proceedings of the IEEE Conference on Decision and Control*. Bahamas, 2004.

[59] A. Ryan et al. "Collaborative flight demonstration with three unmanned aerial vehicles". In: *Proceedings of the AIAA, Guidance, Navigation and Control Conference*. Keystone, CO, 2006.

[60] A. Ryan et al. "Decentralized Control of Unmanned Aerial Vehicle Collaborative Sensing Missions". In: *American Control Conference, 2007. ACC '07*. 2007, pp. 4672 – 4677.

[61] David Silver, Richard S. Sutton, and Martin Müller. "Temporal-difference search in computer Go". In: *Machine Learning* 87.2 (2012), pp. 183–219.

[62] T. Sugawara and V.R. Lesser. "Learning to Improve Coordinated Actions in Cooperative Distributed Problem-Solving Environments". In: *Machine Learning* 33.2-3 (1998), pp. 129–153.

[63] Richard Sutton, Doina Precup, and Satinder Singh. "Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning". In: *Artificial Intelligence* 112 (1999), pp. 181–211.

[64] Richard S. Sutton. "Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding". In: *Advances in Neural Information Processing Systems 8*. MIT Press, 1996, pp. 1038–1044.

[65] Richard S. Sutton. "Learning to Predict by the Method of Temporal Differences". In: vol. 3. 1999, pp. 9–44.

[66] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press, 1998.

[67] Richard S. Sutton et al. "Improved Switching among Temporally Abstract Actions". In: *Advances in Neural Information Processing Systems*. 1999, pp. 1066–1072.

[68] Georgios Theocharous. "Hierarchical learning and planning in partially observable markov decision processes". PhD thesis. 2002.

[69] Georgios Theocharous, Khashayar Rohanimanesh, and Sridhar Mahadevan. *Learning Hierarchical Partially Observable Markov Decision Process Models for Robot Navigation*. 2001.

[70] Andrea Lockerd Thomaz and Cynthia Breazeal. "Socially Guided Machine Learning: Designing an Algorithm to Learn from Real-Time Human Interaction". In: *NIPS 2005 workshop on Robot Learning in Unstructured Environments*. 2005.

[71] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. ISBN: 0262201623.

[72] P. Toth and D. Vigo. *The vehicle routing problem*. Vol. 9. Society for Industrial Mathematics, 2002.

[73] C. Watkins. "Learning from Delayed Rewards". PhD thesis. University of Cambridge, 1989.