Specifications from Demonstrations: Learning, Teaching, and Control

Marcell Vazquez-Chanlatte

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2022-107
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2022/EECS-2022-107.html

May 13, 2022

Acknowledgement

This work was partially supported by NSF grants 1545126 (VeHICaL), 1646208 and 1837132, by the DARPA contracts FA8750–16–C–0043 (BRASS), FA8750-18-C-0101 (Assured Autonomy) and FA8750-20-C-0156 (SDCPS), by Berkeley Deep Drive, by Toyota under the iCyPhy center, and the US ARL Cooperative Agreement W911NF-17-2-0196.

Specifications from Demonstrations: Learning, Teaching, and Control

by

Marcell Jose Vazquez-Chanlatte

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Spring 2022

Specifications from Demonstrations: Learning, Teaching, and Control

Abstract

Specifications from Demonstrations: Learning, Teaching, and Control

by

Marcell Jose Vazquez-Chanlatte

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Sanjit A. Seshia, Chair

This dissertation considers the problem of learning and teaching Boolean task specifications, such as automata, using demonstrations. The resulting framework bridges grammatical inference and maximum-entropy inverse reinforcement learning with applications in human-robot interaction, formal synthesis, and multi-task reinforcement learning.

In the context of inverse reinforcement learning, Boolean task specifications are a class of sparse memory augmented rewards with explicit support for temporal and Boolean composition. These properties make task specifications immune to certain classes of reward hacking bugs that emerge from ad-hoc composition or perturbations to the dynamics. Unfortunately, the discrete nature of task specifications combined with an a-priori ignorance of what historical features are needed to encode the demonstrated task make existing approaches to learning rewards from demonstrations inapplicable.

In the context of specification mining and grammatical inference, demonstrations provide an ergonomic and sample efficient means to communicate formal languages and specifications. For example, this dissertation enables a user to partially specify the desired behavior of a system as example demonstrations and then find an automata or program that explains the user's behavior. Conversely, by synthesizing pedagogic demonstrations, this dissertation enables communicating the nuances of a specification that may be hard to intuit from a formal description.

In either case, this thesis contributes a collection of algorithms and theoretical machinery for systematically mitigating combinatorial explosions inherent in (1) finding specifications that explain an agent's behavior (2) finding pedagogic demonstrations that help humans infer the specification and (3) robustly predicting the behavior of an agent adhering to a specification.

Dedicated with respect and appreciation to all my teachers past, present, and future: gods, men, and demons; being, animate and inanimate, living and dead, alive and dying...

Edward Espe Brown (The Tassajara Bread Book, 1970)

...also my parents, siblings, and family. I would particularly like to dedicate this thesis to my loving and supportive wife, Marissa Ramirez de Chanlatte, without whom I would have written "who" rather than "whom".

Contents

Acknowledgments

When seeking council on writing a dissertation, I was told to write my acknowledgements first, as by the end I would be too tired to do it justice. I must confess that I failed to heed this advice. As such, if you are absent, but feel you made an impact on this dissertation, know that you were (probably) not left out by malice, but by sleep deprivation.

This dissertation was made possible by the support and encouragement of my collaborators, mentors, friends and family. In particular, I would like to thank my advisor Sanjit A. Seshia and my committee members Shankar Sastry, Anca Dragan, and Steven Piantadosi. I can confidently say that this dissertation would not have taken its present form without each of their influences. In addition, I would like to acknowledge the various collaborators and colleagues who directly shaped my research career. For my early work on specification mining, I acknowledge Jyo Deshmukh, Shromona Ghosh, Jim Kapinski, Oded Maler, Alex Donze, Xiaoqing Jin, Vasu Raman and Alberto Sangiovanni Vincentelli. For my work on learning and teaching via demonstrations, I thank Dorsa Sadigh, Ashish Tiwari, Gil Lederman, Mark Ho, Ameesh Shah, Beyazit Yalcinkaya, Niklas Lauffer, Vint Lee, Ben Caulfield, Dexter Scobee, Kaylene Stocking, David McPherson, and Susmit Jha. Similarly, the control part of my thesis - ranging from control improvisation to modeling dynamics a probabilistic circuits, wouldn't have been possible without wonderful discussions with Markus Rabe, Daniel Fremont, Eric Kim, Sebastian Junges, Kuldeep Meel, and Dogan Ulus.

I would also like to thank my lab mates not listed above, e.g., Edward Kim, Hazem Torfah, Elizabeth Polgreen, Kevin Cheang, Ankush Desai, Rohit Sinha, Yash Pant, Tomasso Dreossi, Yasser Shoukry, Federico Rocha, Pramod Subramanyan, Victoria Tuck, and Yatin Manerkar. I would also like to thank my various flatmates: Greg Khan, Michael Kellman, Brian Nemsick, Brian Kilberg, Grant Ho, and Sally Lape, who helped me weather the dystopian reality of rental prices in Berkeley. And, I would be remiss if I did not thank my family. Particularly my wife, daughter, mother, and siblings. Thank you. I know I can be a handful.

Finally, this work was partially supported by NSF grants 1545126 (VeHICaL), 1646208 and 1837132, by the DARPA contracts FA8750–16–C–0043 (BRASS), FA8750-18-C-0101 (Assured Autonomy) and FA8750-20-C-0156 (SDCPS), by Berkeley Deep Drive, by Toyota under the iCyPhy center, and the US ARL Cooperative Agreement W911NF-17-2-0196.

Chapter 1

Introduction

Since I am neither a neurologist nor a psychiatrist, but a mathematician, the work that follows requires some explanation and justification.

John von Neumann (The Computer and the Brain, 1957)

Consider a car turning at a busy intersection. Ideally, the car should be able to infer the intent of other agents, such as cars and pedestrians, from past observations, and infer what assumptions and inferences the other agents are making about the world. For example, is the pedestrian assuming two vehicles will not enter the intersection at the same time? Similarly, we would like the car to communicate its own objective, assumptions, and constraints to other agents, e.g. vehicles and pedestrians.

Such scenarios involving cyber-physical systems (CPS) are becoming increasingly ubiquitous, with applications ranging from driving to manufacturing [93]. This ubiquity brings an appetite for "auditable" systems whose high-level intent and assumptions can be easily communicated to other agents, e.g., human collaborators or regulators.

Taking a step back three questions naturally arise:

1. How should intent and assumptions be represented?
2. By what means should the representations be communicated, e.g., natural language, visual cues, etc?
3. How should the answers to (1) and (2) change when talking to humans versus another CPS?

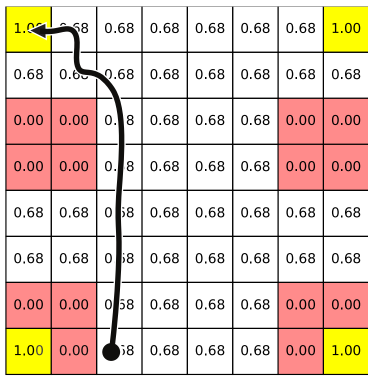
For the sake of brevity, let us refer to the intended behavior of an agent (along with any implicit assumptions about the world) as a task.

Thesis Preview

In this dissertation, we will setup the problems of learning and teaching tasks, represented as formal specifications, using demonstrations. This will require bringing in theory from many disparate areas ranging from grammatical inference [45] to maximum entropy inverse reinforcement learning [162]. We shall see that the core technical difficulties in realizing this vision are the inherent combinatorics of task specification representations, e.g. automata, logic, and programs. In particular, we shall address the combinatorics in (i) predicting the behavior of agents trying to perform a task; (ii) searching for task specifications that explain the demonstrations; and (iii) optimizing trajectories to be more pedagogic. These problems, referred to as Control, Learning, and Teaching, respectively, naturally build on each other, and this dependence will be reflected in the exposition. For example, we shall use the predictions from (i) to define the explanatory power of a given task specifications in (ii). Similarly, we shall use the task inferences of (ii) to measure how effective a demonstration is at teaching a task. In the sequel, we provide background, the motivations, and the contributions of this thesis. We will then outline the structure of the rest of the dissertation.

Representing tasks

Examining the literature, one finds a variety of possible ways to represent tasks, e.g. ([82, 77, 156, 78, 157, 78]). For example, the problem of inferring the intent of agents optimizing some real valued objective in a dynamical system has a rich and well developed literature - with roots dating back to the early work on Inverse Optimal Control [82]. Within this paradigm, the intent and assumptions of the agent are intimately tied into the way that optimizing the objective interacts with the world model. Note however, that when reasoning about such systems, either for collaboration or for regulation, this tight coupling between objective and dynamics proves problematic. Case in point: the particular task encoded by the objective may dramatically change due to simple changes in the environment.


(a)

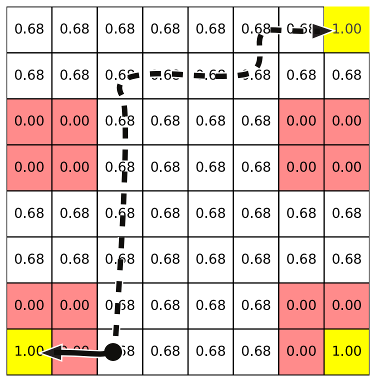

(b)

Figure 1.1: Illustration of a bug in the learnt quantitative Markovian reward resulting from slight changes in the environment.

Example 1.0.1. *Consider the gridworld shown in Fig 1.1a where the agent can move up, down, left, or right and the episode ends at any point with probability* 0.69*. The agent model* *has the robot optimize the expected sum of state rewards, where the rewards at each state are shown as numbers in the tiles and the encoded task represents visiting yellow (🟨) and avoiding red (🟥). For a more concrete example, one might consider this to represent a robot needing to recharge while avoiding lava. The key feature is that removing a recharging station from a workspace (Fig 1.1b) results in the agent entering the lava to take a shortcut to another charging station.*

Additionally, the nature of assumptions or temporal relationships are left implicit, making explicit communication with other agents (using potentially different world models) unnecessarily brittle. At the risk of over emphasizing, just because two agents share a common high level description of behaviors, does not mean they agree on the relative ordering on behaviors! For example, one agent might model time as continuous, another as discrete, and still another as discrete but at a different resolution. Similarly, one agent might model the dynamics as deterministic whereas another uses probabilistic dynamics. These modeling details effectively change the *relative weights* of behaviors, e.g., a path might seem slightly shorter in one model than another. Further, there is no general way to associate a set of "acceptable" behaviors with the relative ordering of the behaviors induced by a Markovian (state-based) reward [2], i.e., a behavior is acceptable so long as it performs at least as good as a given set of behaviors in the given world model.

Representing tasks as specifications

An alternative approach championed throughout this thesis is to make the set of desirable behaviors explicit.[1] This set, realized by means of a formal specification, provides a dynamics-independent classification of behaviors as good or bad. Importantly, the relative preferences induced by the dynamics now only encode how "risky" a particular behavior is. For example, the possibility of slipping makes the (good) path that gets close to the ledge worse than the (good) path that doesn't.[2] Thus, even when two world models disagree on the relative ordering between behaviors, the set of acceptable behaviors remains stable. This change eliminates entire classes of so called reward-hacking bugs [94]. Focusing on Boolean specifications has additional benefits: (1) The ability to naturally express tasks with *temporal dependencies*; (2) the ability to take advantage of the *compositionality* present in many problems; and (3) the use of *formal methods* for planning and verification [130].

Unfortunately, formal specifications are by no means a panacea, and expressing intent using formal specifications is non-trivial even for domain experts. Furthermore, while the semantics of a formal specification may be invariant to small changes in the world model, specifying the wrong behavior is still possible - if not inevitable. Adding insult to injury, many systems lack specifications all together, making any arguments about formal verification and planning moot. Often, the only hint at intended behavior hides in the way an agent or

[1]Again, subject to a common vocabulary for describing observations

[2]Formally, the relative ordering between behaviors is lexicographically ordered. First by a dynamics independent classification of good and bad, and then by a dynamics dependent preference.

the system interacts with the world. These observations lay the foundation for the three central research problems explored in this dissertation, referred to as Learning, Teaching and Control.

Modeling Agents

First and foremost, the Control (or Prediction) problem asks: How can one robustly and efficiently predict the behavior of an agent performing a task in a given workspace? In particular, we wish to model purposefulness informally defined as follows:

Definition 1 (Purposefulness [142])**.** There must be, on the part of the behaving entity, i.e., the agent: (a) a desire, whether actually felt or not, for some object, event, or state of affairs as yet future; (b) the belief, whether tacit or explicit, that a given behavioral sequence will be efficacious as a means to the realization of that object, event, or state of affairs; and (c) the behavior pattern in question. Less precisely, this means that to say of a given behavior pattern that it is purposeful, is to say that the entity exhibiting that behavior desires some goal and is behaving in a manner it believes appropriate to the attainment of it.

Now, as the chapter quote suggests, this dissertation was not written from the perspective of neither a neurologist nor psychiatrist and make no claims of deep insight into the nature of complex agents such as humans. Fortunately, unlike the natural sciences where observed differences between model and theory might be dubbed a catastrophe [49], our goal will only be to make predictions in a way that minimizes how hard it is to describe the actions of a competent agent. Technically, this is achieved by leveraging the established theory of maximum causal entropy for modeling purposeful behavior in sequential decision processes [162]. Operationally, this theory prescribes forecasting the actions of the agent using an entropy-regularized policy that trades off performance for randomness (entropy). The key detail is that this theory provides predictions of a purposeful agent's behavior that minimize the worst case log-loss prediction accuracy, i.e., the number of bits needed to describe the behavior. In this way, we do not care if the agent is human or if the agent is using a slightly different world model for planning. So long as the dynamics model is expressive enough to capture common behavior, the enforced guarantees will (hopefully) paper over modeling discrepancies (an observation formalized in the robust entropy regularized control literature [52]).

Communicating tasks

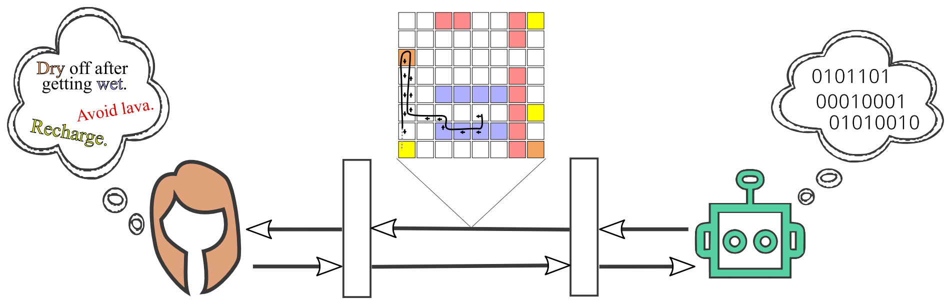


Figure 1.2: Illustration of using demonstrations to communicate a task specification. The human (left) has a natural language description of the task in mind and the robot (right) maintains a binary encoding of a task specification.

The learning and teaching problems successively build on the control problem to communicate formal task specifications using demonstrations (as illustrated in Fig 1.2). The learning problem asks the question: Given a collection of expert demonstrations, which formal task specifications explain the behavior of the expert? Importantly, because of the ambiguous nature of the problem statement, we seek a distribution of specifications that plausibly explain the agent's behavior.

Next, because formal task specifications can be difficult to understand, the teaching problem asks the question: Can one generate a small number of pedagogic demonstrations[3] which help a user understand a task? To ground the notion of generating pedagogic examples, we will lean heavily on the literature on pragmatics [62], legibility [48], and showing vs doing [73]. The core idea will be for the teacher to simulate the learner in order to provide demonstrations that make the true task easier to infer.

1.1 Contributions

This thesis contributes a collection of algorithms and theoretical machinery for systematically mitigating combinatorial explosions inherent in: (1) finding task specifications that explain an agent's behavior; (2) finding pedagogic demonstrations that help humans infer the specification; (3) robustly predicting the behavior of an agent adhering to a specification. Specific contributions are enumerated below:

Maximum Causal Entropy Specification Learning from Demonstrations ([153], Ch 4): We propose using the principle of maximum causal entropy to formulate the problem of learning task specifications from expert demonstrations operating in Markov

[3]Potentially supplemented with formal or natural language descriptions.

Decision Processes. In the context of specification mining [97] and grammatical inference [45], demonstrations provide an ergonomic and sample-efficient means to communicate formal languages and specifications. The key contribution of this work was a formulation compatible with Bayesian pipeline that supported unlabeled and even incomplete demonstrations. Further, in the context of inverse reinforcement learning [111], Boolean task specifications are a class of sparse memory augmented rewards with explicit support for temporal and Boolean composition. This composition enables incremental learning, and the Boolean episode level semantics eliminate entire classes of so-called "reward hacking bugs".

Maximum Entropy Specification Learning from Demonstrations ([149], Ch 4): We study the special case when maximum causal entropy and maximum entropy approximately coincide, e.g., deterministic dynamics. This yields a generalization of the size principle from concept learning [143] for stochastic dynamics and an information theoretic interpretation of this model as quantifying how atypical the demonstrations are under the random action hypothesis [42].

Demonstration Informed Specification Search ([155], Ch 5): We contribute a family of approximate algorithms (DISS) that systematically reduce the problem of learning task specifications from demonstrations into a series of supervised task specification learning problems. Compared to prior work, DISS can search through very large and unstructured concept classes, e.g., automata, even without strong syntactic priors. Furthermore, DISS only requires black-box access to a maximum (causal) entropy planner and a supervised task specification learner. As such, the algorithms are agnostic to the underlying dynamics model and task specification representation. This flexibility enables both efficient learning and teaching (see below) of task specifications. A concrete instance [154] of this algorithm for specifications in the form of automata provides empirical evidence that this approach can scale to combinatorially explosive classes of tasks.

Teaching Specifications using Demonstrations (Ch 6): We provide an adaption of the showing vs doing framework [73] which, given a model of a learner, provides a principled means to measure how pedagogic a series of expert demonstrations is. This enables a formalization of the problem of teaching specifications using demonstrations. To handle large spaces of task specifications, e.g. automata, we provide a counterexample driven algorithm (using DISS as a surrogate learner) for finding pedagogic demonstrations. Finally, we show the efficacy of this algorithm at teaching a collection of task specifications. In particular, we ran an online human study in which we gave participants a series of demonstrations of an unknown task specification and asked them to label behaviors as good or bad. Some of the participants were given pedagogic (showing) demonstrations generated using our algorithm. We show that this improved performance compared to a baseline where one simulates expert behavior (doing).

Control Improvisation in Stochastic Games ([150], Ch 7): We study Entropic Reactive

Control Improvisation (ERCI), a decision variant of maximum causal entropy policy synthesis specialized for task specifications in stochastic games. ERCI provides an algorithmic way to trade performance and randomization in stochastic games. Stochastic games combine both adversarial and probabilistic behavior in an environment, enabling modeling flexibility, which facilitates model compression and applicability to new domains, e.g., using probability ranges rather point probabilities to combine similar states/actions or explicitly capture uncertainty in the dynamics model. Further, we provide algorithms to decide ERCI whose certificates are policies that maximize the causal entropy subject to the performance constraints.

Stochastic Games as Circuits ([152], Ch 8): In order to succinctly represent stochastic games with history-dependent task specifications, we study modeling probabilistic transition systems as circuits. The result is a framework for analyzing queries about stochastic games, e.g., what is the probability of satisfying the task specification when applying actions uniformly at random, using tools such as Boolean Satisfiability (SAT) solvers [44, 101] and Binary Decision Diagrams [22, 23].

Stochastic Games as BDDs ([153], Ch 9): Building on the encoding stochastic games as circuits, we study the implications of using a Binary Decision Diagram (BDD) to represent this circuit. The result is a succinct encoding of a timed unrolled stochastic game that is well suited for entropy regularized planning and ERCI. Furthermore, we show that the worst case size of these BDDs grows linearly in the horizon and quasi-linearly in the number of actions. Finally, because BDDs support function composition, we shall show that the horizon of the time unrolled stochastic game can be incrementally expanded. This enables working with a compressed representation of the dynamics model throughout the entire planning process.

1.2 Reader's Guide

The structure of this dissertation reflects the three problems and contributions discussed above: Learning, Teaching, and Control (see Fig 1.3). Sections that are not necessary to understand the main contributions are denoted by †. These can be skipped during a first reading of this dissertation.

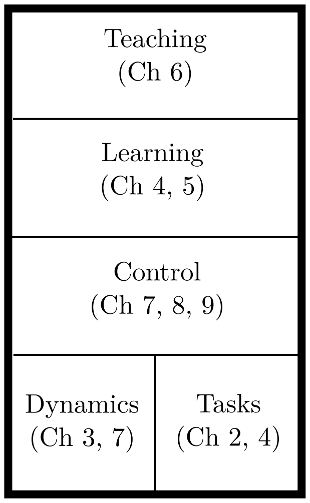


Figure 1.3

Preliminaries

We start with prelude chapters (Ch 2 and 3) which recount technical machinery necessary for the rest of the dissertation. In particular, we establish the formal tools by which to encode concept classes, model stochastic workspaces, and robustly predict the behavior of purposeful agents.

Learning and Teaching

Next, we have Part I which covers the learning and teaching problems. Ch 4 starts the part by formally introducing task specifications and discussing the necessary changes to existing theory on entropy regularized planning and learning from demonstrations to incorporate task specifications. This then leads to a formal statement of the learning problem, modeling tricks that highlight the expressiveness of stochastic dynamics and task specifications, and an analysis of the special case of deterministic dynamics. Unfortunately, the discrete and non-Markovian nature of task specifications makes them ill-suited for standard learning from demonstration algorithms. To address this, Ch 5 derives a systematic reduction from learning from demonstrations to a series of supervised learning problems. This results in a family of approximate algorithms called Demonstration Informed Specification Search (DISS). Ch 6 closes Part I by formalizing the teaching problem and providing a counterexample driven algorithm (using DISS as a surrogate learner) for finding pedagogic demonstrations.

Prediction and Control

The results in Part I assume access to a maximum causal entropy planner to be used black-box when predicting the actions of an agent trying to perform a given task. Part II studies how to efficiently perform maximum causal entropy likelihood estimates in stochastic games. Ch 7 starts with a discussion of control improvisation - the decision problem corresponding to entropy regularized planning in Markov Decision Processes, Interval Markov Decision Processes, and more generally, Stochastic Games. The result will be planning algorithms that scale in the size of the time-unrolled stochastic game. To keep the size of the time-unrolled games tractable, we change perspective in Ch 8 and explore modeling probabilistic transition systems as probabilistic circuits. This sets the stage for Ch 9 in which we discuss storing compressed representations of probabilistic transition systems using Binary Decision Diagrams (BDDs). In particular, we shall see that the maximum causal entropy planning can be done directly on this compressed structure.

1.3 Bibliographic Notes

Each chapter will end with bibliographic notes that highlight connections to other literature and historical notes. This chapter is slightly different in that it also contains acknowledgements of the contributions of peers and mentors (in roughly chronological order).

Thesis conceptualization

The idea to learn task specifications from demonstrations was inspired by the VeHICaL NSF Cyber-Physical Systems (CPS) Frontier project. The formalism in terms of maximum entropy inverse reinforcement learning spawned from interactions with Anca Dragan and Dorsa Sadigh - particularly Anca's class on Algorithmic Human Robot Interaction. This work was then mentored and encouraged by Susmit Jhah and Ashish Tiwari at SRI International. The theory and motivations were then refined through discussions with Mark Ho, leading to our initial work [149] on learning task specifications from demonstration. In terms of the contributions above, this corresponds to the special case of maximum entropy task specification learning. The connection with the size principle is due to Tom Griffiths.

The road to Maximum Causal Entropy Planning

Generalizing to the more complex setting of causal entropy required developing additional machinery to efficiently compute maximum causal entropy policies. The first insight came in modeling Markov Decision Processes as circuits. This required a non-trivial amount of engineering and would not have been possible without the mentorship of Markus Rabe. The insights from this work form the basis for Ch 8 and [151]. This circuit encoding then led to the BDD-based maximum casual entropy planner seen in Ch 9 and [153]. The generalization to the stochastic game setting [150] was joint work with Sebastian Junges and inspired by Daniel Fremont's work on Reactive Control Improvisation [55].

Learning and Teaching

Maximum causal entropy planner in hand, the next question to tackle was how to efficiently search for tasks given very large concept classes - solved in Ch 5 using the DISS algorithm [155]. The first versions of what would become DISS originate from discussions with Gil Lederman with later refinements thanks to the mentorship of Steven Piantadosi. The SAT-based DFA concept sampler used by DISS in Ch 5 was made possible with the help of Ameesh Shah and Vint Lee.

The main ideas for teaching using demonstrations again originate from interactions with Anca Dragan, with the ultimate version drawing heavy inspiration from Mark Ho's work on showing vs doing [73]. The core of the algorithm shown in Ch 6 was conceived shortly after our initial work on maximum entropy task specification inference. An initial version of the ideas appeared in [148] and was joint work with Mark Ho and Tom Griffiths. Unfortunately, a proper treatment of these ideas required a robust and scalable way to learn task specifications from demonstrations. DISS provided this missing ingredient with the algorithm discussed in Ch 6 being the result. The corresponding human study was in no small part due to Mark Ho, Ameesh Shah, and Tom Griffiths.

Finally, implicit in all the contributions is the influence of Sanjit Seshia. It was due to his training and mentorship (particularly in formal methods) that many of the problems that seem combinatorially intractable proved reasonable, and it was his mentorship that crystallized the motivations for learning formal specifications.

Chapter 2

Concepts and their Representations

Knight: *The name of the song is called 'Haddocks Eyes'!*
Alice: *Oh, that's the name of the song, is it?*
Knight: *No, you don't understand, that's what the name is called. The name really is, 'The Aged Aged Man.'*
Alice: *Then I ought to have said "That's what the song is called?"*
Knight: *No, you oughtn't: that's quite another thing! The song is called 'Ways and Means', but that's only what it is called you know!*
Alice: *Well, what is the song then?*
Knight: *I was coming to that. The song really is 'A-sitting on a Gate' and the tune's my own invention.*

Lewis Carroll (1871, Through the Looking Glass)

This thesis lies at the intersection of many sub-disciplines, and as the chapter quote suggests, the following chapters play the important role of developing a consistent terminology and mathematical machinery. The later chapters on learning, teaching, and control will not only rely on models of the dynamics/agent, but on the set of behaviors that constitute the agent's task (objective). Furthermore, we shall later find ourselves concerned with the difficulty (or ease) of describing a given task and the difficulty (or ease) of describing the agent's behavior given the task.

We will begin with a brief introduction to the study of formal languages, concept classes, and representations classes. This will lay the groundwork for encoding the set of behaviors that constitute an agent completing a task in a dynamical system. We start with the abstract formalism of a concept class.

Definition 2. Let $\mathcal{U}$ denote an arbitrary set, called the **universe**, whose elements, $x \in \mathcal{U}$, are called **atoms**. A concept is a set of atoms, $\varphi \subseteq \mathcal{U}$. A **concept class**, $\Phi \subseteq 2^{\mathcal{U}}$, is a collection of concepts. When clear from context, we shall abuse notation and denote by $\varphi : \mathcal{U} \to \{0, 1\}$, the indicator $[x \in \varphi]$.

Example 2.0.1. *Let* $\mathcal{U}$ *be the natural numbers,* $\mathbb{N}$*. The sets,* $\varphi_1 = \{1, 2, 3\}, \varphi_2 = \{x \in \mathbb{N} \mid x \text{ is even}\}$*, and* $\varphi_3 = \mathbb{N}$ *are all concepts. The collection* $\{\varphi_1, \varphi_2, \varphi_3\}$ *is a finite concept class over* $\mathbb{N}$*. The family of concepts* $\{x^k \mid x \in \mathbb{N}\}$ *for all* $k \in \mathbb{N}$ *is an infinite concept class.*

2.1 Formal Languages and Automata

As motivated in the introduction, we will later find ourselves interested with describing what *sequences* of states and observations constitute the successful completion a task. The notion of a sequence is formalized as a string.

Definition 3. Let Σ be a set of symbols, called an **alphabet**. A string is a finite sequence of symbols from Σ. We denote by $|x|$ the length of a string x. The **empty string** is the unique string of length 0, denoted ϵ. The **concatenation** of two strings, x, y, is denoted $x \centerdot y$, where x and y are called the **prefix** and **suffix** respectively. The concatenation of a sequence of strings, $x_1, \ldots, x_n$ is denoted $\prod_{i=1}^{n} x_i$. For any natural number k, we denote by x^k the concatenation of x with itself k-times, where $x^0 \overset{\text{def}}{=} \epsilon$.

Concepts over the universe of strings are called formal languages.[1]

Definition 4. A **formal language**, L, is a set of strings over alphabet Σ. The **concatenation** of two languages L_1 and L_2, denoted $L_1 \centerdot L_2$, is the set $\{x \centerdot y \mid x, y \in L_1 \times L_2\}$. For any natural number k, we denote by L^k the concatenation of L with itself k times, where $L^0 \overset{\text{def}}{=} \{\epsilon\}$. The **Kleene closure** of L is defined as $L^* \overset{\text{def}}{=} \bigcup_{k=0}^{\infty} L^k$. The **residual language** (Brzozowski derivative) of language L by string x, denoted $x^{-1}L$ is $\{y \in \Sigma^* \mid x \centerdot y \in L\}$, i.e., the set of suffixes of strings with prefix x in L.

There are many well studied concept classes of formal languages, e.g., the Chomsky Hierarchy of regular languages, context free grammars, context sensitive grammars, and recursively enumerable languages. For purposes of this thesis, it suffices to focus just on the class of regular languages.

Definition 5. Given a finite alphabet, Σ, a language L is **regular** iff one of the following holds:

- $L \subseteq \Sigma$.
- $L = {L_2}^*$, where L_2 is regular.
- $L = L_1 \cup L_2$, where L_1 and L_2 are regular.
- $L = L_1 \centerdot L_2$, where L_1 and L_2 are regular.

We shall denote the set of regular languages as Reg.

[1]For a more developed introduction to formal languages, we point the reader to [133].

Regular languages have a number of attractive properties such as closure under a number of operations (e.g., union, intersection, complement, concatenation, Kleene closures, string reversal) and containing all finite languages (e.g., languages with bounded string lengths). Furthermore, a language is regular iff the number of residual languages of L is finite. Finally, Reg is exactly the set of languages recognized by a deterministic finite automaton (see below).

Definition 6. A **Deterministic Finite Automaton** (DFA) is a 5-tuple, $D = \langle Q, \Sigma, \delta, q_0, F\rangle$, where Q is a finite set of **states**, Σ is a finite alphabet, $\delta : Q \times \Sigma \to Q$ is the **transition function**, $q_0 \in Q$ is the **initial state**, and $F \subseteq Q$ are the **accepting states**. The transition function is lifted to strings, $\delta^* : Q \times \Sigma^* \to Q$, via recursive application of δ, i.e.,

$$\delta^*(q, x) \stackrel{\text{def}}{=} \begin{cases} q & \text{if } x = \epsilon \\ \delta(\delta^*(q, y), \sigma) & \text{if } x = y \centerdot \sigma \end{cases}. \tag{2.1}$$

A string x is an **access-string** for state q if $\delta^*(q_0, x) = q$. D is said to be trimmed if each state $q \in Q$ has at least one access-string. One says D **accepts** x if $\delta^*(q_0, x) \in F$. Denote by $L[D]$ the set of strings accepted by D from q_0. Finally, we say that D **recognizes** L iff $L[D]$.

Remark 2.1.1. Unless otherwise stated, we shall assume that D is trimmed.

In this thesis, DFAs are visualized as directed multi-graphs. Nodes map to states and edges map to symbols. An edge connects q to q' if $q \neq q'$ and the corresponding symbol $\sigma \in \Sigma$ transitions q to q', i.e., $\delta(q, \sigma) = q'$. The initial state is annotated with "start" and the accepting states are annotated with a concentric circle.

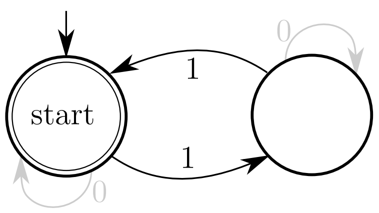
Example 2.1.2. *Consider the parity language, L_{parity}, over $\Sigma = \{0, 1\}$ containing the set of strings whose sum is even, i.e.,*

$$L_{parity} = \Big\{x \in \Sigma^* \mid \sum_{\sigma} x = 0 \pmod 2\Big\}. \tag{2.2}$$

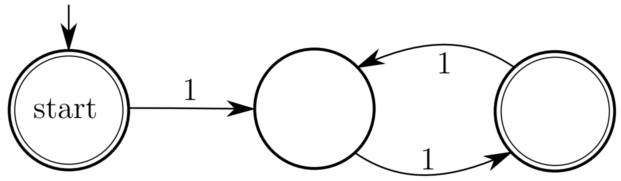
Figure 2.1 illustrates two distinct DFAs that recognize the L_{parity}. Because L_{parity} is recognized by a DFA, it is regular.

2.2 Representation Classes

The last example illustrates that multiple DFAs can represent the same concept. In fact, concepts can be generally expressed in a variety of ways, with some ways being easier to describe than others.



(a) Minimal DFA encoding of the parity language, implicit self loops greyed out.



(b) DFA encoding of the parity language with a redundant state.

Figure 2.1: Example of two DFAs that recognize the parity language.

Example 2.2.1. *Consider the concept,* $\varphi = \{1, 2, 3\}$. *One could describe the concept using a unary encoding of the elements, e.g.,* $\{|, ||, |||\}$, *as the intersection of two other primitive concepts, e.g.,* $\{1, 2, 3, 4, 6\} \cap \{0, 1, 2, 3\}$, *or using an indicator for* φ, *e.g.,* $x \mapsto (x > -1 \land x < 4)$.

Example 2.2.2. *Similar to DFAs, non-deterministic finite automata (NFAs) are a representation of regular languages as labeled transition systems. The two differences are that (i) NFAs allow a symbol to non-deterministically transition to one of several states. Lifted to strings, an NFA's transition function thus maps strings to a subset of its states and (ii) an NFA accepts a string if any of these non-deterministically accessed states is accepting. Importantly, this non-determinism allows certain regular languages to be expressed exponentially more succinctly than DFAs.*

We formalize the distinction between a concept and its description by means of a representation class which specifies two maps: an encoding map which associates objects with series of 0's and 1's (called a bit-string) and another which associates objects with a concept. The difficulty of describing a concept is formalized as the minimum length of the bit-string associated with that concept.

Definition 7. A **representation class** is a set $\mathcal{R}$ equipped with two maps:

$$\text{encode} : \mathcal{R} \to \{0, 1\}^* \qquad \text{concept} : \mathcal{R} \to \Phi, \tag{2.3}$$

where encode is injective. A representation class is **bijective** if **concept** is a bijection. The size or **description length** of a representation is given by the length of its encoding:

$$\text{size} : \mathcal{R} \to \mathbb{N} \qquad \text{size}(r) \overset{\text{def}}{=} |\text{encode}(\varphi)|. \tag{2.4}$$

Finally, given $\mathcal{R}$, we define the description length of a concept to be $\min_{r \in \mathcal{R}} \text{size}(r)$.

Remark 2.2.3. Note that because the encode map is injective with the countably infinite set, $\{0, 1\}^*$, the representation class must be countably infinite. This does not preclude the concept class from being over an uncountably infinite universe. For example, interval concepts with rational bounds are a valid representation class for intervals over the real numbers.

Example 2.2.4. *Consider the concept class, Φ, of all finite ordered sequences of natural numbers, which can be represented as tally marks, e.g., $5 \mapsto \text{𝍸}$. Observing that a sequence of tally marks can be encoded as a sequence of ones delimited by zeros (where the number of ones denotes the tally) yields:*

$$\begin{aligned} \textit{encode} &: (|||, |, , \text{𝍸}) \mapsto 1110100111110, \\ \textit{concept} &: (|||, |, , \text{𝍸}) \mapsto (3, 1, 0, 5). \end{aligned}$$

Note that for ordered sequences, the representation of tuples of tally marks is bijective. For un-ordered sequences (multi-sets), the representation class contains multiple representations of the same concept, and thus is not bijective.

While seemingly pedantic, this distinction proves immensely valuable when discussing the learnability of a concept and assigning prior probabilities on concepts, e.g., appealing to Occam's razor to order prior probabilities by size. Nevertheless, when clear from context, we shall often conflate representations and concept classes, particularly in the special case when concept($\bullet$) is bijective.

The DFA Representation Class

We shall find ourselves chiefly concerned with concept classes that model some variety of formal languages, and in particular regular languages. As such, we turn to constructing a particular representation class for Reg based on DFAs. We start by picking an arbitrary DFA for each regular language and relabeling states via a breadth first traversal from q_0, where transitions are taken using an arbitrary (but fixed) order on Σ. Thus, each state and symbol are associated to an integer and DFAs that are equivalent up to relabeled states are all represented by the same object. Then, we encode an arbitrary DFA, $D = \langle Q, \Sigma, \delta, q_0, F\rangle$, as follows.

Let b_Q and b_Σ denote the number of bits do encode $|Q|$ and $|\Sigma|$ respectively[2], i.e., $b_Q \overset{\text{def}}{=} \lceil \log_2 |Q| \rceil$ and $b_\Sigma \overset{\text{def}}{=} \lceil \log_2 |\Sigma| \rceil$. Define the encoding map:

$$\text{encode}(D) \overset{\text{def}}{=} \text{encode}(Q) \centerdot \text{encode}(\Sigma) \centerdot \text{encode}(F) \centerdot \text{encode}(\delta) \tag{2.5}$$

[2]Note that because Q and Σ can't be empty, no bits are allocated to this case.

where

$$\begin{aligned}
&\text{encode}(Q) \overset{\text{def}}{=} 1^{b_Q} \centerdot 0 \centerdot \text{encode}(|Q|-1),\\
&\text{encode}(\Sigma) \overset{\text{def}}{=} 1^{b_\Sigma} \centerdot 0 \centerdot \text{encode}(|\Sigma|-1),\\
&\text{encode}(\delta) \overset{\text{def}}{=} \prod_{\substack{q,\sigma,q' \\ q' \neq \delta(q,\sigma)}} \text{encode}(q) \centerdot \text{encode}(\sigma) \centerdot \text{encode}(q'),\\
&\text{encode}(F) \overset{\text{def}}{=} \begin{cases} 0 \centerdot \text{encode}(\ \ |F|-1\ \ ) \ \centerdot\ \prod_{q\in F} \text{encode}(q) & \text{if } 2|F| < |Q|+1 \\ 1 \centerdot \text{encode}(|Q \setminus F|-1) \ \centerdot\ \prod_{q\notin F} \text{encode}(q) & \text{otherwise} \end{cases},
\end{aligned} \tag{2.6}$$

for any natural number x, encode(x) is a standard binary encoding, and states, $q \in Q$ and symbols, $\sigma \in \Sigma$, are interpreted interpreted as integer indices. The encoding starts by declaring the number of bits required to encode the states. This is done by a unary encoding delimited by zeros. The unary declaration is followed by a binary encoding of the number of states. The number of bits and size of the alphabet is provided in the same manner. The next bit determines whether the accepting set is provided or its complement. The corresponding set is encoded by providing the binary encoding of its size followed by the binary encoding of the elements. Finally, the binary encoding of the state, token, next state tuples of the non-stuttering transitions is provided.

Letting size(D) $\overset{\text{def}}{=}$ |encode(D)| and letting m denote the number of labeled transitions s.t. $q \neq q'$, we see that for the DFA representation class:

$$\text{size}(D) = 3 + 2b_Q + 2b_\Sigma + (|F|+1)b_Q + m(b_\Sigma + 2 \cdot b_Q). \tag{2.7}$$

Assuming that Σ is fixed and noting that D being trimmed implies $|Q| \leq m+1$, we then have:

$$\text{size}(D) \in O\Big(m \log |Q|\Big) \tag{2.8}$$

Example 2.2.5. *Using the standard ordering,* $0 < 1$*, the minimal DFA shown in Figure 2.1a for the parity language,* D_{parity}*, has encoding:*

$$\textit{encode}(D_{parity}) = \underbrace{101}_{\textit{encode}(Q)} \centerdot \underbrace{101}_{\textit{encode}(\Sigma)} \centerdot \underbrace{001}_{\textit{encode}(F)} \centerdot \underbrace{011110}_{\textit{encode}(\delta)}. \tag{2.9}$$

The non-minimal DFA, D'_{parity} *shown in Figure 2.1b has encoding:*

$$\textit{encode}(D'_{parity}) = \underbrace{11010}_{\textit{encode}(Q)} \centerdot \underbrace{101}_{\textit{encode}(\Sigma)} \centerdot \underbrace{0000}_{\textit{encode}(F)} \centerdot \underbrace{1000100010101110101 01}_{\textit{encode}(\delta)}. \tag{2.10}$$

The MinDFA Representation Class

Note that within the above DFA representation class, for any regular language, there is an infinite number of DFAs that recognize the language. A natural question is if it is possible to construct a bijective representation class where only a single "canonical" DFA is associated with each language. Fortunately, up to state relabeling, for each regular language L, there exists a unique minimal DFA. To see this, the key observations are that (i) each access string of a state of D maps to a residual language; and (ii) the residual language of a state is independent of which access string was used. First, this implies that $|Q|$ is at least the number of residual languages. Second, this provides a "canonical" DFA construction based on respecting residual languages. Given a minimal DFA, D, one constructs a new DFA, $D' = \langle Q', \Sigma, \delta, q_0', F\rangle$, where Q' is the set of residual languages of $L[D]$, $q_0' = L[D]$, and $F' = \{q' \in Q' \mid \epsilon \in q'\}$. Next, to make $L[D] = L[D']$, δ' must respect the residual language structure, i.e., $\delta'(q', \sigma) = \sigma^{-1}q'$. Finally, to show the D' is canonical, one observes that the bijective mapping f from $q \in Q$ to its residual language respects transition functions and set acceptance, i.e., for all $q \in Q$, $q' \in Q'$ and $\sigma \in \Sigma$:

$$q \in F \iff f(q') \in F' \qquad \delta(q, \sigma) = f^{-1}\Big(\delta'\Big(f(q), \sigma\Big)\Big). \tag{2.11}$$

Thus, (δ, F) and (δ', F') are the same up to relabeling via f.

Remark 2.2.6. One can interpret the number residual languages of a language to be the amount of "memory" required to recognize a given language. In this way, the size of canonical DFA for a regular language, L, explicitly provides the minimum amount of memory needed to recognize L.

2.3 Learning Concepts from Examples

The ultimate goal of this thesis will be to learn concepts from unlabeled examples. As we shall elaborate in the next part, a fruitful way to tackle this problem is via a series of reductions to learning concepts from labeled examples. To facilitate this reduction, we review relevant definitions and approaches to learning concepts, and in particular DFAs, from labeled examples.

Definition 8. Given a universe, $\mathcal{U}$, a **labeled example** is tuple, $(x, l) \in \mathcal{U} \times \{0, 1\}$. A labeled example is **consistent** with a concept φ if $l = [x \in \varphi]$. A collection of labeled examples, $\mathbb{X}$, is consistent with concept, φ, if all its elements are consistent with $\mathbb{X}$. Similarly, labeled examples are consistent with a representation r if they are consistent with concept(r).

Example 2.3.1. *Let* $\mathbb{X} = \{(0, 0), (1, 1), (01, 1), (10, 1), (11, 0)\}$ *be a collection labeled examples over the universe of bit-strings,* $\mathcal{U} = \{0, 1\}^*$. $\mathbb{X}$ *is consistent with the parity language and the universal language,* $\mathcal{U}$. $\mathbb{X}$ *is not consistent with the empty language,* $\emptyset$*, or the complement of the parity language.*

Given a set of labeled examples and a representation class, a natural question is to find a subset of representations that are consistent with the labeled examples. This is formalized by means of an identification map.

Definition 9. Given a representation class, $\mathcal{R}$, an *identification map*, $\mathcal{I}$, associates a tuple of labeled examples and a reference representation, $\langle \mathbb{X}, r\rangle$, to a distribution over consistent representations in $\mathcal{R}$, i.e.,

$$\mathcal{I} : \mathbb{X} \times \Big(\mathcal{R} \cup \{\bot\}\Big) \to \mathit{Distr}(\mathcal{R} \cup \{\bot\}),$$

where $\bot$ is adjoined to the representation class to indicate the lack of a representation.

Remark 2.3.2. Because representation classes are generally countably infinite, identification maps inherit limitations of distributions over natural numbers. In particular, there is no uniform distribution over countably infinite sets, and thus, unless the representation class is finite, $\mathcal{I}$ is necessarily favors certain representations over others.

Remark 2.3.3. Let $\mathbb{X}$ be a set of labeled examples. If for all representations, $\mathcal{I}$ can be viewed as conditioning on sampling a consistent representation, i.e.,

$$\mathcal{I}(r' \mid \mathbb{X}, r) \propto \begin{cases} \mathcal{I}(r' \mid \bot, r) & \text{if } r \text{ is consistent with } \mathbb{X}, \\ 0 & \text{otherwise,} \end{cases}$$

then $\mathcal{I}$ can be re-expressed as an exponential bias to representations with small **relative description length** given r. Formally, define:

$$\text{size}(r' \mid r) \overset{\text{def}}{=} -\log\Big(\mathcal{I}(r' \mid \bot, r)\Big). \tag{2.12}$$

Then,

$$\mathcal{I}(r' \mid \mathbb{X}, r) \propto e^{-\text{size}(r' \mid r)}. \tag{2.13}$$

For example, when we are later learning DFAs from demonstrations, we will find it useful to sample a consistent DFA exponentially weighted by how many bits are needed to describe the new DFA given a reference DFA, i.e., which states, edges, and accepting labels changed.

As the previous remark illustrates, one can construct an identification map from an “exact” learning algorithm that finds concepts consistent with the examples and weights them according to their size. Many instances of exact algorithms exist for a wide range of representation classes, e.g. DFAs, decision trees, half planes, etc. In this thesis, we shall use identification algorithms black-box. Nevertheless, as an illustrative example, we shall outline the problem finding DFAs consistent from labeled examples. Through this exposition, we shall introduce several ideas that are used later in this thesis and in particular, augmented prefix trees.

Prefix Trees

When learning formal languages in the form of DFAs, it is useful to model the set of labeled examples as a single object that encodes the prefix relations between strings. This idea is formalized by means of a prefix tree.

Definition 10. Let X be a finite set of strings and let X' be its prefix-closure. Let $\preceq$ denote the **prefix-relation** on X', i.e., given strings x, x', $x \preceq x'$ iff $x' = x \centerdot y$. A **prefix-tree** (also called a trie or digit tree), T_X is a rooted tree with nodes X' whose edges represent the transitive reduction of $\preceq$, i.e. if x' is a descendent of x then $x' = x \centerdot y$.

An example is provided in Figure 2.2. Prefix trees are used in a number of areas ranging from predictive text completion [40] to Bioinformatics [102]. Below we observe a few properties of

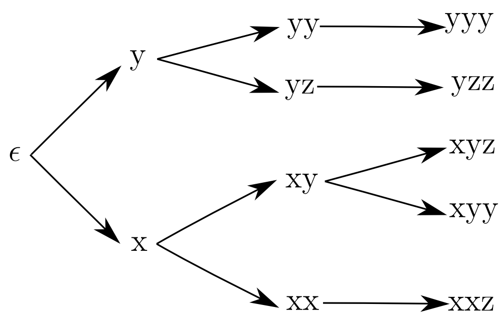


Figure 2.2: Prefix tree for the strings: $\{xx, xyz, xxz, xyy, yzz, yyy\}$.

prefix trees to be later used in this thesis. Fix a prefix tree T_X for a set of strings X. First, the root of the prefix-tree is always the empty string: ϵ. Second, the number of leaves in a prefix tree is exactly $|X|$. Third, the depth of the prefix tree is the maximum length of a string in X, i.e. $\max_{x \in X} |x|$. Fourth, the maximum out degree of a node in T, called the *branching factor* of T_X, is at most the number of visited symbols, i.e., $|\{\sigma \in \Sigma : \exists w \in X \,.\, w = x \centerdot \sigma \centerdot y\}|$. Fifth, expanding Σ to include additional elements does not change T_X.

Remark 2.3.4. Importantly, if $|\Sigma|$ is much bigger than X and the longest word in X, then the size of T_X will be be much smaller than Σ. We will later use this fact to efficiently represent the paths through a workspace when learning task specifications. Here the alphabet will model the set of actions and observations and typically be much larger than the prefix tree of the demonstrations.

Next, observe that if two sets have the same prefix closure, then they have the same prefix tree. For example $X_1 = \{xx, xyz, xxz, xyy, yzz, yyy\}$ and $X_2 = \{xyz, xxz, xyy, yzz, yyy\}$ have the same prefix tree. To make the original (multi-)set of strings, X, recoverable, one can augment the nodes of the prefix tree, X', with a *visitation counter*:

$$\# : X' \to \mathbb{N} \qquad \# : x \mapsto |\{y \in X \mid y = x \centerdot z\}|. \tag{2.14}$$

A prefix in $x \in X$ is then detected whenever (i) x is a leaf in T; or (ii) the visitation count of a node does not add up to the visitation counts of its children. For example, for X_1 and X_2 we have $\langle \#(xx), \#(xxz)\rangle = \langle 3, 2\rangle$ and $\langle 2, 2\rangle$ respectively. Since xxz is the only child of xx in T_X, one recovers that $xx \in X_1$ and $xx \notin X_2$ as desired.

Finally, we remark that by associating prefix tree nodes with a label $\{0, 1, \bot\}$ one obtains a representation of a (multi-)set of labeled examples, i.e.,

$$\text{label} : X' \to \{0, 1, \bot\} \qquad \mathbb{X}_T = \Big\{\Big(x, \text{label}(x)\Big) \mid x \in X', \text{label}(x) \neq \bot\Big\}, \tag{2.15}$$

where X' is the prefix closure of the strings of the labeled examples. Such a prefix tree is called an **augmented prefix tree** and is denoted, $T_{\mathbb{X}}$.

Learning DFAs from labeled examples†

Next, we study how one can identify DFAs from labeled examples. The first point to make clear is that this problem is fundamentally under constrained. For any finite set, $\mathbb{X}$, of labeled examples, there exists an infinite number of regular languages consistent with $\mathbb{X}$. Thus, rather than identifying a single DFA, we shall seek to find a distribution of DFAs. To do so, we first characterize the collection of DFAs that are consistent with a given example set, $\mathbb{X}$.

Let $\mathbb{X}$ be a collection of labeled examples and $T_{\mathbb{X}}$ its augmented prefix tree. A $k-$**coloring** of the prefixes, X', is a mapping,

$$\text{color} : X' \to \{1, \ldots, k\},$$

that induces a partitioning of the nodes (prefixes) of $T_{\mathbb{X}}$, i.e., $X' = \bigcup_{q \in Q} q$ and each $q, q' \in Q$ is disjoint. The partition of prefix $x \in X'$ is denoted q_x.

Remark 2.3.5. The notation for the partitioning, Q, was chosen to highlight that the coloring partially defines the states of a DFA (expanded on below).

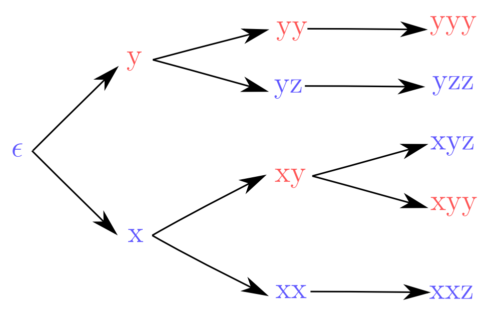
A coloring is consistent with the augmented prefix tree if it respects the labeling function:

$$\forall x, x' \in Q \,.\, \Big(\text{label}(x) \neq \bot \wedge \text{label}(x') \neq \bot\Big) \implies \text{label}(x) = \text{label}(x'), \tag{2.16}$$

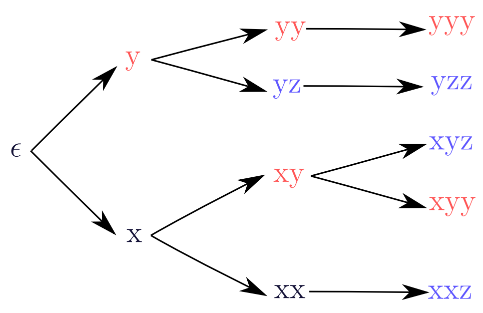
and it respects the prefix ordering:

$$\forall q \in Q \,.\, \forall x, x' \in q \,.\, \forall y \in \Sigma^* \,.\, \Big(x \centerdot y, x' \centerdot y \in X'\Big) \implies \Big(q_{x \centerdot y} = q_{x' \centerdot y}\Big). \tag{2.17}$$

That is, a coloring is consistent if any two nodes with the same color are indistinguishable given the label and transition constraints imposed by the augmented prefix tree.



(a) Consistent 2 state coloring:
$q1 = \{\epsilon, x, xx, xxz, xyz, yzz, yz\}$
$q2 = \{y, yy, yyy, xy, xyy\}$.



(b) Consistent 3 state coloring.
$q_1 = \{\epsilon, x, xx\}$ $q_2 = \{xxz, xyz, yzz, yz\}$
$q_3 = \{y, yy, yyy, xy, xyy\}$.

Figure 2.3: Two consistent colorings for the prefix tree for example 2.3.6.

Example 2.3.6. *Consider the labeled example set,*

$$\mathbb{X} = \{(yyy, 1), (xyy, 1), (y, 1), (yzz, 0), (xyz, 0), (xxz, 0)\}.$$

Two consistent colorings for $\mathbb{X}$ *are provided in Fig 2.3.*

Remark 2.3.7. Note that assigning a unique color to each prefix, i.e., $q_x = \{x\}$, yields a trivially consistent coloring. Thus, when searching for a single consistent coloring, one can restrict their attention to $|X'|-$colorings.

Remark 2.3.8. Given any consistent k-coloring, one can construct a $(k+1)$-coloring by extending the co-domain of the color map. The result are "unused" colors.

Operationally then, we see that a coloring partially defines a DFA, $D = \langle Q, \Sigma, \delta, q_0, F\rangle$. The states Q are the colors, the transition function is partially defined by the prefix relation, i.e. if there is a path from x to $x' = x \centerdot y$ in the prefix tree, then $\delta^*(q_x, y) = q_{x'}$, the initial state is the color of the empty string, i.e., $q_0 = q_\epsilon$, and the accepting states are upper and lower by label constraints, i.e.,

$$\{x \in X' \mid \text{label}(x) = 1\} \subseteq F \subseteq Q \setminus \{x \in X' \mid \text{label}(x) = 0\}. \tag{2.18}$$

A general DFA extraction algorithm then works as follows:

1. Select a $k \in \mathbb{N}$.
2. Select a $k-$coloring consistent with $T_{\mathbb{X}}$. If none exists, return to step 1.
3. Construct a DFA, by arbitrary completion of the partially defined accepting set and transition function.

Example 2.3.9. *Building on example 2.3.6, Fig 2.4 shows two DFAs that are consistent with the coloring in Fig 2.3a. These DFAs differ in the completion of the transition function. In particular, the transitions with symbol* x *after seeing symbol* y *are left unconstrained by* *the coloring and augmented prefix tree. There are 12 two state and 2,243 three state DFAs consistent with* $\mathbb{X}$.

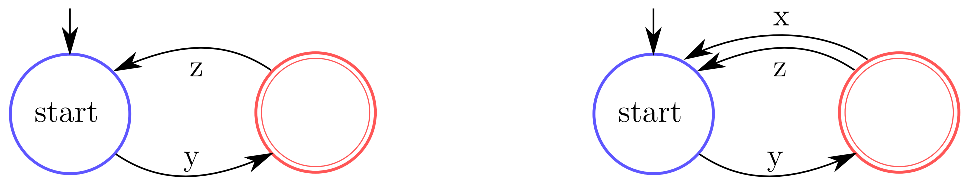


Figure 2.4: Two consistent DFAs derived from the 2-coloring in Fig 2.3a.

Remark 2.3.10. From our previous observations, we see that there is always a $|X'|$ state DFA that is consistent with the augmented prefix tree. Similarly, we see that extending a k-coloring to a $(k+1)$-coloring introduces a state that is not accessed by any of the labeled examples. Such extensions are necessary for making the set of DFAs that are partially defined by a coloring complete. Note however that when considering canonical (minimized) DFAs the augmented state may or may not be equivalent to a visited state, and thus arbitrarily filling in the rest of the partial DFA's details may result in a DFA with less than $k+1$ states.

Through the lens of coloring, one is able to derive several algorithms for finding DFAs that are consistent with a labeled example set. Generally, these are characterized by (i) if the coloring is generated incrementally, e.g., via greedy state merging [92] (ii) if one or several DFAs is to be found, and (iii) if the found DFAs are ordered by description complexity. We shall focus on algorithms that enumerate the smallest k-DFAs, as measured by the size function.

Remark 2.3.11. The problem of deciding if a DFA is a minimally-sized DFA consistent with a set of labeled examples is known to be NP-hard [59]. As such, there exist a number of reductions to the canonical NP-complete problem, Boolean satisfiability (SAT) [41, 70, 146]. The high-level approach to the SAT reduction of exact learning of DFAs used in the experiments of this thesis are as follows: (i) Create a SAT query, i.e., a Boolean predicate $f : \{0,1\}^n \to \{0,1\}$, which is true iff there exists DFA with a given number of states, m; (ii) Perform a linear search to determine the minimum number of states, m^*, required; (iii) Create a SAT query that is true iff there exists a DFA with m^* states and k edges in its multi-graph; (iv) Binary search for the minimum number of multi-graph edges required; (v) Until satisfied, enumerate all solutions[3], increasing m and k as necessary. Finally, by enumerating a fixed number of DFAs (with distinct regular languages), one can create an identification map by weighting the DFAs according to the size relative to another reference DFA.

[3]Given a SAT query f and a solution, x^*, one can create a new SAT query: $f(x) \mapsto f(x) \wedge (x \neq x^*)$, which is either un-satisfiable, or only satisfiable with a new solution. This enables enumerating solutions.

2.4 Bibliographic Notes

The perspectives on concept and representation classes reviewed in this chapter derive from computation learning theory [85] and to a lesser extent Solomonoff's theory of inductive inference [136]. Both literatures directly address the (sobering) reality that computers operate on discrete objects with finite description lengths. The implications of this reality include the inability to encode arbitrary real numbers or assert uniform priors on infinite representation classes. While not a large problem when representations are indexed by parameters, e.g., linear rewards, this becomes a fundamental issue when encoding historical dependencies, where each bit of memory requires a larger and larger description. Furthermore, the computational resources used by algorithms for learning concepts from non-trivial concept classes are invariably sensitive to description length of the target concept in the chosen representation class.

As a illustrative example, we largely focused on minimized DFAs, an incredibly well studied representation of regular languages [133]. The focus on regular languages stems from their ability to support temporal and Boolean composition and their foundational nature in the Chomsky hierarchy of formal languages [133]. DFAs support a number of inference algorithms [45] in both the passive learning setting [70] and the active learning setting [12]. In fact, the notion of a prefix tree as discussed in this chapter derives from the passive learning of DFAs. Nevertheless, as much as possible, the later material will try to be agnostic to the particular representation class being considered.

Finally, the key distinction in this thesis from prior work on grammatical inference [45] will be the treatment of learning from unlabeled and potentially noisy demonstrations, where demonstrations differ from examples due to the existence of a dynamics model. As we'll see in the next chapter, this notion of demonstration derives from the Inverse Reinforcement Learning literature [111]. In doing so, we shall see that one sometimes only requires a few positive demonstrations to confidently identify a language. This goes against conventional wisdom in formal language induction that one cannot identify regular languages using only positive examples [60]. The loophole of course is that we do not seek to exactly learn the regular language, but rather a belief distribution over regular languages which will (hopefully) concentrate on the correct language. This goal mirrors other results on learning from positive examples on simple distributions [47, 160]. As we shall see though, our results will go further and not even require positive labels.

Chapter 3

Workspaces and Decision Processes

You have brains in your head. You have feet in your shoes. You can steer yourself in any direction you choose. You're on your own, and you know what you know. And you are the guy who'll decide where to go.

Dr. Seuss (Oh, the Places You'll Go!, 1990)

In the previous chapter, we reviewed machinery for representing concepts and formal languages. This chapter reviews additional machinery for predicting the behavior of agents that are trying to perform tasks in stochastic environments. We start with a brief review a few useful probability distributions. We shall assume the reader is familiar with elementary probability theory.

Definition 11. Let X be a discrete set. The **uniform** distribution on X is denoted $\text{Uniform}(X)$. The distribution with all probability mass on a single element $x \in X$ is called the **Dirac** distribution, δ_x. If X is equipped with a map $f : X \to \mathbb{R}$, then we denote the **Log Sum of Exponentials** (or smooth max) of f by:

$$\text{LSE}(f) \overset{\text{def}}{=} \log \sum_{x \in X} e^{f(x)}. \tag{3.1}$$

The **softmax** (or soft-argmax) is the gradient of $\text{LSE}(f)$ interpreted as a distribution over X:

$$\text{softmax}(f) \overset{\text{def}}{=} \nabla \, \text{LSE}(f) \qquad \text{softmax}(f)(x) \propto e^{f(x)}. \tag{3.2}$$

A **random variable**, $\mathcal{X}$, is a function relating two discrete distributions.

Remark 3.0.1. As the names suggest the smooth max and soft-argmax are differentiable relaxations of the max and $\arg\max$ functions respectively. For example, one can easily verify that $0 \leq \text{LSE}(f) - \max_{x \in X}(f(x)) \leq \ln |X|$ with the left inequality becoming tighter as the difference between the maximum of f and other elements becomes larger. A similar analysis shows that softmax(f) interpolates between the uniform distribution (when f is a constant) and the Dirac distribution (when the max of f becomes much larger than all other elements).

3.1 Markov Decision Processes

We model an agent acting in a stochastic domain as a Markov Decision Process.

Definition 12. A **probabilistic labeled transition system** (PLTS) is a tuple $M = \langle S, s_0, A, P \rangle$, where S is a finite set of **states**, $s_0 \in S$ is the **initial state**, A is a finite set of **actions**, and $P\colon S \times A \to \mathit{Distr}(S) \cup \{\bot\}$ is the *transition function* of M. Viewing states and actions as random variables, we denote by $P(s' \mid s, a)$ the distribution over next states, s', induced by P given the current state s and action a. If M is equipped with a **reward function**, $r : S \to \mathbb{R}$, it is known as a finite **Markov decision process** (MDP).

We identify the **available actions** as: $A(s) \overset{\text{def}}{=} \{a \mid P(s, a) \neq \bot\}$, e.g., a door can only be opened when close enough to the door. States without available actions, i.e., states with $A(s) = \emptyset$ are called **terminal states**. Finally, we shall assert the Luce choice axiom, which requires that each action, $a \in A(s)$, be **distinct**, i.e., no actions are interchangeable or redundant at a given state [99].

Remark 3.1.1. When *learning* history-dependent rewards, it is a-priori unclear what state augmentations are necessary to make the unknown reward function state-based. Nevertheless, unless the distinction is important, we shall often conflate Markov Decision Processes and their underlying Probabilistic Labeled Transition Systems.

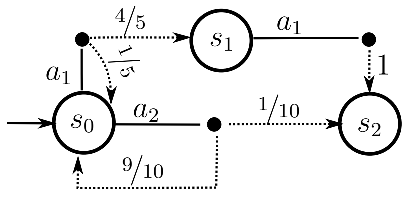


Figure 3.1: Example MDP in the form of a bi-partite graph.

Example 3.1.2. *A PLTS with three states and two actions is illustrated as a directed bi-partite graph in Fig 3.1. The nodes of the graph are split into* ***ego-nodes*** *and* ***dist-nodes****. Ego-nodes, depicted as circles with black outlines, represent states and the dist-nodes, depicted as black disks, represent state distributions, e.g.* $P(\bullet \mid s, a)$*. The outgoing edges of dist-nodes are annotated with weights that sum to 1. For example, the PLTS shown in Fig 3.1 has*

$P(s_0 \mid s_0, a_2) = 9/10$ and s_2 is the unique terminal state of this PLTS. Given a state reward function, e.g., $r : s_i \mapsto i$, the PLTS would also be an MDP.

Ultimately, we shall be interested in what behaviors an agent can and does generate in a workspace. A behavior is formalized by the notion of a path through an MDP.

Definition 13. Given a MDP, $M = \langle S, s_0, A, P\rangle$, a $n-$length **path** ξ is a sequence:

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots \xrightarrow{a_n} s_n, \tag{3.3}$$

in $(S \times A)^n \times S$ where $P(s_{i+1} \mid s_i, a_i) > 0$ for each i. We denote the length of a path with $|\xi| \overset{\text{def}}{=} n + 1$, and denote s_n, i.e., the last element of ξ, with $\text{last}(\xi)$. A path, ξ, is said to be **complete** if $\text{last}(\xi)$ is a terminal state. The set of all paths is denoted Paths^M and the set of complete paths is denoted $\mathsf{Paths}_\$^M$. We omit M when it is clear from the context.

We will often find it convenient to view paths as strings that alternate between states and actions. So much so that this perspective was already implicitly employed to visualize paths in (3.3). To this end, we allow paths to inherit string operations such as concatenation and decomposition into prefix-suffix pairs. Next, we turn to the question of modeling an agent's decisions and control strategies. This is captured by the notion of an agent policy.

Definition 14. A **policy**, $\pi\colon \text{Paths} \to \mathit{Distr}(A)$, is a map from paths to distributions over enabled actions, i.e.,

$$\{a \in A \mid \pi(s) > 0\} \subseteq A(s).$$

Viewing actions and paths as random variables, for a given path, ξ and a policy π, we denote by $\pi(a \mid \xi)$ the distribution of actions induced by π given the path ξ.

Employing a policy, π, in a given MDP induces a Markov Chain over paths. More precisely, the probability of a path, $\Pr(\xi \mid \pi, M)$, is recursively given by:

$$\Pr(\xi \mid \pi, M) = \begin{cases} 1 & \text{if } |\xi| = 1, \\ P(s' \mid s, a) \cdot \pi(a \mid s) \cdot \Pr(\xi' \mid \pi, M) & \text{if } \xi = s \xrightarrow{a} s' \centerdot \xi'. \end{cases} \tag{3.4}$$

The probability of a prefix-free set $X \subseteq \mathsf{Paths}$ of paths, is the sum over the individual path probabilities,

$$\Pr(X \mid \pi) = \sum_{\xi \in X} \Pr(\xi \mid \pi). \tag{3.5}$$

We denote sampling a complete path, ξ, given policy π and MDP M by $\xi \sim (\pi, M)$.

Example 3.1.3. *Consider again the MDP shown in Fig 3.1. Assuming the policy:*

$$\pi(a \mid \xi) \propto \begin{cases} 1 & \textit{if } a = a_1, \\ |\xi| - 1 & \textit{otherwise}, \end{cases} \tag{3.6}$$

the complete path $\xi = s_0 \xrightarrow{a_1} s_0 \xrightarrow{a_2} s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_1} s_2$ *has probability:*

$$\left(1 \cdot \frac{1}{5}\right) \cdot \left(\frac{1}{2} \cdot \frac{9}{10}\right) \cdot \left(\frac{1}{3} \cdot \frac{4}{5}\right) \cdot (1 \cdot 1) = \frac{3}{125}.$$

Remark 3.1.4. In the standard MDP setting, it is sufficient to only consider state based policies, $\pi : S \to \mathit{Distr}(A)$. While useful for control, for our application of learning from behaviors, there is no reason to believe that the demonstrator uses a stationary state-based policy. In fact, we shall later see that bias minimizing predictions of the agents behavior, formalized as entropy, necessarily imply forecasting using history dependent policies.

3.2 Behavior Prediction and Agent Models

Next, we turn to the question of how to predict and explain the behavior of an agent acting in an MDP. Traditionally one models an agent operating in an MDP as using a policy that (approximately) optimizes the expected sum of rewards[1], i.e.,

$$\max_{\pi} \ \mathbb{E}\left[\sum_{t=1}^{\infty} r(s_t) \mid \pi, M\right]. \tag{3.7}$$

A path generated from such an agent is called a **demonstration**. The inverse problem, known as inverse reinforcement learning [1] (or inverse optimal control [82]), is informally defined below.

Definition 15. An **inverse reinforcement learning** (IRL) problem is a tuple, $\langle M, \Xi, R\rangle$, where M is an MDP, Ξ is a multi-set of demonstrations, and R is a class of reward functions.

A solution to an IRL problem is a reward, $r \in R$, that "explains" Ξ given M.

Unfortunately, in the context of IRL, (3.7) simultaneously proves to be a very strong and under-constrained assumption. At one extreme, any policy optimizes *all* constant rewards. At the other extreme, sufficiently expressive reward classes may contain rewards that "overfit" to the demonstrations. For example, if the transition function is deterministic, then the membership indicator $[\xi \in \Xi]$ makes the demonstrations (and only the demonstrations!) look optimal. Either way, some other criteria must be used to disambiguate rewards.

[1]To make the expectation is well defined one often assumes the episode ends with constant probability γ.

Secondly, when connecting MDPs to physical environments, the policy that optimizes (3.7) may require super-human precision, planning, and consistency. Thus the policies prescribed by (3.7) are often ill-suited for reasoning about human generated paths. This problem is further compounded if there is a gap between the dynamics model used for inference and the model used by the agent.

Finally, for any reward, one notes that there is often more than one policy that maximizes (3.7). For example, given a constant reward, both a uniform distribution of the actions and a deterministic policy are optimal, but the predictions of both are very different. Thus, again for prediction, some other criteria is necessary.

3.3 Entropy Regularized Planning and Prediction

A popular, and in practice effective, solution to the ambiguity of IRL is to appeal to the principle of maximum causal entropy. The result is a unique agent model that is no more biased (formalized as causal entropy) than is required to match various assumed or observed characteristics of the agent, e.g., keeping an average of 10 feet from a wall. In the sequel, we review relevant definitions from information theory and directed information theory. Then, we review maximum causal entropy inverse reinforcment learning.

Definition 16. The **surprisal** (or information content) of an event with probability p is $\log {}^1\!/_p$. Let P and Q denote two distributions over a discrete set X. The **cross entropy** of P under Q is the average surprisal w.r.t. Q under distribution P:

$$\begin{aligned} H_P(Q) &\overset{\text{def}}{=} \mathop{\mathbb{E}}_{x\sim P}\left[\log\frac{1}{Q(x)}\right] \\ &= -\sum_{x\in X} P(x)\log Q(x), \end{aligned} \tag{3.8}$$

where $0\log 0$ is taken to be 0. The **entropy** of distribution P is:

$$H(P) \overset{\text{def}}{=} H_P(P). \tag{3.9}$$

Finally, entropy is naturally extended a random variable, $\mathcal{X}$, using the distribution over the outputs of $\mathcal{X}$, and is denoted $H(\mathcal{X})$.

Remark 3.3.1. Note H is an functional and is invariant under relabelings (bijective transformations) of X.

Remark 3.3.2. Entropy is defined with respect to a given logarithmic base - with all variants being equivalent up to multiplicative constants. Common bases are 2 and e with the corresponding entropy units being bits and nats resp.

When log is taken to be base 2, entropy $H(P)$ can be interpreted (via the Source Coding Theorem [42]) as the average number of bits required to record values of X drawn from P. In this way, entropy informally captures the degree of uncertainty in a distribution. For example (again assuming base 2 entropy), if P is uniform, then $H(P) = \log |X|$. After rounding up to the nearest integer) $H(P)$ is exactly the number bits required to index all of the elements of X. On the other hand, if P is entirely concentrated on a single element of X, then $H(P) = 0$, i.e., no bits are required because the result of sampling a value from P is a-priori known.

Similarly, the cross entropy $H_P(Q)$ is interpreted to be the expected number of bits required to record values from X drawn from P - given an encoding optimized for Q. The excess number of bits used by $H_P(Q)$ relative to $H(P)$ is to known as the Kullback-Leibler divergence (or relative entropy).

Definition 17. Let P and Q denote two distributions over a discrete set X. The **KL-divergence** from Q to P is:

$$\begin{aligned} D_{\mathrm{KL}}(P \mid\mid Q) &\overset{\text{def}}{=} H_P(Q) - H_P(P) \\ &= \mathop{\mathbb{E}}_{x \sim P}\left[\frac{\log P(x)}{\log Q(x)}\right] \end{aligned} \tag{3.10}$$

Remark 3.3.3. Via Gibb's inequality, one can show that $D_{\mathrm{KL}}(P \mid\mid Q) \geq 0$ with equality iff $P = Q$. In terms of encodings, one interprets the result as formalizing the intuition that designing for distribution Q and evaluating on distribution P is sub-optimal in the number of bits used *unless* $P = Q$.

Remark 3.3.4. In many ways KL-divergence is a more fundamental measure of uncertainty than entropy. For example, on discrete spaces (as we've been discussing), one observes that the entropy $H(P)$ is the KL-divergence from P to the uniform distribution, i.e., $H(P) = D_{\mathrm{KL}}(P \mid\mid \text{Uniform})$.

Next, we formalize how uncertainty decreases as observations are made via the joint and conditional entropy.

Definition 18. Let $\mathcal{X}$ and $\mathcal{Y}$ be discrete random variables with joint distribution P. We define the **joint entropy** of $\mathcal{X}$ and $\mathcal{Y}$ as:

$$H(\mathcal{X}, \mathcal{Y}) \overset{\text{def}}{=} H(P) \tag{3.11}$$

The **conditional entropy** of $\mathcal{Y}$ given $\mathcal{X}$ is:

$$\begin{aligned} H(\mathcal{X} \mid \mathcal{Y}) &\overset{\text{def}}{=} H(\mathcal{X}, \mathcal{Y}) - H(\mathcal{Y}) \\ &= \mathop{\mathbb{E}}_{(x,y)\sim(\mathcal{X},\mathcal{Y})}[-\log \Pr(\mathcal{X} = x \mid \mathcal{Y} = y)] \end{aligned} \tag{3.12}$$

Remark 3.3.5. Consider a sequence of random variables, $\mathcal{X}_{1:T} \overset{\text{def}}{=} \mathcal{X}_1, \ldots, \mathcal{X}_T$. By manipulating (3.12) one derives a **chain rule** for constructing joint entropies from increasingly constrained conditional entropies:

$$H(\mathcal{X}_{1:T}) = \sum_{t=1}^{T-1} H(\mathcal{X}_{t+1} \mid \mathcal{X}_{1:t}). \tag{3.13}$$

Remark 3.3.6. Noting that $\mathcal{Y} = y$ is itself a random variable, one derives that:

$$H(\mathcal{X} \mid \mathcal{Y}) = \mathop{\mathbb{E}}_{y \sim \mathcal{Y}} H(\mathcal{X} \mid \mathcal{Y} = y). \tag{3.14}$$

Thus, conditioning by a random variable can be viewed as taking the convex combination of the entropies of all conditional distribution. Combined with the chain rule, this provides a straightforward means to recursively compute entropies. Note that this also implies that conditional entropy is always greater than or equal to zero. Applying Jensen's inequality, i.e. $f(\mathbb{E}[\mathcal{X}]) \leq \mathbb{E}[f(\mathcal{X})]$ whenever f is convex, one derives the desirable property that conditioning never increases entropy (uncertainty), i.e., for all triples of random variables, $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$,

$$H(\mathcal{X} \mid \mathcal{Y}) \geq H(\mathcal{X} \mid \mathcal{Y}, \mathcal{Z}) \geq 0. \tag{3.15}$$

Causal Entropy

When modeling temporally generated sequences of random variables such as paths, it is useful to introduce the notion of causal conditioning. Intuitively, and contrary to "non-causal" entropy, causal entropy does *not* condition on variables that have not been revealed, e.g., on events in the future. This makes causal entropy particularly well suited for measuring predictability in *sequential* decision making problems, as the agents cannot observe the future [162].

Definition 19. Let $\mathcal{X}_{1:T}$ and $\mathcal{Y}_{1:T}$ be two sequences of random variables. The probability of a sequence $\mathcal{X}_{1:T}$ **causally conditioned** on sequence $\mathcal{Y}_{1:T}$ is:

$$\Pr(\mathcal{X}_{1:T} \,\vdots\, \mathcal{Y}_{1:T}) \overset{\text{def}}{=} \prod_{t=1}^{T} \Pr(\mathcal{X}_t \mid \mathcal{Y}_{1:t}, \mathcal{X}_{1:t-1}) \tag{3.16}$$

The **causal entropy** of $\mathcal{X}_{1:T}$ given $\mathcal{Y}_{1:T}$ is defined as,

$$\begin{aligned} H(\mathcal{X}_{1:T} \,\vdots\, \mathcal{Y}_{1:T}) &\overset{\text{def}}{=} \sum_{t=1}^{T} H(X_t \mid X_{1:t-1}, Y_{1:t}) \\ &= \mathop{\mathbb{E}}_{Y_{1:T}, X_{1:T}} [-\log \Pr(Y_{1:T} \,\vdots\, X_{1:T})] \end{aligned} \tag{3.17}$$

Remark 3.3.7. Causal conditioning is often denoted using $||$. We opt to use $\,\mathrel{\dagger}\,$ to avoid confusion with relative entropy.

Remark 3.3.8. Recalling that conditioning only decreases uncertainty (3.15) and comparing the definition of causal entropy (3.17) with the entropy chain rule (3.13), one observes that causally conditioned entropy is lower bounded by statically conditioned entropy,

$$H(\mathcal{X}_{1:T} \mathrel{\dagger} \mathcal{Y}_{1:T}) \geq H(\mathcal{X}_{1:T} \mid \mathcal{Y}_{1:T}) \geq 0. \tag{3.18}$$

Interpreted as bits, the definition of causal entropy can be viewed as the description complexity of describing events as they occur. Static entropy is then the description length with hindsight. For example, when describing a race in real-time, one would describe the position of the participants given only their past positions. However, with knowledge of future positions, one would only need to describe when the ordering changes, which constrains the uncertainty of the time in-between changes.

To study the predictability of a given agent's policy, recall that a policy induces a Markov Chain, and thus a distribution over sequences of states and actions. Formally, given an MDP M and a policy π, let us denote by $\mathcal{A}_{1:i}$ and $\mathcal{S}_{1:i}$ random variable sequences actions and states respectively. The causal entropy of a policy for T-length paths is given by:

$$\begin{aligned} H(\mathcal{A}_{1:T} \mathrel{\dagger} \mathcal{S}_{1:T}) &= \sum_{t=1}^{T} H(\mathcal{A}_t \mid \mathcal{S}_{1:t}) \\ &= H(\mathcal{A}_{1:T-1} \mathrel{\dagger} \mathcal{S}_{1:T-1}) + H(\mathcal{A}_T \mid \mathcal{S}_{1:T}), \\ &= H(\mathcal{A}_{1:T-1} \mathrel{\dagger} \mathcal{S}_{1:T-1}) + \mathop{\mathbb{E}}_{s_{1:t}}[H(\mathcal{A}_T \mid \mathcal{S}_{1:T} = s_{1:T})], \end{aligned} \tag{3.19}$$

where the equalities follow from the Markov property, i.e., state transitions only depend on the current state and action and the policy only depends on the previous states. Matching intuition, the measure of uncertainty of the agent's actions given by (3.19) is bound by:

$$0 \leq H(\mathcal{A}_{1:T} \mathrel{\dagger} \mathcal{S}_{1:T}) \leq T \log(|A|),$$

with the maximum and minimum being realized by uniform policy and deterministic policies respectively.

Remark 3.3.9. The third equality in (3.19) provides a natural recursive algorithm for computing $H(\mathcal{A}_{1:T} \mathrel{\dagger} \mathcal{S}_{1:T})$ backwards in time.

Remark 3.3.10. The causal entropy of a policy can be made to apply to variable length paths by padding shorter paths with $\perp$ for the action and state. For example, a path of length 2 can be padded in to path of length 4 by:

$$\left(s_0 \xrightarrow{a_0} s_1\right) \mapsto \left(s_0 \xrightarrow{a_0} s_1 \xrightarrow{\perp} \perp \xrightarrow{\perp} \perp\right).$$

Using this padding, if $i > |\xi|$ then $H(\mathcal{A}_i \mid \mathcal{A}_{1:i-1}, \mathcal{S}_{1:i-1}, \mathcal{S}_i = \perp) = 0$, which means that the padding does not increase the uncertainty of the policy. Computationally, this means that the recursive algorithm can use the length of the path *or* being a complete path as the base case condition.

Remark 3.3.11. Unfortunately, the upper bound being tight (realized by uniform policies) implies that causal entropy tends to infinity as T does. Thus, in general, one cannot define causal entropy over all complete paths. Fortunately, for many "practical" dynamical systems, the first three moments: mean, variance, and skewness, of the path distribution length, for all policies, are bounded. For example, if there is a constant probability of the episode ending or if the episode ends when a battery, whose charge decrements with constant probability, runs out of charge. Using the padding trick, letting $\mathcal{T}$ denote the random variable for path length, and applying Markov's inequality extended to the third moment yields:

$$\Pr(\mathcal{T} \geq T) \leq \frac{\mathbb{E}[\mathcal{T}^3]}{T^3}. \tag{3.20}$$

Since the upper bound on the causal entropy only grows linearly in the horizon, the contribution of the causal entropy of complete paths with lengths greater than $\mathbb{E}[\mathcal{T}]$ attenuates at least quadratically. Thus, in such settings:

1. The infinite horizon causal entropy, i.e., the limit when taking T to ∞ is well defined and bounded. We refer to this limit as the **causal entropy of the policy**.
2. The causal entropy can be arbitrarily well approximated by using a sufficiently large planning horizon T.

Maximum Causal Entropy IRL

We are now ready to return to the problem of inverse reinforcement learning (IRL). Recall that the IRL problem states: given a set of demonstrations, find the reward that best "explains" the agent's behavior, where by "explain" one typically means that under the conjectured reward, the agent's behavior was approximately optimal. Further, recall that (i) many undesirable rewards satisfy this property, e.g.,

$$r : s \mapsto 0, \tag{3.21}$$

and (ii) the set of optimal policies can be infinitely large with individual policies predicting very different behavior. A popular solution to the lack of unique policy conundrum is to appeal to the **principle of maximum causal entropy** [163]. The principle of maximum causal entropy suggests forecasting using the policy whose action sequence, $A_{1:\mathcal{T}}$, has the highest causal entropy, conditioned on the state sequence, $S_{1:\mathcal{T}}$. That is, find the policy that maximizes:

$$H(A_{1:\mathcal{T}} \,\mathbin{\vdots}\, S_{1:\mathcal{T}}), \tag{3.22}$$

subject to policy feature, $\mathbf{f} : S \to \mathbb{R}^n$, constraint, $\mathbb{E}[\mathbf{f}(\mathcal{S})] = f^*$, e.g., maximize entropy subject to keeping the same average distance to other vehicles as seen in the data. Compared to all other distributions, this policy (i) ensures that (by definition) the agent's predicted policy does not depend on the future; (ii) is (by definition) consistent with observed feature statistics; (iii) minimizes the worst case prediction log-loss (see below).

Corollary 3.3.12 (Instance of Theorem 5.10 from [163])**.** Let n be a natural number, $f : S \to \mathbb{R}^n$ be a feature function, and P be a distribution of actions $\mathcal{A}_{1:T}$ and states $\mathcal{S}_{1:T}$. If P maximizes causal entropy subject to feature moments, $\mathbb{E}_P[\mathbf{f}(\mathcal{S})] = f^* \in \mathbb{R}^n$, then it minimizes the worst-case prediction log-loss:

$$\sup_{P'} \mathop{\mathbb{E}}_{(\mathcal{A}_{1:T}, \mathcal{S}_{1:T}) \sim P'} \log \frac{1}{P(\mathcal{A}_{1:T} \mid\mid \mathcal{S}_{1:T})}, \tag{3.23}$$

where (1) P' is causal, i.e., $P'(\mathcal{A}_{1:T}, \mathcal{S}_{1:T}) = P'(\mathcal{A}_{1:T} \mid\mid \mathcal{S}_{1:T})P'(\mathcal{S}_{1:T} \mid\mid A_{1:T-1})$, and (2) the feature moments, $\mathbb{E}_{P'}[\mathbf{f}(\mathcal{S})]$ are f^* when $\mathcal{S}$ is sequentially revealed from the known causally condition dynamics distribution, $\Pr(\mathcal{S}_{1:T} \mid\mid \mathcal{A}_{1:T})$.

In the language of description complexity and surprisal, one says that the maximum causal entropy distribution minimizes the worst case number of bits needed to describe the agent's behavior. Thus, when applied to Occam's razor style arguments, the principle of maximum causal entropy serves as a robust proxy for how hard it is to describe the agent's behavior.

Connecting back to reward motivated agents in MDPs, if $r(s)$ is assumed to be a linear combination of features, $r_\lambda(s) = \lambda \cdot \mathbf{f}(s)$, where $\lambda \in \mathbb{R}^n$, the principle of maximum causal entropy prescribes the following **entropy regularized** policy [163]:

$$\log \pi_\lambda(a \mid \xi) \overset{\text{def}}{=} V_\lambda(\xi \centerdot a) - V_\lambda(\xi) \tag{3.24}$$

where

$$V_\lambda(\xi) \overset{\text{def}}{=} \begin{cases} r_\lambda(\text{last}(\xi)) & \text{if } \xi \in \text{Paths}_\$, \\ \text{LSE}_{a \in A(\text{last}(\xi))} V_\lambda(\xi \centerdot a) & \text{if } \xi \in \text{Paths} \setminus \text{Paths}_\$, \\ r_\lambda(s) + \mathbb{E}_{s'}\left[V_\lambda(\xi \centerdot s') \mid s, a\right] & \text{if } (\xi = x \centerdot s \centerdot a) \wedge (s, a \in S \times A), \end{cases} \tag{3.25}$$

and λ is such that (3.25) results in a policy which matches feature demonstrations.

Remark 3.3.13. Our exposition of the maximum causal entropy policy is non-standard in that it is defined in terms of paths. Typically, V is split into two functions, V and Q, called the state value and action value respectively. As the names suggest, V and Q are defined in terms of (time-indexed) states and actions. V corresponds to the first two cases of (3.25) and Q corresponds to the last case. As we shall see, when the reward is unknown and history dependent, we will often find it useful to refer to the value of arbitrary prefixes of a path. Thus, to simplify future notation, unless explicitly required, we shall conflate the two value functions.

Remark 3.3.14. Replacing LSE with max in (3.25) yields the standard Bellman Backup [16] used to compute the optimal policy in tabular reinforcement learning. For this reason, (3.25) is called the soft-Bellman backup.

Remark 3.3.15. It can be shown that maximizing causal entropy corresponds to believing that the agent is exponentially biased towards high reward policies [163]:

$$\Pr(\pi_\lambda \mid M) \propto \exp\left(\mathop{\mathbb{E}}_{\xi}\left[R_\lambda(\xi) \mid \pi_\lambda, M\right]\right). \tag{3.26}$$

Remark 3.3.16. In the special case of scalar state features, $\mathbf{f} : S \to \mathbb{R}_{\geq 0}$, the maximum causal entropy policy (3.25) becomes increasingly optimal as $\lambda \in \mathbb{R}$ increases (since LSE monotonically approaches max). In this setting, we shall refer to λ as the agent's **rationality coefficient**. As we will revisit when discussing stochastic games (Ch 7), the rationality coefficient casts the smooth Bellman backup as a multi-objective optimization:

$$J(\pi) = \mathbb{E}\left[R(\mathcal{S}_{1:T}) \mid \pi, M\right] + \lambda \cdot H(\mathcal{A}_{1:T} \mathrel{\Vert} \mathcal{S}_{1:T}, \pi, M),$$

where one trades performance for randomness.

3.4 Bibliographic Notes

In this chapter, we considered the problem of predicting the actions of purposeful agents acting in Markov Decision Processes. The problem of learning objectives by observing an expert has a rich and well developed literature dating back to early work on Inverse Optimal Control [82] and more recently via Inverse Reinforcement Learning (IRL) [111]. In IRL, an expert demonstrator optimizes an *unknown* reward function by acting in a stochastic environment, i.e., a Markov Decision Process. The goal of IRL is to find a reward function that explains the agent's behavior. A fruitful approach has been to cast IRL as a Bayesian inference problem to predict the most probable reward function [123]. To make this inference robust to demonstration/modeling noise, one commonly appeals to the principle of maximum causal entropy [79, 163]. Intuitively, this results in a forecasting model that is no more committed to any given action than the data requires (formalized as bounding the worst-case expected description length of future demonstrations). Furthermore, in the case of linear rewards, there exist several efficient gradient based algorithms for finding the most likely objective [163]. In this chapter, we largely ignored these algorithms since they are not applicable to our ultimate goal of learning discrete concepts.

Finally, the description length being worst-case minimal means that the principle of maximum causal entropy provides a succinct encoding of the demonstrations. This enables discussing the total description length of both the objective (drawn from some representation class) and the path. Viewed this way, our inference problem can be cast as a data compression problem [3] in which one first describes which encoding to use (by describing the objective) and then describes the demonstrations using this encoding. This provides a direct way to discuss the trade-off between how well a given objective explains the behavior of the agent and how hard the objective is to describe.

Part I

Learning and Teaching

Demonstrations are a popular and often ergonomic way to partially specify high-level tasks. However, most methods for communicating via demonstrations either (i) do not provide guarantees that the learned artifacts can be safely composed; (ii) do not explicitly capture temporal properties; or (iii) intimately link the encoding of the task with details of the workspace.

Motivated by this deficit, Ch 4 advocates for learning Boolean *task specifications*, a class of sparse Boolean non-Markovian rewards which admit well-defined composition, explicitly handle historical dependencies, and are robust to certain classes of "reward hacking bugs." As we shall see, the connection with sparse rewards sketches a path for adapting for leveraging the literature on maximum causal entropy inverse reinforcement learning techniques to learn task specifications.

Unfortunately, three features make learning temporal task specifications difficult: (1) the (countably) infinite number of tasks under consideration; (2) an a-priori ignorance of what memory is needed to encode the task; and (3) the lack of gradients to guide the search for explanatory tasks. In Ch 5 we explore these issues and derive a family of approximate algorithms that systematically reduce the problem of learning from demonstrations into a series of supervised learning problems.

Finally, in Ch 6, we change perspective from a learner to a teacher. This perspective has three motivations. First, formal task specifications can be difficult to understand, even for domain experts. Thus, being able to supplement formal or natural language descriptions of a task with carefully selected demonstrations can be invaluable. Second, alternating teaching and learning forms a foundation for future work on organic team-based collaboration, where team members automatically specialize given inferred assumptions about other agents. Finally, teaching provides further academic insight into what kind of demonstrations are helpful for learning.

Chapter 4

Specifications from Demonstrations

Words may show a man's wit, but actions his meaning.

Benjamin Franklin (Scientist, Political Philosopher 1706-1790)

When learning from demonstrations, one typically models the demonstrator (e.g., a human expert) as episodically operating within a Markov Decision Process where the demonstrator's goals are expressed as a function of the state. However, even if the dynamics are Markovian, many problems are naturally modeled in non-Markovian terms.

Example 4.0.1. *Consider the task illustrated in Figure 4.1. In this task, the agent moves in a discrete gridworld and can take actions to move in the cardinal directions (north, south, east, west). Further, the agent can sense abstract features of the domain represented as colors. The task is to reach any of the yellow (recharge) tiles without touching a red (lava) tile. Additionally, if a blue (water) tile is stepped on, the agent must step on a brown (drying) tile before going to a yellow tile. The last constraint requires recall of two state bits of history (and is thus not Markovian): one bit for whether the robot is wet and another bit encoding if the robot recharged while wet.*

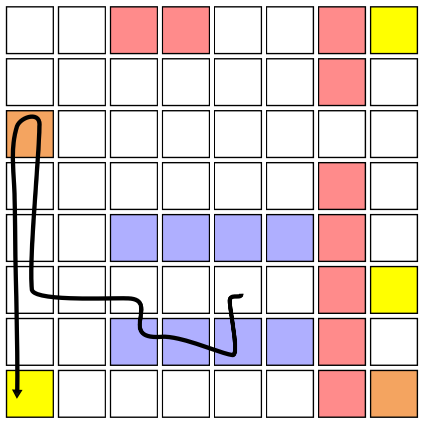


Figure 4.1: Example of agent entering the water and needing to dry off before recharging.

Further, like Ex 4.0.1, many tasks naturally decomposed into several sub-tasks. This work aims to address the question of how to systematically and separately learn non-Markovian tasks such that they can be readily and safely recomposed into the larger meta-task.

A case for Boolean task specifications

Here, we argue that *non-Markovian Boolean specifications* provide a powerful, flexible, and easily transferable formalism for task representations when learning from demonstrations. This stands in contrast to the quantitative scalar reward functions commonly associated with Markov Decision Processes. Focusing on Boolean specifications has certain benefits: (1) The ability to naturally express tasks with *temporal dependencies*; (2) the ability to take advantage of the *compositionality* present in many problems; and (3) use of *formal methods* for planning and verification [130].

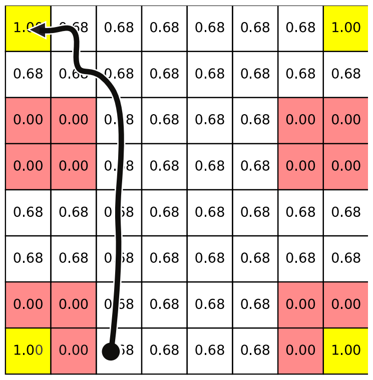
Although standard quantitative scalar reward functions could be used to learn this task from demonstrations, three issues arise. First, consider the problem of *temporal tasks*: reward functions are typically Markovian, so requirements like those in Ex 4.0.1 cannot be directly expressed in the task representation. One could explicitly encode time into a state and reduce the problem to learning a Markovian reward on new time-dependent dynamics; however, in general, such a reduction suffers from an exponential blow up in the state size (commonly known as the *curse of history* [117]). When inferring tasks from demonstrations, where different hypotheses may have different historical dependencies, naïvely encoding the entire history quickly becomes intractable.

A second limitation relates to the *compositionality* of task representations. For instance, Ex 4.0.1 naturally decomposes into three sub-tasks. Ideally, we would want an algorithm that could learn each sub-task and combine them into the complete task, rather than only be able to learn single monolithic tasks. However, for many classes of quantitative rewards, combining rewards remains an ad-hoc procedure. The situation is further exacerbated by humans being notoriously bad at anticipating or identifying when quantitative rewards will lead to unintended consequences [72], which poses a serious problem for AI safety [10] and has led to investigations into reward repair [58]. For instance, we could take a linear combination of rewards for each of the subtasks in Ex 4.0.1, but depending on the relative scales of the rewards, and temporal discount rate, wildly different behaviors would result.

In fact, the third limitation - brittleness due to simple changes in the environment - illustrates that often, the correctness of the agent can change due to a simple change in the environment. Recall the gridworld example from the introduction. In particular, imagine learning a reward that encodes the "recharge while avoid lava" task in Ex 4.0.1. Fig 4.2a illustrates a reward resulting from performing Maximum Entropy Inverse Reinforcement Learning [164] with binary features: red (lava tile) and yellow (recharge tile). As is easy to verify, a reward optimizing agent, $\sum_{i=0}^{\infty} \gamma^i r_i(s)$, with a discount factor of $\gamma = 0.69$ would generate the trajectory shown in Fig 4.2a which avoids lava and eventually recharges.

Unfortunately, using the *same* reward and discount factor on a nearly identical world can result in the agent entering the lava. For example, Fig 4.2b illustrates the learned reward being applied to a change in the gridworld where the top left charging tile has been removed. An acceptable trajectory is indicated via a dashed arrow. Observe that in this new workspace,

[0]As we shall see later, this discount rate is equivilent to assuming at every timestep that the episode will end with probability 1 - 0.69.


(a)

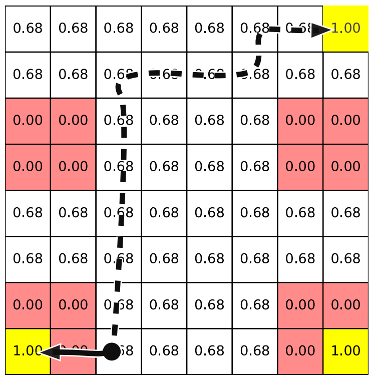

(b)

Figure 4.2: Illustration of a bug in the learnt quantitative Markovian reward resulting from slight changes in the environment.

the discounted sum of rewards is maximized on the solid arrow's path, resulting in the agent entering the lava! While it is possible to find new discount factors to avoid this behavior, such a supervised process would go against the spirit of automatically learning the task.

Finally, we briefly remark that while non-Markovian Boolean rewards cannot encode all possible rewards, e.g., "run as fast as possible", often times such objectives emerge from a Boolean tasks. For example, consider modeling a race. If at each time step there is a non-zero probability of entering a losing state, the agent will run forward as fast as possible even for the Boolean task "win the race".

In summary, quantitative Markovian rewards are limited as a task representation when learning tasks containing *temporal specifications* or *compositionality* from demonstrations [2]. Moreover, the need to fine tune learned tasks with such properties seemingly undercuts the original purpose of learning task representations that are generalizable and invariant to irrelevant aspects of a task [98].

Contributions

Before proceeding, we briefly outline the contributions present in this chapter. First and foremost, this chapter proposes representing tasks as Boolean specifications and then uses the principle of maximum causal entropy to formulate learning specification from demonstration as maximum a posteriori (MAP) inference. To facilitate this, we review a number of (folk) modeling tricks that illustrate the expressiveness of mixing task specifications and MDPs. We end the chapter with an analysis of the special case in which the entropy of the agent's actions causally conditioned on the revealed states coincides with the entropy of the agents actions conditioned on all states (past and future).

4.1 Task Specifications

In this chapter, we highlight the various adaptations to the machinery developed in Ch 2 and 3 to handle Boolean task specifications (defined below).

Definition 20. Let M be an MDP and let $\mathcal{R}$ be a representation class for sets of complete paths, $\text{Paths}_\$$, in M, i.e., for all $\varphi \in \mathcal{R}$,

$$\text{concept}(\varphi) \subseteq \text{Paths}_\$. \tag{4.1}$$

A representation φ from $\mathcal{R}$ is known as a Boolean **task specification**.

Remark 4.1.1. When clear from context, we will often shorten Boolean task specification to simply task specification, specification, or even task.

Remark 4.1.2. We will generally assume that task specifications are closed under Boolean operations. For example, given two task specifications φ, φ' we will assume that their exist specifications denoted $(\varphi \wedge \varphi')$ and $\neg\varphi$ such that:

$$\text{concept}(\neg\varphi) = \text{Paths}_\$ \setminus \text{concept}(\varphi) \qquad \text{concept}(\varphi \wedge \varphi') = \text{concept}(\varphi) \cap \text{concept}(\varphi'). \tag{4.2}$$

In this way, we can discuss trying to avoid performing a task and simultaneously performing multiple tasks. Furthermore, since conjunction and negation are sufficient to realize any Boolean circuit, one can represent performing at least one task in a list of tasks or even assume-guarantee tasks, e.g., assuming that the other car will not enter the intersection, cross the road.

Featurized task specifications

In order to make a task applicable to a wide number of workspaces, and in analogy to featurized rewards, one often defines a task in terms of features.

Definition 21. Let Σ be an alphabet. A **feature** function (or observation function) is a prefix preserving map $f : \text{Paths} \to \Sigma^*$, i.e., if $\xi \in \text{Paths}$ is a prefix of $\xi' \in \text{Paths}$ then $f(\xi)$ is a prefix of $f(\xi')$. The image of f restricted to φ, denoted $\widehat{\varphi}$, is called an **abstract task specification**.

We say then that a (abstract) task specification, φ, is regular if φ (or $\widehat{\varphi}$) is a regular language, and thus recognized by a finite state machine.

Example 4.1.3. *Let us again return to the MDP shown in Fig 3.1. Let $f : Paths \to \{0,1\}^*$ be the feature map, $f : s_i \mapsto i \pmod 2$. The parity language, $\sum_{\sigma \in f(\xi)} \sigma \pmod 2 = 0$, is an abstract task specification, $\widehat{\varphi}$. Further, because the parity language recognized by the DFA shown in Fig 2.1, $\widehat{\varphi}$ is regular.*

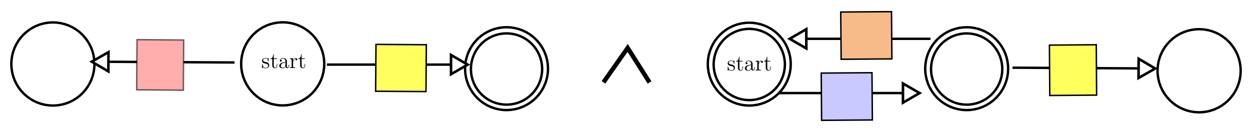


Figure 4.3: Decomposition of gridworld task into two DFAs.

Example 4.1.4. *We formalize the abstract task specification of our gridworld example. Let* $\Sigma = \{$🟥, 🟦, 🟨, 🟧, □$\}$ *denote an alphabet. The task "go to yellow tiles, avoid red tiles, and visiting a blue tile means you need to visit a brown tile before a yellow tile." can be represented by the conjunction of the two DFAs shown in DFA over* Σ *shown in Fig 4.3.*

Remark 4.1.5. Tasks defined on automata admit various temporal compositions such as performing two tasks in sequence or performing a particular sub-task whenever an event is trigger. An instance of the latter is shown in our gridworld example where touching a blue tile triggers a sub-task: Go to a drying tile.

4.2 Task conditioned behavior prediction

The machinery developed in Ch 3 was targeted at predicting the behavior of agents driven to optimize the sum of state-based rewards. Note that this same machinery can be adapted to task specifications by abusing notation and interpreting task specifications as sparse binary rewards, i.e.,

$$\varphi : \text{Paths} \to \{0, 1\}, \tag{4.3}$$

where $\varphi(\xi) = [\xi \in \varphi]$ if ξ is complete and 0 otherwise. If φ is represented by an automata, then (4.3) can be made Markovian by augmenting the MDP states with the automata state.

Next, observe that since task specifications only correspond to complete paths, they are necessarily prefix-free. Thus, given a policy π, and MDP M, the probability $\Pr(\xi \in \varphi \mid \pi, M)$ is a well-defined (3.5). This enables defining the competency of an agent.

Definition 22. Given a fixed MDP, M, a policy, π, is (p, φ)-*competent* if

$$\Pr(\xi \in \varphi \mid \pi, M) = p, \tag{4.4}$$

where p is called the **satisfaction probability** of π.

Using the above terminology, we can express maximum a posteriori (MAP) inference problem we wish to study as follows:

Task Inference from Demonstrations Problem: Let a M, $\mathcal{R}$, and P be a fixed MDP, task specification representation class, and task prior, respectively. Further, let π^* be a (p^*, φ^*)-competent policy, π^*, where p^*, φ^*, and π^* are all unknown. Given a multi-set of i.i.d. demonstrations, $\xi_1^*, \ldots \xi_m^* \sim (\pi^*, M)$, find:

$$\varphi \in \arg\max_{\psi \in \mathcal{R}} \Pr(\xi_1^*, \ldots \xi_m^* \mid \psi, M) \cdot P(\psi \mid M). \tag{4.5}$$

Of course, by itself, the above inference is ill-posed as $\Pr(\xi_1^*, \ldots \xi_m^* \mid M, \varphi)$ is left undefined. As we discussed in Ch 3, a popular, and in practice effective, solution to this conundrum is to appeal to the principle of maximum causal entropy.

Maximum entropy planners

Concretely, since π^* is unknown, we shall use a proxy for π^* the maximum causal entropy policy that is (p^*, M)-competent. Unfortunately, the competency of the agent is also assumed to be unknown. Here we have two options. Namely, the competency of the agent can be treated as a hyper-parameter or estimated empirically, e.g.,

$$p^* \approx \frac{1}{m}\sum_{i=1}^{m} \varphi(\xi).$$

The former is useful when given only a few demonstrations and the latter is useful when given a large number of demonstrations. In either case, the main point is that using a competency estimator and the principle of maximum causal entropy defines a mapping from tasks to policies (and thus likelihoods).

$$\text{maxEntPlanner} : \varphi \mapsto \pi_\varphi, \tag{4.6}$$

where π_φ belongs to the family of (entropy regularized) policies $\pi_\lambda(\xi \mid a) \propto \exp\Big(V_\lambda(\xi \centerdot a)\Big)$ defined by the smooth Bellman backup (3.25):

$$\begin{gathered}\ln \pi(a \mid \xi) \overset{\text{def}}{=} V_\lambda(\xi \centerdot a) - V_\lambda(\xi) \\ V_\lambda(\xi) \overset{\text{def}}{=} \begin{cases} \lambda \cdot \varphi(\xi) & \text{if } \xi \in \text{Paths}_\$, \\ \text{LSE}_{a \in A(\text{last}(\xi))} V_\lambda(\xi \centerdot a) & \text{if } \xi \in \text{Paths} \setminus \text{Paths}_\$, \\ \mathbb{E}_{s'}\left[V_\lambda(\xi \centerdot s') \mid s, a\right] & \text{if } (\xi = x \centerdot s \centerdot a) \wedge (s, a \in S \times A). \end{cases}\end{gathered} \tag{4.7}$$

Note that we have simplifed (3.25) by using fact that $\varphi(\xi)$ is zero for all but complete paths. To associate λ with p^*, observe that the satisfaction probability of π_λ is monotonic is λ. Thus, one can find an appropriate λ to match p^* via a bracketed search, e.g. bisection [1].

[1]The upper bound of the bracketed range may need to also be discovered by repeated doubling. See Ch 7 for details on realizing such planners.

When possible, we will use the maximum causal entropy planner black-box and simply refer to the policy π_φ. The corresponding value function will be denoted V_φ.

We shall postpone further discussions of implementing maxEntPlanner until Part II.

Solution Sketch

Maximum causal entropy policy, π_φ, in hand, we sketch our high-level strategy for learning tasks from demonstrations:

1. Select a candidate task, φ.
2. Compute the prior probability, $\Pr(\varphi \mid M)$.
3. Compute the likelihood of the demonstrations given the task:

$$\Pr(\xi_1^*, \ldots \xi_m^* \mid \varphi, M).$$

4. Score tasks based on the posterior probability.
5. Return the maximum reward a posteriori.

What remains to realize this scheme is derive a way to select candidates with increasing posterior probability. Unfortunately, three features make efficent inference using this algorithm difficult: (1) the (countably) infinite number of tasks under consideration; (2) an a-priori ignorance of what memory is needed to encode the task; and (3) the lack of gradients to guide the search for explanatory tasks. We shall postpone addressing these problems until Ch 5. Instead, for the remainder of the chapter, we shall address the choice of binary co-domain for φ (Sec 4.2), various useful modeling tricks to make task specifications more expressive (Sec 4.3), and simplifications for the special case when maximum causal entropy coincides with maximum entropy (Sec 4.4).

Choice of binary co-domain

One might wonder how sensitive this formulation to changing the co-domain of φ from $\{0, 1\}$ to any other real values, i.e.,

$$\varphi' : \text{Paths} \to \{a, b\}.$$

We briefly remark that, subject to some mild technical assumptions, almost any two real values could be used for φ's co-domain. Namely, observe that unless both a and b are zero, the expected satisfaction probability, p, is in one-to-one correspondence with the expected value of φ', i.e.,

$$\mathbb{E}[\varphi'] = a \cdot p + b \cdot (1 - p).$$

Thus, if a policy is feature matching for φ, it must be feature matching for φ' (and vice-versa). Therefore, the space of consistent policies is invariant under such transformations. Finally, because the space of policies is unchanged, the maximum causal entropy policies must remain unchanged. In practice, we prefer the use of $\{0, 1\}$ as the co-domain for φ since it often simplifies many calculations.

4.3 Task Specification Modeling Tricks†

At first, restricting one's attention to task specifications (which can only take on binary values) may seem limiting. Note however that the ability to explicitly model history, e.g., in DFA states, and the probabilistic nature of the underlying MDPs enable the emergence of a number of *quantitative* properties. These include temporal discounting, preferences to maintain distances to obstacles, and minimizing violations of a specification. Thus, task specifications offer a rich modeling formalism that inherits many of the strengths (and problems) of the Markovian reward setting.

Discount Rates and Shortest Paths

In many settings, it is natural to assume that agents have a bias towards performing tasks efficiently, e.g., taking the shortest path between the current location and a goal. A common encoding of such preferences assumes the agent optimizes a discounted sum of rewards:

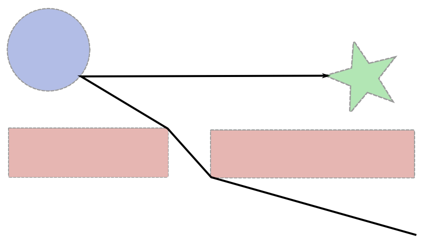
$$\mathop{\mathbb{E}}_{\xi}\left[\sum_{s_k \in \xi} \gamma^k \cdot r(s_k) \mid \pi, M\right], \tag{4.8}$$

where $\gamma \in [0, 1)$. This discounted sum is often transformed into a non-discounted objective by observing that (4.8) is equivalent to assuming that there is a special terminal state, \$, which receives no reward, i.e., $r(\$) = 0$, and that applying an action from any non-terminal state transitions to \$ with probability $1 - \gamma$. Thus, at any time step the expected future reward attenuates by γ as desired.

The standard interpretation is that the agent operates in a MDP in which the episode may end with probability $1 - \gamma$ at any time point. The result is that the agent prefers to complete its task as quickly as possible, with an emergent preference for shorter paths. This perspective shows the planning with a discounted reward implies planning assuming the path lengths follow a geometric distribution with mean $1/\gamma$.

Batteries

There are of course may contexts in which one wishes to encode some form of temporal discounting, but the path length distribution is decidedly not geometric. One such example is that of a robot operating with a stochastically depleting battery [69]. Specifically, if sink \$ is transitioned to when the battery reaches 0 and the battery level either decreases or stays


(a) Optimal path in agent model.

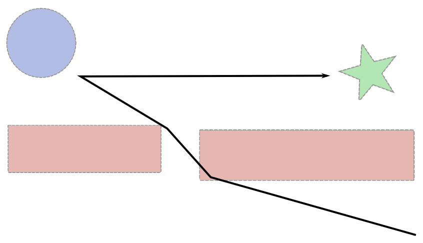

(b) Model mismatch leads to task failure.

Figure 4.4

the same with constant probability, i.e.,

$$\Pr(\text{battery\_level}(s') = \Pr(k' \mid \text{battery\_level(s)} = k) = \begin{cases} 1 & \text{if } k = k' = 0, \\ \gamma & \text{if } k = k' \neq 0, \\ 1-\gamma & \text{if } 0 \leq k - k' \leq 1, \\ 0 & \text{otherwise,} \end{cases} \tag{4.9}$$

then the path lengths follow a negative binomial distribution. The geometric distribution above is then a special case when the initial battery level is 1. For larger battery levels, one sees that the above model differs in that the probability of the episode ending after the current action is 0, allowing the agent to plan more optimistically.

Remark 4.3.1. One of the benefits of assuming the initial battery level is 1, is that the internal battery state need not be observed. For larger battery levels, this necessarily imposes a larger assumption or the need to estimate the battery level from other observations, e.g., if the agent is taking a short, but risky, path to the goal vs taking a long, but safe, path the goal.

Slipping and Obstacle Padding

The last two modeling tricks concerned modifying the way an agent accounts for time when planning. For variety, the next trick modifies the way an agent accounts for the distance between obstacles (or goals) and itself. In particular, there is often a modeling gap between the workspace, the agent's model of the workspace, and the model of the workspace used for predicting the agent's behavior. A common examples are distances between objects and transition probabilities. Such disconnects can lead to an agent just barely missing (reaching) an obstacle (goal) in one model, but not the others! This phenomenon is made worse by temporal discounting where an agent's objective often prefers shorter paths!

Example 4.3.2. *Consider the agent operating in work space shown in Fig 4.4a. This agent agent attempts to perform the task: "Visit the circular region and then the star region while avoiding the rectangular regions." Assuming temporal discounting, the agent may try to take the shortest path to achieve the task. This results in path that barely avoids the red region* *and skims the goal regions. Unfortunately, the true workspace contains slightly different obstacle distances, and thus the agent's plan would lead to missing the targets and entering the rectangular region.*

The classic (conservative) solution to this problem is to "bloat" (shrink) each obstacle (goal) by some amount and plan in the new model. While sufficient for the shown example, and perhaps even a good idea for safe control, in the case of inference this leads to very little probability mass assigned to the original optimal path. The result is an undesirably biased prediction of the agent's behavior. While some of these issues are papered over using entropy regularized planning, this structural bias remains. For example, if the goal is only reachable using a narrow corridor, bloating may remove this corridor, leading to a trivially false specification!

Instead, one might seek a way to encourage the agent to keep its distance from the obstacle. This can be done by adding in a probability of slipping. Thus, rather than bloating the size of each obstacle, one induces a distribution over the location a given action may lead you to. The result is an agent that acts as if obstacles are effectively padding, but is willing to risk entering narrow corridors if no other options are available.

Stochastic Observations

Unfortunately, while useful for modeling a preference to maintain distance from obstacles, modeling slipping does not address the incentive for an agent to skim a goal region. Framed another way, the paths are not robust to uncertainty in the goal regions. A lightweight[2]trick to model this uncertainty is to make the observation map stochastic, i.e.,

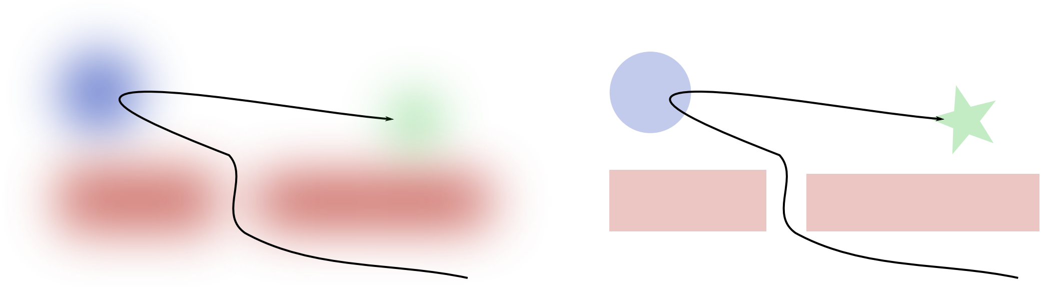
$$f : \text{Paths} \to \mathit{Distr}(\Sigma^*), \tag{4.10}$$

where Σ indicates whether the agent left of entered an region and the probability of observing a change increases as the agent gets closer to the center of the obstacle and decreases given the previous observation.

Example 4.3.3. *Consider the reach avoid task from Example 4.3.2, but using a noisy observation function. here the probability of observing a enter region token is weighted by the relative intensities shown in Fig 4.5a. An optimal path would both keep its distance from the rectangular obstacles and approach the centers of the goal regions.*

Remark 4.3.4. Note that stochastically modeling the entering/leaving of a region as above does not actually require a stochastic observation function. Namely, augmenting the MDP state with the previous observation and whether or not it belongs to the region is sufficient to create an equivalent MDP with a deterministic observation. This is ultimately what makes this approach tractable compared to the general partially observable MDP setting.

[2]The "heavyweight" alternative being moving to partial-observed Markov decision processes, which while expressive are in general intractable (or even undecidable).



(a) Optimal path using noisy observations. **(b)** Path robustly performs reach-avoid.

Figure 4.5

Minimizing Violations and Redemption

Stochastic observations as implemented above also offer a resolution to the question of how to model situations where the goal is to minimize "violations" of a given specification. To illustrate, consider again the sequential reach avoid task from Example 4.3.2. Now, suppose we observe the path of agent, but not the set of observations the agent received. For example, imagine we see the agent very briefly enter the rectangular avoid region, and then continue on to the two goals. Noting that the task is *unsatisfiable* after entering the rectangular region, one is forced to conclude that the agent must not have observed entering the rectangular region. That is, after marginalizing over the possible outcomes, the agent's behavior is still probable given the task specification. The result is an agent model that minimizes the "violations" to the specification.

Remark 4.3.5. Modeling the reach-avoid task as a DFA, an alternative variant on this modeling trick is turn the DFA into a MDP. Thus, when the token to be avoided is seen, there can be some probability that the new transition system does not transition to the failing sink state. While a useful mental model, for simplicity, we will avoid this perspective in this thesis and assume the stochastic observations are built into the underlying MDP.

4.4 Maximum Entropy Special Case†

In this section, we study the special case when maximum causal entropy coincides with maximum entropy, i.e.,

$$H(\mathcal{A}_{1:T} \,\|\, \mathcal{S}_{1:T}) \approx H(\mathcal{A}_{1:T} \mid \mathcal{S}_{1:T}). \tag{4.11}$$

This occurs whenever the agent's policy does not (typically) depend on the revealed state outcome, e.g., deterministic systems where the revealed state is pre-determined by the action.

To begin, let $M = \langle S, s_0, A, P \rangle$ be a fixed MDP where applying a fixed action sequence yields the same state sequence with high probability. That is, in M causally conditioning on a state sequence is approximately the same static conditioning (4.11). Thus, action sequences effectively determine their path and thus act as policies.

Next, recall that maximizing causal entropy subject to the expected reward corresponds to believing that the agent is exponentially biased towards high reward (3.26). Substituting the satisfaction probability for expected reward and paths for policies (due to effective determinism), we have:

$$\Pr(\xi \mid M, \varphi, \lambda) \approx \frac{e^{\lambda \cdot \varphi(\xi)}}{Z_\varphi}, \tag{4.12}$$

where Z_φ is a normalizing coefficient and λ is again tuned to match observed competency p^*. Perhaps surprisingly, under (4.12) we find that the Boolean nature of φ forces a simple form for the path likelihoods.

Proposition 4.4.1. Let p_φ denote the competency of π_φ and let q_φ denote the competency of the policy that selects actions uniformly at random. If maximum causal entropy coincides with maximum entropy (4.11), then the likelihood of a path ξ under π_φ is:

$$\Pr(\xi \mid \pi_\varphi, M) \propto \begin{cases} p_\varphi / q_\varphi & \text{if } \xi \in \varphi \\ (1 - p_\varphi)/(1 - q_\varphi) & \text{if } \xi \notin \varphi. \end{cases} \tag{4.13}$$

Proof. For brevity, we will find it helpful to define the probability mass of just the state transitions by defining:

$$w_\xi \overset{\text{def}}{=} \prod_{i=0}^{\tau-1} P(s_{i+1} \mid s_i, a_i) \qquad W_\varphi \overset{\text{def}}{=} \sum_{\xi \in \varphi} w_\xi, \tag{4.14}$$

where $\xi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots \xrightarrow{a_n} s_n$. Further, define $c \overset{\text{def}}{=} e^{\lambda_\varphi}$. Next, observe that by assumption

$$\begin{aligned} Z_\varphi \cdot p_\varphi &= 1 \cdot \sum_{\xi \in \varphi} c^1 \cdot w_\xi + 0 \cdot \sum_{\xi \notin \varphi} c^0 \cdot w_\xi = cW_\varphi \\ Z_\varphi &= c^1 \sum_{\xi \in \varphi} w_\xi + c^0 \sum_{\xi \notin \varphi} w_\xi = cW_\varphi + W_{\neg\varphi} \end{aligned} \tag{4.15}$$

Combining gives $Z_\varphi = W_{\neg\varphi}/(1 - p_\varphi)$. Next, observe that if $\xi \notin \varphi$, then $e^{\lambda_\varphi \varphi(\xi)} = 1$ and substituting in (4.12) yields,

$$\Pr(\xi \mid \pi_\varphi, M, \xi \notin \varphi) = w_\xi \cdot \frac{1 - p_\varphi}{W_{\neg\varphi}}.$$

If $\xi \in \varphi$ (implying $W_\varphi \neq 0$) then $e^{\lambda_\varphi} = Z_\varphi p_\varphi / W_\varphi$ and

$$\Pr(\xi \mid \pi_\varphi, M, \xi \in \varphi) = w_\xi \cdot \frac{p_\varphi}{W_\varphi}.$$

Finally, observe that $q_\varphi = W_\varphi / W_{\text{true}}$. Substituting and factoring yields (4.13). □

Remark 4.4.2. If the dynamics were deterministic, then q_φ would be proportional to $|\text{concept}(\varphi)|$. In this case, Prop 4.4.1 exactly coincides with the size principle from concept learning [143], where the likely of drawing an element from a concept is inversely proportional to its cardinality (size).

Remark 4.4.3. Together, Prop 4.4.1 and (4.12) imply that if (4.11) holds, then:

$$\lambda = \ln \frac{p_\varphi}{1-p_\varphi} - \ln \frac{q_\varphi}{1-q_\varphi} = \text{logit}(p_\varphi) - \text{logit}(q_\varphi), \tag{4.16}$$

where $\text{logit}(p) = \ln \frac{p}{1-p}$. Thus, π_φ is entirely determined by the assumed competency p_φ and the random action competency q_φ.

Likelihood of multiple demonstrations

If the teacher gives a sequence demonstrations, $\xi_1^*, \ldots \xi_m^*$, drawn i.i.d. from (4.13), then the log likelihood, of $\xi_1^*, \ldots \xi_m^*$ under (4.13) is:[3]

$$\ln \Pr(\xi_1^*, \ldots \xi_m^* \mid \pi_\varphi, M) = N_\varphi \ln \left(\frac{p_\varphi}{q_\varphi}\right) + (m - N_\varphi) \ln \left(\frac{1-p_\varphi}{1-q_\varphi}\right) + C \tag{4.17}$$

where $N_\varphi \overset{\text{def}}{=} \sum\limits_{\xi \in \xi_1^*, \ldots \xi_m^*} \varphi(\xi)$, we take $(0 \cdot \ln(\ldots) = 0)$, and C is a constant independent of φ. Assuming $\xi_1^*, \ldots \xi_m^*$ is "representative" such that $N_\varphi \approx p_\varphi \cdot |\xi_1^*, \ldots \xi_m^*|$, we can (up to a φ independent normalization) approximate (4.17) as:

$$\Pr(\xi_1^*, \ldots \xi_m^* \mid \pi_\varphi, M) \underset{\sim}{\propto} \exp\Big(m \cdot D_{\text{KL}}\big(p_\varphi \parallel q_\varphi\big)\Big), \tag{4.18}$$

where $D_{\text{KL}}\big(p \parallel q\big) \overset{\text{def}}{=} p \ln \frac{p}{q} + (1-p) \ln \frac{1-p}{1-q}$ denotes the information gain (KL divergence) between two Bernoulli distributions with means p and q.

Unfortunately, the approximation $|\xi_1^*, \ldots \xi_m^*| \cdot p_\varphi \approx N_\phi$ implies that, $p_{\neg\varphi} = 1 - p_\varphi$ which introduces the undesirable symmetry, $\Pr(\xi_1^*, \ldots \xi_m^* \mid M, \pi_\varphi) = \Pr(\xi_1^*, \ldots \xi_m^* \mid M, \pi_{\neg\varphi})$. To break this symmetry, we assert that the demonstrator must be at least as good as random. This leads to the approximate path distribution:

$$\Pr(\xi_1^*, \ldots \xi_m^* \mid \pi_\varphi, M) \underset{\sim}{\propto} \mathbb{1}[p_\varphi \geq q_\varphi] \cdot \exp\Big(m \cdot D_{\text{KL}}\big(p_\varphi \parallel q_\varphi\big)\Big), \tag{4.19}$$

Remark 4.4.4. Employing Sanov's Theorem, (4.19) can be interpreted as quantifying the atypicality of demos over random action hypothesis [42]. Thus, the we seek to find tasks where the demonstrations seem "typical."

[3]We have suppressed the multinomial coefficient required if any two demonstrations are the same. However, this term will not change as φ varies, and thus cancels when comparing across specifications.

Remark 4.4.5. This special case illustrates that relying just on the likelihood distribution will result in overfitting to the data. In particular, consider two task specifications, $\varphi \subset \varphi'$ that are consistent with the same demonstrations. Since φ is a subset of φ', it must be that $q_\varphi < q_{\varphi'}$. Thus, assuming the same competency for for φ and φ' the KL-divergence in (4.19) must decrease when adding the missing paths from φ' to φ.

This effect can be mitigated in several ways. For example, we shall employ a prior that exponentially favors tasks with small size. Furthermore, note that moving to domains in which causal entropy does not coincide with static entropy helps mitigate this overfitting. Intuitively, for stochastic domains, the MAP must balance making actions of the agent seem like the only good moves and making the actions not seem risky. The former requires removing paths from φ where as the latter requires adding paths to φ.

4.5 Bibliographic Notes

In this chapter, we argued that Boolean specifications provide an interesting means to formalize tasks which address several of the limitations of arbitrary objective functions, e.g., lack of compositionality, explicit historical dependencies, and the tight coupling with the dynamics. To address these deficits, recent works have proposed learning Boolean task specifications, e.g. logic or automata, which admit well defined compositions, explicitly encode temporal constraints, and have workspace independent semantics. The development of this literature mirrors the historical path taken in reward based research, with works adapting optimal control [84, 37], Bayesian [131, 161], and maximum (causal) entropy [149, 153] IRL approaches.

Interestingly, the generalized size principle derived in Prop 4.4.1 provides a connection between the Bayesian perspective of [131] and this work. In particular, [131] represents tasks as a conjunction of logical statements. A Bayesian model is then asserted which has the likelihood odds ratio depend on the relative number of conjunctions in the task. Since conjunction corresponds to language intersection, increasing conjunctions correlates with decreasing the concept size. This in turn correlates with decreasing the random action competency.

The folk modeling tricks for handling violations (Sec 4.3) enables relating this work with the prior work [84] which seeks to minimize the violations of a temporal logic formula. In particular, [84] asserts an objective that minimizes the violations (measured as the size of the path substring one needs to eliminate to make the task satisfied) relative to random actions. The folk trick for modeling violations works similarly by effectively introducing probabilistic transitions to the DFA. This results in ignoring subsequences of the string, at the cost of the additional description complexity to describe not transitioning to the failure sink. Prop 4.4.1 finishes the connection by making the comparison relative to random actions.

Finally, we remark that often, it makes sense to use task specifications to define constraints rather than objectives, e.g. run as fast as possible *while avoiding hurdles*. As such, a parallel literature has emerged for learning constraints (typically with a known reward) that explain expert demonstrations [128, 36, 103]. The key difference is that rather than changing the reward function, this literature (implicitly) modifies the underlying MDP. Nevertheless, conceptually, many of the problems in this literature echo those in the specification learning described above. That is, the key missing ingredient in all these works is a means by which to strategically explore a large representation class. The next chapter will attempt to address this deficiency.

Chapter 5

Finding Explanatory Specifications

For every complex problem there is an answer that is clear, simple, and wrong.

H. L. Mencken (Journalist / Satirist, 1880-1956)

As was motivated in the introduction and previous chapter, expert demonstrations provide an expressive means to informally specify a task. In this chapter, we continue studying the problem of inferring, from demonstrations, tasks represented by formal task specifications, e.g., automata and temporal logic. The key difficulty when learning task specifications from

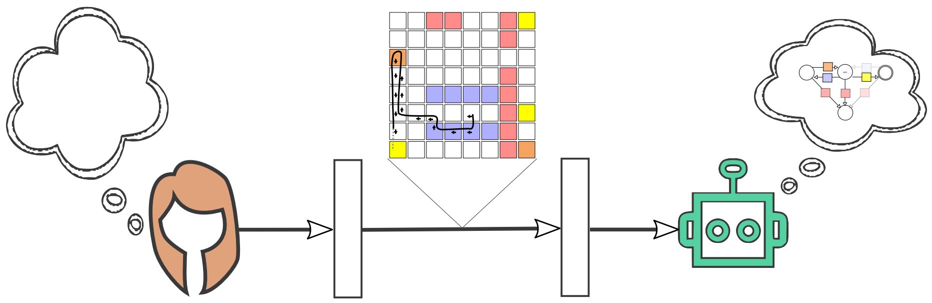


Figure 5.1: Illustration of learning an unknown task specification, represented on the right by an automaton, from a human demonstration.

demonstrations is how to search an intractably large (often infinite) representation class. For example, in contrast to the reward setting, the discrete nature of automata and logic, combined with the assumed *a-priori* ignorance of the relevant memory required to describe the task, makes existing gradient based approaches either intractable or inapplicable. Instead, current literature either (syntactically) enumerates representations [149, 37, 131, 161] or hill climbs via simple probabilistic (syntactic) mutations [84, 24].

Contributions

To this end, this chapter introduces a family of approximate algorithms called *Demonstration Informed Specification Search* (DISS). DISS is the result of asking the question: Given the current candidate task, what counter-factuals still require explanation? For example, one might need to explain why the agent did not take a particular shortcut. This results in a series of supervised learning problems, where the surprising counter-factuals are addressed, e.g., the shortcut is not taken because it is a negative example. This process is illustrated in Fig 5.2 in which DISS cycles between guessing a task, finding a counter-factual that needs explanation, and selecting which counter-factuals to focus on next.

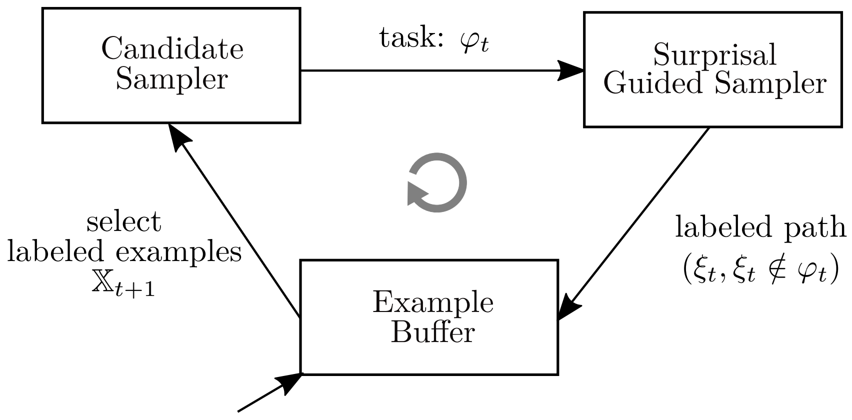

Candidate Sampler
task: φ_t
Surprisal Guided Sampler
select labeled examples $\mathbb{X}_{t+1}$
labeled path $(\xi_t, \xi_t \notin \varphi_t)$
Example Buffer

Figure 5.2: Demonstration Informed Specification Search overview.

The result will be an approximate solution to the Task Inference from Demonstrations Problem from Ch 4. After describing DISS, we perform two experiments validating that DISS indeed enables efficiently searching for tasks that explain the demonstrations even in large/unstructured representation classes like DFAs given unlabeled and potentially incomplete demonstrations. Finally, we end the chapter by discussing how to address violations in the Luce axiom, i.e., redundant actions.

Remark 5.0.1. The choice of DFAs as the representation class for our experiments is motivated by three observations. First, DFAs explicitly encode memory, making the contribution of identifying relevant memory more clear. Next, to our knowledge, all other techniques for learning finite path properties from demonstrations focus on syntax defined representation classes. Thus, learning DFAs is understudied in this context. Third, DFAs constitute a very large and mostly unstructured representation class. Therefore, DFAs facilitate studying the efficiency of DISS without introducing too much user defined inductive biases. By comparison, existing techniques for learning task specifications from demonstrations all use syntactically defined logics each with their own inductive biases. Thus direct comparisons would conflate search efficiency with the inductive biases of the representation classes.

5.1 Running Example

Let us return to the gridworld example from the previous chapter. In particular, consider an agent operating in the 8x8 grid world as shown in Fig 5.3.The agent can attempt to move up, down, left, or right. With probability $1/32$, wind will push the agent down, regardless of the agent's action. The black path is the *prefix* of an episode, in which the agent attempts to move left, slips into the blue tile (🟦), visits a brown tile (🟧), and then proceeds downward.

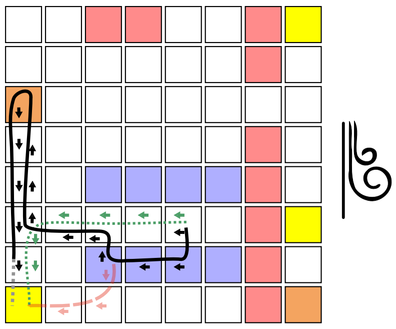


Figure 5.3

Given the black demonstration, call ξ_b, and the *prior* knowledge that the agent's task implies that it will avoid red tiles (🟥), what task, as a DFA, explains the agent's behavior?

Upon inspecting the demonstration, one might hypothesize that the complete path formed by extending ξ_b with the grey dashed lined to 🟨 is a positive example of the task. Appealing to Occam's razor, one might conjecture that the task was just to reach 🟨 and avoid 🟥. Note however that under this hypothesis, and assuming a temporal discount, ξ_b is quite surprising. For one, the detour to visit 🟧 seems unjustified. Furthermore, why would the agent not take the red dashed path directly to 🟨?

To remedy these concerns, one might conjecture that the agent's true task requires visiting 🟧 after visiting 🟦 - thus explaining why the agent does not take the red dashed path. Similarly, the demonstration seems less surprising if one assumes that the agent needs to avoid red tiles - thus explaining why the agent does not take the light blue dotted path. The result is the task represented by the DFA shown in Fig 5.4. We shall later systematize this line of reasoning and provide a learner that recovers an explanatory DFA given a demonstration.

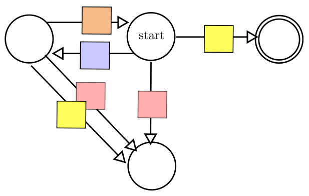


Figure 5.4

In the sequel, we will (i) discuss how to find surprising paths given our planner; and (ii) propose a variant of simulated annealing (implemented through the example buffer) to approximately solve our task inference from demonstration problem.

5.2 Prefix Tree Perspective

We start by discussing the prefix tree of the demonstrations. As we shall see, the prefix tree will serve as a mechanism to reason about the various paths *not* taken.

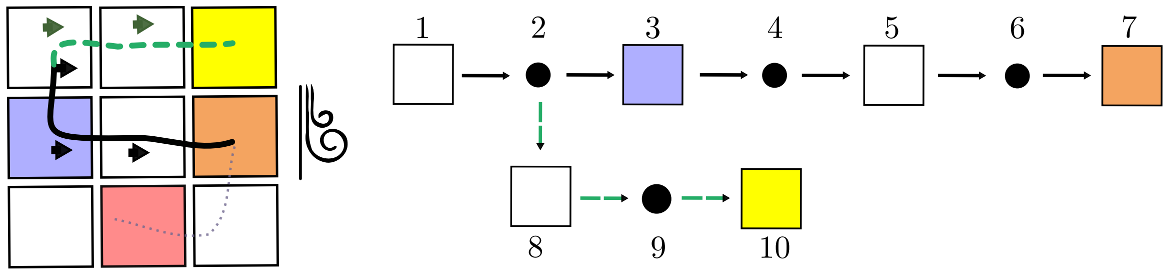


Figure 5.5: Prefix tree with 12 nodes for the paths shown on the left.

Definition 23. Let $\xi_1^*, \ldots \xi_m^*$ be a multi-set of demonstrations (paths) and denote by $\mathcal{T} = (N, E)$ the prefix tree of the $\xi_1^*, \ldots \xi_m^*$, where N and E are the prefixes (nodes) and edges of $\mathcal{T}$, respectively. A node, ρ, is said to be an **ego node** if its prefix ends in a state, i.e. $\text{last}(\rho) \in S$. A node that is not an ego node is called an **environment (env)** node. A path, ξ, **pivots** at node ρ if ρ is the longest prefix of ξ in N. The **pivot actions** (and **pivot states**) of a node, ρ, are the set of available actions (states) that result in pivoting at ρ, i.e.,

$$A_\rho \overset{\text{def}}{=} \{a \mid \rho \centerdot a \in \text{Paths} \setminus N\} \qquad S_\rho \overset{\text{def}}{=} \{s \mid \rho \centerdot s \in \text{Paths} \setminus N\}. \tag{5.1}$$

Example 5.2.1. *Consider the MDP shown in Fig 5.5 with two demonstrations ξ_1^* and ξ_2^* shown as a green dashed and black solid line resp. The prefix tree of $\{\xi_1^*, \xi_2^*\}$ is shown on the right. For convenience an index is associated with each node (prefix). For example, ρ_1 is the root node and ρ_7 is a leaf. Node ρ_3 has $A_{\rho_3} = \{\uparrow, \downarrow\}$. There is a path that pivots at every node except node ρ_2, since both possibilities (slipping/not slipping) appear in the demonstrations yielding $S_{\rho_2} = \emptyset$. The grey dotted line shows a suffix, y, such that $\rho_7 \centerdot y$ pivots at ρ_7. This implies that ρ_7 is not a complete path, i.e., $\rho_7 \notin Paths_\$$.*

Next, observe that because weighted averaging and LSE are commutative, one can aggregate the values of a set of actions or set of states (environment actions). This motivates defining the **pivot value** of a node ρ as:

$$\mathbb{V}_\rho^\varphi \overset{\text{def}}{=} \begin{cases} \text{LSE}_{a \notin A_\rho} V_\varphi(\rho \centerdot a) & \text{if } i \text{ is ego,} \\ \mathbb{E}_s[V_\varphi(\rho \centerdot s) \mid \rho, M, s \notin S_\rho] & \text{if } i \text{ is env,} \end{cases} \tag{5.2}$$

We shall denote by $\mathbb{V}^\varphi \in \mathbb{R}^N$ the node-indexed vector of pivot values associated with task φ under our maximum entropy agent model. We note two properties of pivot values. First, they strictly increase as the language of a task specification is made larger:

Proposition 5.2.2 (Pivot values respect subsets)**.** Let ξ be a complete path that pivots at node i. If $\varphi \subsetneq \psi$ and $\xi \in \psi \setminus \varphi$, then $\mathbb{V}_i^\varphi < \mathbb{V}_i^\psi$.

Proof. Follows inductively from the monotonicity of $\mathbb{E}$, $\sum$, and ln. □

Second, using the soft Bellman backup (3.25), one sees that the pivot values **entirely determine** (see Fig 5.6) the values, V, of the prefixes of the demonstrations. Namely, let $\hat{V}_k(\mathbb{V})$ denote the *derived* value at node k in the prefix tree, and let $\Pr(i \rightsquigarrow k \mid \mathbb{V})$ denote the probability of transitioning from node i to node k under the (local) policy: $e^{\hat{V}_j(\mathbb{V})-\hat{V}_i(\mathbb{V})}$.

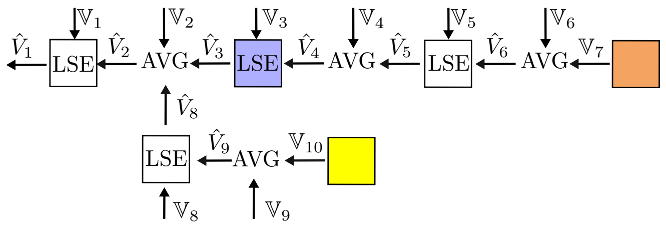


Figure 5.6: Computation tree of $\hat{V}$ values for each node of prefix tree given by soft Bellman backup (3.25) and pivot values, $\mathbb{V}$.

Example 5.2.3. *Consider again the prefix tree shown in Fig 5.5. Node 3 has* $A(\mathit{last}(\rho_3)) = \{\uparrow, \downarrow, \rightarrow\}$ *and* $A_{\rho_3} = \{\uparrow, \downarrow\}$. *Suppose the value of each action is* 1, *i.e.,* $V(\rho \centerdot a) = 1$ *for* $a \in \{\uparrow, \downarrow, \rightarrow\}$. *Since node* 3 *is an ego node, then*

$$\mathbb{V}_\rho = \ln(e^{V(\rho \,\cdot\, \uparrow)} + e^{V(\rho \,\cdot\, \downarrow)}) = \ln(2e) = 1 + \ln(2).$$

Now by definition,

$$V(\rho) = \ln(e^{V(\rho \,\cdot\, \rightarrow)} + e^{V(\rho \,\cdot\, \uparrow)} + e^{V(\rho \,\cdot\, \downarrow)} = \ln(e^1 + e^1 + e^1) = 1 + \ln(3).$$

Which due to associativity is the same as:

$$V(\rho) = \ln(e^{V(\rho \,\cdot\, \rightarrow)} + e^{\mathbb{V}_\rho}) = \ln(e^1 + e^{\ln(2\cdot e)}) = 1 + \ln(3)$$

And thus, either way, the probability going from node 3 to 4 (applying action right from ρ_3*) is*

$$\Pr(\rho_3 \rightsquigarrow \rho_4 \mid \mathbb{V}) = e^{1-(1-\ln(3))} = {}^1\!/_3.$$

Benefits of the local perspective

While strange at first, we argue that working with prefix trees and pivots offers a number of advantages to working directly with demonstrations. First, pivot values allow defining a local policy and (thus demonstration likelihood) that is divorced from the underlying dynamics, and notably, the number of actions and states. More concretely, note that not only is the local policy entirely determined by the pivot values, $\mathbb{V}$, the pivot values are entirely determined by the local policy[1]. Thus, knowing the local policy is sufficient to derive the pivot values.

Further note that, for non-zero temperatures, the maximum causal entropy policy assigns non-zero probability to every action. Thus, one can determine if a node can be a pivot by summing the probabilities of the descendants in the prefix tree, e.g.,

$$A_\rho = \emptyset \iff \sum_{a \notin A_\rho} \Pr(\rho \rightsquigarrow \rho \centerdot a \mid \mathbb{V}) = 0. \tag{5.3}$$

Thus, the underlying planner is free to use a large (perhaps even continuous) workspace model. So long as the local probabilities do not change, the likelihood of the demonstrations will also not change.

Independence of pivot values

Second, the pivot actions and states at a node indicate which pivot values can be changed *independently*. In particular, note that (i) task specifications are history dependent (ii) each pivot action (state) necessarily accesses a subtree - each of which disjoint from the others (iii) in sufficiently expressive representation classes, e.g. minDFA, one can always construct two tasks, φ, φ' whose languages only differ only on single path. Since pivot values strictly increase as a language is expanded (Prop 5.2.2) one observes that $\mathbb{V}^\varphi$ and $\mathbb{V}^{\varphi'}$ only differ in a single coordinate.

5.3 Manipulating Likelihoods and Surprisal

As stated at the start of the chapter, the question motivating our algorithm design is the following: Given the current candidate task, what counter-factuals still require explanation? In this section, we shall formalize this question by (i) measuring how surprising (in nats) the demonstrations are given the pivot values induced by the current task specification, (ii) seeing how the description length would change as the pivot values change, and (iii) discussing how to propagate a suggested change in pivot values to a change in task specification.

To start, we define the surprisal of the local policy induced by the pivot values as follows:

Definition 24. Let $\mathcal{T} = (N, E)$ be a prefix tree of demonstrations, $\xi_1^*, \ldots \xi_m^*$. The **pivot surprisal** of given $\mathcal{T}$ is map, $\hat{h} : \mathbb{R}^d \to \mathbb{R}$, where d is the number of nodes that can be pivots and:

$$\hat{h}(\mathbb{V}) \overset{\text{def}}{=} -\sum_{(i,j)\in E} \#_{(i,j)} \cdot \ln \Pr(i \rightsquigarrow j \mid \mathbb{V}). \tag{5.4}$$

The **task surprisal** of a task specification, φ, is the pivot surprisal of $\mathbb{V}^\varphi$, i.e.,

$$h(\varphi \mid \xi_1^*, \ldots \xi_m^*) \overset{\text{def}}{=} \hat{h}(\mathbb{V}^\varphi). \tag{5.5}$$

[1]Follows inductively from the strict monotonicity of expectation and LSE.

Remark 5.3.1. Note that the task surprisal induced by a prefix tree is simply the negative log likelihood of the demonstrations:

$$\Pr(\xi_1^*, \ldots \xi_m^* \mid \varphi, M) = e^{-h(\varphi | \xi_1^*, \ldots \xi_m^*)}.$$

Working with surprisal rather than likelihood is done for two reasons: (i) Mechanically, the gradient of $\hat{h}$ admits a simple form (ii) As we saw in the preliminaries, the surprisal has a natural interpretation in terms of description length. Thus working with surprisal allows discussing the "combined" description length of the demonstrations and task as a single object, e.g., using a size prior on tasks yields:

$$\Pr(\varphi \mid \xi_1^*, \ldots \xi_m^*, M) \propto \exp\Big(-(h(\varphi \mid \xi_1^*, \ldots \xi_m^*) + \text{size}(\varphi))\Big).$$

The interpretation is that one seeks an easy to describe task specification that also explains the demonstrations.

Pivot surprisal gradients

Again motivated by the question of what counter-factuals still require explanation, we ask a related question: How could the pivot values change to make the demonstrations more likely. For example, for ego nodes, one might want to make the value of the observed actions large and the pivot value small. The result would be an agent with no incentive to pivot. Unfortunately, changing a pivot value changes the policy in a non-local way, e.g., changing $\mathbb{V}_9$ in Fig 5.6 also changes the policy for nodes 9, 8, 2, and 1. Fortunately, these upstream effects are easily summarized by the gradient of $\hat{h}$.

Proposition 5.3.2 ($\nabla \hat{h}$ determined by local policy)**.** Let $p_{xy}(\mathbb{V})$ denote the probability of starting at node x and pivoting at y, i.e.,

$$p_{xy}(\mathbb{V}) \overset{\text{def}}{=} \Pr(x \rightsquigarrow y \mid \mathbb{V}) \cdot \Bigg(1 - \sum_{(y,z) \in E} \Pr(y \rightsquigarrow z \mid \mathbb{V})\Bigg) \tag{5.6}$$

then,

$$\frac{\partial \hat{h}}{\partial \mathbb{V}_k} = \sum_{\substack{(i,j) \in E \\ i \text{ is ego}}} \#_{(i,j)} \cdot \Big(p_{ik}(\mathbb{V}) - p_{jk}(\mathbb{V})\Big) \tag{5.7}$$

The proof of Prop 5.3.2 is fairly mechanical and can be found at the end of the chapter.

Remark 5.3.3. Prop 5.3.2 illustrates that gradients are simple to compute given only access to the policy on the prefix tree.

Relation to heuristics

Eq (5.7) captures several intuitive heuristics for changing $\mathbb{V}$ to make the demonstrations more likely. Consider ego edge (i, j), which corresponds to an action the agent took. Pivoting at i, i.e., $k = i$ above, yields, $p_{jk}(\mathbb{V}) = 0$. Thus, edge (i, j) contributes positively to the gradient. Since we want to minimize surprisal, this suggests that we want a to decrease $\mathbb{V}_k$ and thus make the action the agent took look more valuable by comparison. Similarly, suppose $k = j$, implying that:

$$p_{ik} - p_{jk} = \Pr(i \rightsquigarrow j \mid \mathbb{V}) \cdot p_{jk} - p_{jk} \leq 0,$$

where equality only holds iff only one action is available. Thus, when $k = j$, edge (i, j) typically provides a negative contribution to the surprisal gradient. Again, because we want to minimize surprisal, the above suggests an increase in $\mathbb{V}_k$, and by extension, an increase in the value of the action used on (i, j). These two cases can be understood as:

1. Make the actions taken more optimal by decreasing the value of other actions.
2. Make the actions taken less risky by increasing the value of possible outcomes.

Mislabeled counter-factuals

One may view $-\nabla\hat{h}(\mathbb{V}^{\varphi})$ as a prescription for modifying the pivot values in order to make the demonstrations less surprising. That said, because of the expressivity of representation classes like DFAs, there is a real concern that globally optimizing $\hat{h}$ will overfit to the demonstrations and ignore the prior distribution. Thus, the goal will not be to simulate gradient descent under $\nabla\hat{h}$, but to instead help identify counter-factual paths that require explanation.

More precisely, if $\frac{\partial\hat{h}}{\partial\mathbb{V}_\rho} > 0$ for a given pivot ρ, then one can decrease the surprise by decreasing $\mathbb{V}_\rho$. Recalling Prop 5.2.2, a concrete way to decrease $\mathbb{V}_\rho$ is to remove a path from the language of φ. Said another way, $\frac{\partial\hat{h}}{\partial\mathbb{V}_\rho}$ being large is evidence that there exists some counter-factual path, $\xi \in \varphi$, that pivots at ρ and requires additional explanation e.g., "The agent did not take the shortcut through the lava because it was avoiding lava." Similarly, $\frac{\partial\hat{h}}{\partial\mathbb{V}_\rho} < 0$ suggests that there is a counter-factual path pivoting through ρ that should be added to φ, e.g., "The agent moved near the ledge, because even if it slips, ξ shows a way to perform the task." This analysis can be summarized below.

Let ξ be a complete path with pivot ρ such that:

$$\xi \in \varphi \iff \frac{\partial\hat{h}}{\partial\mathbb{V}_\rho} > 0, \tag{5.8}$$

If ξ is a likely path under π_φ (and thus has a large effect on $\mathbb{V}$) and the pivot surprisal gradient $\frac{\partial\hat{h}}{\partial\mathbb{V}_\rho}$ is large in absolute value, then ξ may be mislabeled by φ.

5.4 Specification Search

In this section, we take the insights developed in the previous sections and propose the Demonstration Informed Specification Search (DISS) algorithm. As stated throughout the chapter, DISS can be viewed as systematizing the following question: Given the current candidate task, what counter-factuals still require explanation.

More specifically, DISS assumes (i) access to a multi-set of expert demonstrations: $\xi_1^*, \ldots \xi_m^*$; (ii) *black box* access to an identification algorithm, $\mathcal{I}$, that maps positively/negatively labeled paths to a distribution over tasks; and (iii) *black box* access to a planner that estimates the probability of a path given a candidate task (see Part II). DISS operates by cycling between three components (shown in Fig 5.7):

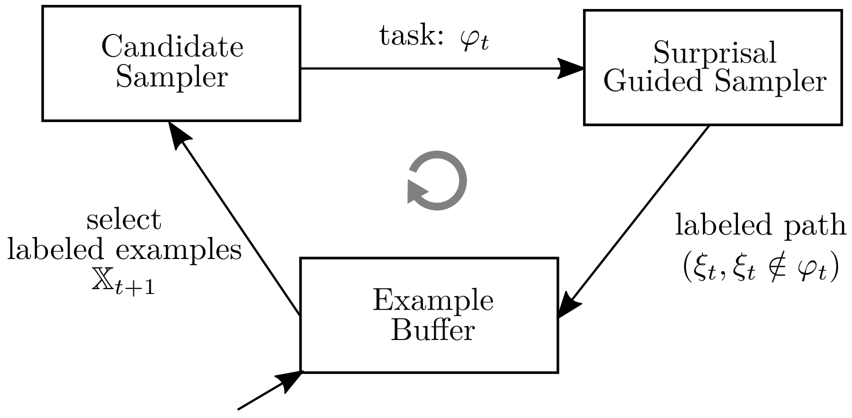

Candidate Sampler
task: φ_t
Surprisal Guided Sampler
labeled path $(\xi_t, \xi_t \notin \varphi_t)$
Example Buffer
select labeled examples $\mathbb{X}_{t+1}$

Figure 5.7: Demonstration Informed Specification Search overview.

1. **Candidate Sampler**: A candidate task, φ, is sampled using $\mathcal{I}(\bullet \mid \mathbb{X}, \varphi')$, where $\mathbb{X}$ is a collection of labeled examples (initialized to the empty set) and φ' is the previously proposed task.

2. **Surprisal Guided Sampler**: Using the surprisal gradient, the planner is used to find paths that may be mislabeled by the current task.

3. **Example Buffer**: Given previously seen data, the example buffer yields a set of positive and negative example paths (see below). The example buffer plays the critical role of enabling back tracking and dropping labeled paths that do not end up constraining the concept identifier in a useful way.

Because the candidate sampler is assumed to be user provided, what remains then is to discuss the Surprisal Guided Sampler and the Example Buffer.

Remark 5.4.1. DISS is a random walk through labeled example space guided by the surprisal gradient of sampled concepts. Each new labeled example constrains the next task to address various counter-factuals. As we shall see, this accumulation of corrective constraints systematically focuses the concept sampler on more and more probable (lower energy) task specifications.

Remark 5.4.2. Note that adapting DISS to a new representation class simply requires changing the concept identifier used. For example, we have already discussed learning DFAs from positive examples. Similar algorithms exist for decision diagrams [144], Linear Temporal Logic [108], and syntactically defined programs [7].

Remark 5.4.3. As we shall see, the surprisal guided sampler discussed above is agnostic to the particular way that entropy regularized planning is realized, using only predictions along the demonstration. Thus, DISS is also agnostic to the size of the underling state and action space. In fact, because there are only ever a finite number of demonstrations - and thus finite number of state and actions - the workspace can be made continuous with no changes to DISS.

Surprisal Guided Sampler

The Surprisal Guided Sampler (Alg 1) builds off the insights developed when studying the gradient of the pivot surprisal gradient and summarized in (5.8).

Algorithm 1 Surprisal Guided Sampler (SGS)

```
1: Input: $\varphi, \mathbb{X}, \mathcal{T}, M, \beta$
2: Compute $\pi_\varphi$ given $M$ and $\mathcal{T}$.
3: Sample a path $\xi \sim (\pi_\varphi, M)$ and a pivot $\rho \sim \text{softmax}_i \left|\frac{1}{\beta} \cdot \frac{\partial \hat{h}}{\partial \mathbb{V}_i}\right|$ s.t.
       i $\xi$ pivots at $\rho$.
       ii $\xi \in \varphi \iff \frac{\partial \hat{h}}{\partial \mathbb{V}_\rho} > 0$.
       iii $\exists \varphi' \in \mathcal{R}$ s.t. $\varphi'$ is consistent with: $\mathbb{X} \cup \{(\xi, \xi \notin \varphi)\}$.
4: return $\xi$                                   ▷ Conjecture mis-labeled path.
```

Alg 1 takes as input the current task specification, φ, a set of labeled examples, $\mathbb{X}$, the prefix tree, $\mathcal{T}$, of the demonstrations, the dynamics, M in the form of a MDP, and a temperature parameter β. Next, on line 2, a maximum causal entropy planner, π_φ, for φ is constructed. Using π_φ and $\mathcal{T}$, a distribution over pivots and paths is created as follows. First, the gradient of the pivot surprisal is used to define a β annealed softmax distribution over pivots. Second, a path distribution is defined using π_φ. Finally, a pivot, ρ, and path, ξ, are sampled from the joint distribution over paths and pivots is conditioned on: (i) ξ actually pivoting as ρ; (ii) the label of ξ under φ being align with the pivot surprisal gradient; and (iii) adding ξ to the previous examples, $\mathbb{X}$, does not result in all tasks being inconsistent. Conditions (i) and (ii) are motivated by our previous argument that likely paths with large pivot values are likely mislabeled. Condition (iii) is a technical requirement needed to restrict attention to tasks within the representation class, e.g., we know that the agent should not visit red tiles and so labeling such as path as positive is inconsistent with the representation class. This process is summarized in in Fig 5.8.

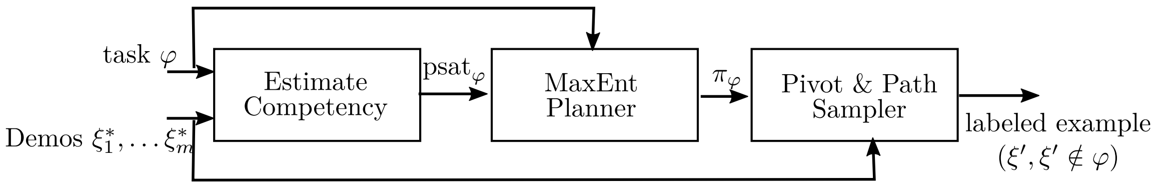


Figure 5.8: Overview of the surprisal guided sampler.

Example 5.4.4. *Recall our ad-hoc analysis on Fig 5.3 in Sec 5.1. Under the reach* 🟨 *while avoiding* 🟥 *hypothesis, φ, it is surprising that the agent moves up from* 🟦 *rather than following the red dashed suffix. That is, there is a non-trivial probability to deviate at this prefix, call k. Using a planning horizon of 15 steps, $\frac{\partial \hat{h}}{\partial \mathbb{V}_k}$ is positive and indeed larger in magnitude than all other pivots. Props 5.2.2 and 5.3.2 suggest sampling (using π_φ) a new path, ξ, that pivots from the demonstration at k and satisfies φ. Note that the illustrated red dashed suffix indeed fits this description. Finally, (5.8) prescribes marking ξ as a negative path which matches our ad-hoc analysis at the start of the chapter.*

Remark 5.4.5. For some pivots there is no path that satisfies conditions (i), (ii), and (iii). For example if ρ accesses a fail sink state in a DFA, but (ii) requires sampling a positive path. This suggests sampling from the SGS distribution by first sampling a pivot and then sampling an extension. If a condition is impossible to realize at the given pivot, then that pivot is removed form the pivot distribution since it has 0 probability mass. This form of rejection sampling is repeated until a (pivot, path) pair is found.

Remark 5.4.6. Alg 1 requires only black-box access to π_φ for assigning edge probabilities, $\Pr(i \rightsquigarrow j \mid \mathbb{V})$ and sampling suffixes given a pivot. If the satisfaction probability of an action is also known, i.e., $\Pr_{\xi'}(\xi \centerdot \xi' \in \varphi \mid \xi, M, \pi_\varphi)$, then one can more efficiently sample suffixes using Baye's rule.

Example Buffer and Backtracking

The final component of DISS to discuss is the Example Buffer. As mentioned above, the Example Buffer plays the important role of managing the "state" of DISS by determining which subset of labeled examples seen so far is currently constraining the concept identification algorithm. To see why this is necessary, imagine greedily accumulating labeled examples. Since SGS is only an approximate method for determining which counter-factuals need to be labeled, it is possible to make mistakes and fall in local optimal. To mitigate this issue, the example buffer implements a variant of Simulated annealing, a well studied technique for heuristic global optimization.

Simulated Annealing

At a high level, *Simulated Annealing* (SA) [134] is a probabilistic optimization method that seeks to minimize an energy function $U : Z \to \mathbb{R} \cup \{\infty\}$. To run SA, one requires three ingredients: (i) a *cooling schedule* which determines a monotonically decreasing sequence of temperatures; (ii) a *proposal* (neighbor) distribution $q(z' \mid z)$; and (iii) a *reset* schedule, which periodically sets the current state, z_t, to one of the lowest energy candidates seen so far.

A standard simulated annealing algorithm then operates as follows: (i) An initial $z_0 \in Z$ is selected; (ii) T_t is selected based on the cooling schedule; (iii) A neighbor z' is sampled from $q(\bullet \mid z_t)$; (iv) z' is accepted ($z_{t+1} \leftarrow z'$) with probability:

$$\Pr(\text{accept} \mid z', z_t) = \begin{cases} 1 & \text{if } dU > 0 \\ \min\left\{1, e^{dU/T_t}\right\} & \text{otherwise} \end{cases}, \tag{5.9}$$

where $dU \overset{\text{def}}{=} U(z) - U(z')$; (v) Finally, if a reset set is trigger, z_{t+1} is sampled from previous candidates, e.g., uniform on the argmin.

As previously stated, we propose a variant of simulated annealing adapted for our specification inference problem. We will start by assuming the posterior distribution on tasks takes the form:

$$\Pr(\varphi \mid \xi_1^*, \ldots \xi_m^*, M) \propto e^{-U(\varphi)}, \tag{5.10}$$

where the *energy*, U, is given by:

$$U(\varphi) \overset{\text{def}}{=} h(\varphi \mid \xi_1^*, \ldots \xi_m^*) + \theta \cdot \text{size}(\varphi), \tag{5.11}$$

and $\theta \in \mathbb{R}$ determines the relative weight of the surprisal. That is, we appeal to Occam's razor and assert that the task distribution is exponentially biased towards simpler tasks, where simplicity is measured by the description length of the task, size(φ), and the description length (i.e. surprisal) of $\xi_1^*, \ldots \xi_m^*$ under (π_φ, M).

Remark 5.4.7. θ plays two roles: First, it is responsible for making h and size comparable, e.g., converting from bits to nats. Second, θ, determines which geometric distribution (with mean $1/\theta$) over size(r) is used for the prior distribution. Thus, one should set θ such that $1/\theta$ is larger than description length of a "large" representation. For example, below, we will use $\theta = 1/50$ to tell DISS that we believe the demonstrated task does not require much more than 50 nats to describe. For reference, the ground truth DFA uses about ≈ 40 nats.

Demonstration Informed Specification Search

At last, using the language of SA, we define DISS as follows (with pseudo code in Alg 2).

1. Z is the set of all a tuple of labeled examples and a task specification - each adjoined with $\perp$. The algorithm starts with the tuple: $z_0 = (\perp, \perp)$.

2. The proposal distribution, $q(\mathbb{X}', \varphi' \mid \mathbb{X}, \varphi)$ is defined to first sample a concept using an identification map, $\varphi' \sim \mathcal{I}(\bullet \mid \mathbb{X}, \varphi)$, then run SGS on φ' to conjecture a labeled path ξ, yielding $\mathbb{X}'' = \mathbb{X} \cup \{(\xi, \xi \notin \varphi')\}$. Next, with probability, p_{drop}, examples are dropped from $\mathbb{X}''$, yielding $\mathbb{X}'$.
3. Resets occur every $\kappa \in \mathbb{N}$ time steps. If a reset is triggered, $\mathbb{X}_{t+1}$ is sampled from $\text{softmin}_{i \leq t} U(\varphi_i)$, and φ_{t+1} is sampled from $\mathcal{I}(\bullet \mid \mathbb{X}_{t+1}, \bot)$.

Remark 5.4.8. Again, the main role of simulated annealing is to guide the search through labeled example space for constraints on the identification algorithm. The examples are encouraged to be useful via three mechanisms: First, labeled examples whose next proposal does not decrease the energy are only only sometimes accepted. Next, at each iteration, there is some probability that an example will be dropped. Thus, the examples that tend to been seen in $\mathbb{X}$ are those that are repeatedly conjectured when missing. Finally, resets help make global progress after mistakes lead to a local minima.

Algorithm 2 Demonstration Informed Specification Search.

```
 1: input: $(\xi_1, \ldots, \xi_m), M, \theta, N, \kappa, p_{\text{drop}}$
 2: Compute $\mathcal{T}$ given $(\xi_1, \ldots, \xi_m)$.          ▷ Create prefix tree.
 3: $\Phi \leftarrow \emptyset$.
 4: for $t$ in $1, \ldots, N$ do
 5:     if $t \equiv 0 \pmod{\kappa}$ then
 6:         $\mathbb{X} \sim \arg\max_{\psi \in \Phi} U(\psi)$           ▷ Reset periodically.
 7:         $(\varphi, d\mathbb{X}) \leftarrow (\bot, \emptyset)$
 8:     $\mathbb{X}' \leftarrow \text{update}(\mathbb{X}, d\mathbb{X}, p_{\text{drop}})$        ▷ Add and drop examples.
 9:     $\varphi' \sim \mathcal{I}(\bullet \mid \mathbb{X}', \varphi)$.               ▷ Sample candidate task.
10:     $\Phi \leftarrow \Phi \cup \{\varphi'\}$.                   ▷ Update visited specs.
11:     $T \leftarrow \text{cooling\_schedule}(t)$                ▷ User defined.
12:     $dU \leftarrow U(\varphi') - U(\varphi)$
13:     $\alpha \sim \text{Uniform}(0, 1)$
14:     if $dU < 0$ or $\exp(-{dU}/{T}) \leq \alpha$ then
15:         $(\varphi, \mathbb{X}) \leftarrow (\varphi', \mathbb{X}')$
16:         $d\mathbb{X} \leftarrow \{\text{SGS}(\varphi, T, M)\}$        ▷ Conjecture labeled example.
17:     else
18:         $d\mathbb{X} \leftarrow \emptyset$                        ▷ Reject proposal.
19: return $\Phi$
```

5.5 Experiments

In this section, we illustrate the effectiveness of DISS by having it search for a ground truth specification, represented as a DFA, given the expert demonstrations, ξ_b, ξ_g, from our motivating example (shown in Fig 5.3). The (dotted) green path, ξ_g, goes directly 🟨. The (solid) black path, ξ_b, immediately slips into 🟦, visits 🟧, then proceeds towards 🟨. This path is incomplete, with a possible extension, σ_b, shown as a dotted line. The ground truth task is the right DFA in Fig 5.4.

We consider two specification inference problems by varying the representation class and the provided demonstrations. These variants respectively illustrate that (i) our method can be used to incrementally learn specifications from unlabeled incomplete demonstrations; and (ii) the full specification can be learned given unlabeled complete demonstrations.

1. **Monolithic**: ξ_g and $\xi_b \cdot \sigma_b$ are provided as (unlabeled) *complete* demonstrations. The representation class is MinDFA.
2. **Incremental**: ξ_b is provided as an (unlabeled) *incomplete* demonstration. The representation class, $\mathcal{R}$, is a variant of MinDFA. Let φ' denote the three state DFA for avoiding 🟥 and reaching 🟨. If a task, φ is in $\mathcal{R}$, then {🟨, 🟨🟨} $\subseteq \text{concept}(\varphi) \subseteq \text{concept}(\varphi')$. That is, prior knowledge is provided that you must reach 🟨, you must avoid 🟥, and you know two positive examples. The size of φ is given by:
$$\text{size}(\varphi) = \text{size}'(\varphi) - \text{size}'(\varphi'),$$
where size$'$ is the size function for the MinDFA representation class.

The surprisal weight, θ, is set to $^1/_{50}$ for both variants. Finally, two additional inductive biases, which empirically proved necessary for optimizing the baselines, are applied: (i) we remove white tiles, ☐, from labeled examples (ii) we transform sequences of repeated colors into a single color thus biasing towards DFA that do not count. For example, ☐🟦🟦☐🟨🟦 $\mapsto$ 🟦🟨🟦.

DISS parameters.

Our implementation of DISS [154] resets every 30 iterations, has $p_{\text{drop}} = {}^1/_{20}$, and uses the following cooling schedule:
$$T_t = 100 \cdot (1 - {}^t/_{100}) + 1. \tag{5.12}$$
Our experiment will sweep through various of SGS temperatures, $\beta \in [2^{-10}, \infty)$.

Maximum Entropy Planner

The maximum entropy planner used will be discussed in detail in Ch 9. The planning horizon was 15 steps. Because our experiments operate with one or two demonstrations, the rationality, λ, corresponding to π_φ is taken to be 10 for all specifications.

Concept Sampler

Our concept sampler, $\mathcal{I}(\bullet \mid \mathbb{X}, \varphi')$, was designed to respect the relative size[2]prior size$(\varphi \mid \varphi')$, i.e., $\mathcal{I}(\varphi \mid \mathbb{X}, \varphi') \propto \exp(-\text{size}(\varphi \mid \varphi'))$ if φ is consistent with $\mathbb{X}$ and 0 otherwise. To implement, $\mathcal{I}$, we adapted an existing SAT-based DFA identification algorithm [146] to enumerate the first 20 consistent DFAs, lexicographically ordered by the number of states and non-self loops they have. Note that lexicographic order is similar, but not exactly the same as the size defined for the minDFA representation class. In particular, it may be the case that a DFA with more states has less edges, and thus smaller size. This effect is even more pronounced if the reference task, φ' is not $\perp$ or the DFA has a sink state, e.g., an irrecoverable failure state.

To make the concept sampler respect the size prior of minDFA, we first sample a DFA from the enumerated DFAs, exponentially weighted by $-\text{size}(\varphi \mid \varphi')$. Next, to mitigate the blind spot the lexicographic order has for sink states, if a subset of symbols, Σ', contains only negative examples, a new DFA is created by intersecting the sampled DFA's language with the set of strings not containing Σ'.

Example 5.5.1. *Suppose* $\mathbb{X} = \{($🟨🟥$, 1), ($🟦🟥$, 0)\}$. *Then* $\Sigma' = \{$🟦$\}$ *and the concept sampler will return a DFA that is the intersection with a DFA consistent with* $\mathbb{X}$ *and that always rejects any strings with* 🟦. *Similarly, for* $\mathbb{X} = \{($🟨🟥🟦$, 1), ($🟦🟥$, 0)\}$, $\Sigma' = \emptyset$, *and so the initially sampled DFA is returned unchanged.*

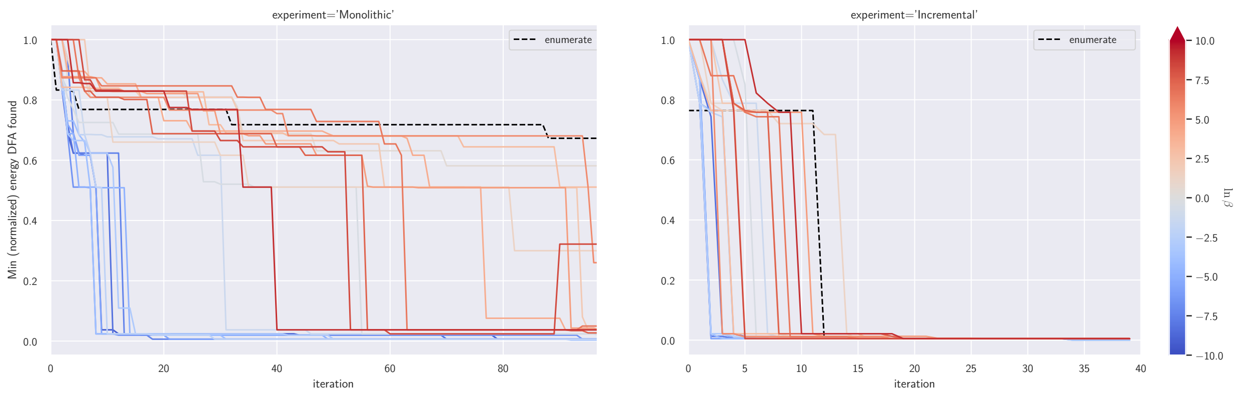
Baselines.

As mentioned in the introduction, existing techniques for learning specifications from demonstrations use various *syntactic* concept classes, each with their own inductive biases. Thus, we implemented two DFA-adapted baselines that act as proxies for the enumerative and probabilistic hill climbing style algorithms of existing work:

1. **Prior Guided Enumeration.** This baseline uses the same SAT-based DFA identification algorithm to enumerate DFAs in ordered by the size prior. This is done by finding the N smallest DFAs in lexicographic order (node then edges) as above and then ordering by size. $N = 80$ in the monolithic experiment and $N = 40$ in the incremental experiment. As an alternative to DISS's competency assumption, we allow the enumerative baseline to restrict the search to task specifications that accept the provided demonstrations.[3]
2. **Random Pivot DISS.** As mentioned above, we will evaluate DISS on various SGS temperatures, one of which has $\beta = \infty$. This results in a (labeled example) mutation based search with access to the same class of mutations as DISS, but samples pivots

[2]The relative size for minDFA, is defined as the number of bits required to described the changes to the DFA, e.g., the change in states, the change in edges, the change in the accepting set.

[3]For the incremental experiment, a counterexample loop is used to add labeled examples that bias the DFAs to imply φ_1.



(a) Result of monolithic experiment. **(b)** Result of incremental experiment.

Figure 5.9: DISS finds explanatory DFAs much faster than baselines.

uniformly at random, i.e., no gradient based bias. Note that this ablation still samples suffixes conditioned on the sign of the gradient, and thus the mutations are still informed by the surprisal.

Results and Analysis

To simplify our analysis, we present time in iterations, i.e., number of sampled DFAs, rather than wall clock time. This is for three reasons. First, for each algorithm, the wall clock-time was dominated by synthesizing maximum entropy planners for each unique DFA discovered, but the choice of planner is ultimately an implementation detail[4]. Second, because many DISS iterations correspond to the same DFAs (due to resets and rejections) the enumeration baseline explored significantly more *unique* DFAs than DISS (a similar effect occurs with the random pivot baseline, since the different pivots give more diverse example sets). Third, the enumeration baseline first enumerates DFAs in lexicographic order (without planning) and then computes the energies in order of increasing size. This incurs a significant (≈15s) overhead. Thus, using wall clock-time would skew the results below in DISS's favor.

Search efficiency

Fig 5.9a and Fig 5.9b show the minimum energy DFA for the monolithic and incremental experiments respectively. To reduce variance, we take the median of 5 runs for each β. We see that for both experiments, DISS was able to significantly outperform the enumeration baseline (recall that energy is the negative log of the probability) and tended to degrade in its search efficiency as β increased. For example for $\ln \beta < -5$, the monolithic experiment took between

[4]For reference our planner took around 4-10s per DFA.

10-30 iterations to find a energy minimizing DFA. Similarly, for for incremental, $\ln\beta < -5$ typically required only 1-2 iterations (compared to the 13 iterations of enumeration)!

The key takeaways are that:

1. DISS was significantly more (cycle) efficient at finding explanatory DFAs than prior based enumeration.
2. Relying on the surprisal gradient (by decreasing the pivot temperature) enables efficient exploration in large concept classes.
3. Using a stronger inductive bias such as asserting partial knowledge of the specification increases the search efficiency of DISS.
4. DISS can be effective even with a few incomplete and unlabeled examples.

Diversity of DFAs

In addition to finding the most probable DFAs much faster than the baselines, DISS also found *more* high probability DFAs. The most likely DFAs found by DISS for each experiment are shown in Fig 5.10. We observe that for both experiments, DISS is able to learn that if the agent visits 🟦, it needs to visit 🟧 before 🟨!

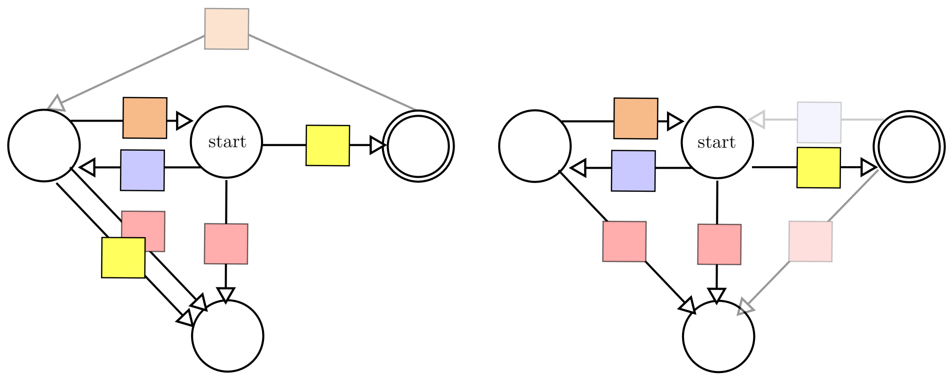


Figure 5.10: Most probable DFA found by DISS in the monolithic experiment (left) and the incremental experiment (right).

Nevertheless, our learned DFAs differ from ground truth, particularly when it comes to the acceptance of strings *after* visiting 🟨. We note that a large reason for this is that our domain and planning horizon make the left most 🟨 effectively act as a sink state. That is, the resulting sequences are effectively indistinguishable, with many even having the exact same energy. In Fig 5.10, we make such edges lighter, and note that the remainder of the DFAs show good agreement with the ground truth. Finally, while impressive, this points to a fundamental limitation of demonstrations. Namely, if two tasks have very correlated policies in a workspace, then without strong priors or side information, one is unable to distinguish the tasks. For example, if a task requires the agent to avoid 🟥, but no 🟥 are shown in the workspace, then one cannot hope to learn this aspect of the task.

5.6 Relaxing the Luce axiom[†]

Our design and analysis of DISS assumed the Luce axiom and in particular that all actions accessible by states in the demonstrations are distinct, i.e., if for all non-terminal states s,

$$\forall a_1, a_2 \in A(s) \ . \ a_1 \neq a_2 \implies \delta(s, a_1) \neq \delta(s, a_2).$$

This assumption is important several reasons. Chiefly, a violation of the Luce axiom along the demonstrations means that the pivot values are no longer independent. We illustrate with an example.

Example 5.6.1. *Consider the demonstration $\xi = s_0 \xrightarrow{a_2} s_1$, where $A(s_0) = \{a_1, a_2\}, A(s_1) \neq \emptyset$. Thus, for prefix $\rho = s_0$, we have $a_1 \in A_\rho$ and $a_2 \notin A_\rho$. Since a_1 is the only action that pivots, we have $\mathbb{V}_\rho = V(\rho \centerdot a_1)$. Similarly, since $A(s_1) \neq \emptyset$, ξ is not terminal and has its pivot value determined be the subtree accessed by $\rho \centerdot a_2$, i.e., $\mathbb{V}_\xi = \mathbb{V}_{\rho \centerdot a_2} = V(\rho \centerdot a_2)$. But since a_2 accesses the same subtree as a_1, then*

$$\mathbb{V}_\rho = V(\rho \centerdot a_1) = V(\rho \centerdot a_2) = \mathbb{V}_{\rho \centerdot a_2}.$$

Thus, $\mathbb{V}_\rho = \mathbb{V}_{\rho \centerdot a_2}$, implying the that pivot values are not independent.

This of course has serious implications on our use of the surprisal gradient. The most glaring issue is that two logically equivalent pivots may have opposing signs in the gradient. This can lead to conjecturing a counter-factual that is exactly counter to your goal. For example, if the last action in a positive demonstration is redundant with another action, pivoting at that point will likely result in the demonstration being marked negative! While it is possible for DISS to correct this conjecture in a future iteration, this can seriously degrade performance in practice. Furthermore, note that a similar argument can be made for redundant states!

Counter Example Guided Action Refinement

For these reasons, it behooves us to develop a way to handle redundant actions and states. Luckily there is a simple solution that only requires a minor modification to DISS:

1. Start by assuming that all (time-indexed) states and actions in the demonstrations are the equivalent. Thus, the prefix tree is initially a chain and no nodes can be pivots.

2. Whenever proof is found that two states (actions) are distinct at a given node in the prefix tree, create a new prefix tree where the prefix indexed equivalence class of states (actions) is partitioned accordingly.

To formalize this algorithm, we introduce the idea of value-distinguishability.

Definition 25. Let ξ and ξ' be two paths such that $\xi = \rho \centerdot x \centerdot y$ and $\xi' = \rho \centerdot x' \centerdot y'$, where $|x| = |x'| = 1$. Given a subset of the representation class $\Psi \subseteq \mathcal{R}$, we say that prefixes $\rho \centerdot x$ and $\rho \centerdot x'$ are Ψ-**value-distinguishable** if there exists a task, $\varphi \in \Psi$, such that:

$$V_\varphi(\rho \centerdot x) \neq V_\varphi(\rho \centerdot x'). \tag{5.13}$$

When $\Psi = \Phi$, we suppress Ψ and simply say value-distinguishable.

Importantly, if two prefixes are value distinguishable, then they maintain the independence of pivot values since they must not access the same subtree. Conversely, if two prefixes are not value distinguishable, then they access must access functionally equivalent sub-trees. Thus, value-distinguishability fits the needs of proof that two states (actions) are distinct.

The problem of course is that refuting value-distinguishability requires examining the whole representation class (which may be impossible). Instead, we define a series of equivalence relations based on an expanding sets of specifications, $\Psi_1 \subseteq \Psi_2 \subseteq \ldots$, where Ψ_i is the set of specifications visited by DISS by iteration i. Thus, at iteration i, prefix ρ and ρ' are in the same equivalence class if they are not Ψ_i-value-distinguishable.

In summary, to account for the possibility of redundant states and actions, maintain an series equivalence relations. Whenever two actions are deemed to be value-distinguishable, the equivalence relation is updated, which results in a new prefix tree over representatives of each equivalence class. Noting that the initial set of visited specifications is empty and that no prefixes are $\emptyset$-value-distinguishable, we have that (i) initial prefix tree is a chain; and (ii) no nodes can (initially) be pivots.

Remark 5.6.2. This variant of DISS requires knowing the set of states and actions reachable from each state. This can be further relaxed by dynamically testing states and actions for value-distinguishability as they observed, e.g., by sampling. While not maintaining pivot value independence, it does prevent conjecturing conflicting labels to equivalent paths.

5.7 Proof of Prop 5.3.2[†]

The proof of Prop 5.3.2 is fairly mechanical and mostly relies on two facts (i) trees only have a single path between any two nodes; and (ii) the gradient of LSE being the soft max distribution. The statement of Prop 5.3.2 is repeated below as a reminder.

Let $p_{xy}(\mathbb{V})$ denote the probability of starting at node x and pivoting at y, i.e.,

$$p_{xy}(\mathbb{V}) \overset{\text{def}}{=} \Pr(x \rightsquigarrow y \mid \mathbb{V}) \cdot \left(1 - \sum_{(y,z)\in E} \Pr(y \rightsquigarrow z \mid \mathbb{V})\right) \tag{5.14}$$

then,

$$\frac{\partial \hat{h}}{\partial \mathbb{V}_k} = \sum_{\substack{(i,j)\in E \\ i \text{ is ego}}} \#_{(i,j)} \cdot \Big(p_{ik}(\mathbb{V}) - p_{jk}(\mathbb{V})\Big) \tag{5.15}$$

Our proof will be centered on the following lemma.

Lemma 5.7.1. For any two nodes, i, k, in the prefix tree,

$$\frac{\partial}{\partial \mathbb{V}_k} \hat{V}_i = p_{ik}.$$

Proof. Let us first consider the where only a single action or state that pivots at k, i.e., $A_k = \emptyset$ or $S_k = \emptyset$. For any edge (a,b), observe that if a is an environment node, then $\Pr(a \rightsquigarrow b \mid \mathbb{V})$ is a constant, denoted q_{ab}. Next, observe that because the nodes are arranged as a tree either: (1) k is not reachable from i or (2) only a single edge, call (i,j), can reach k from i. Thus,

$$\begin{aligned} \frac{\partial \hat{V}_i}{\partial \mathbb{V}_k} &\overset{\text{def}}{=} \frac{\partial}{\partial \mathbb{V}_k} \sum_{\substack{(a,b)\in E \\ i=a}} q_{ib} \cdot \hat{V}_b(\mathbb{V}) \\ &= \Pr(i \rightsquigarrow j \mid \mathbb{V}) \cdot \begin{cases} 0 & \text{if } \Pr(i \rightsquigarrow k) = 0 \\ \frac{\partial}{\partial \mathbb{V}_k} \hat{V}_j(\mathbb{V}) & \text{otherwise,} \end{cases} \end{aligned} \tag{5.16}$$

Similarly, note that because the derivative of LSE is the softmax function, for any ego node i,

$$\begin{aligned} \frac{\partial \hat{V}_i}{\partial \mathbb{V}_k} &\overset{\text{def}}{=} \frac{\partial}{\partial \mathbb{V}_k} \log \sum_{\substack{(a,b)\in E \\ i=a}} \hat{V}_b(\mathbb{V}) \\ &= \begin{cases} 0 & \text{if } \Pr(i \rightsquigarrow k) = 0 \\ e^{\hat{V}_j(\mathbb{V}) - \hat{V}_i(\mathbb{V})} \cdot \frac{\partial}{\partial \mathbb{V}_k} \hat{V}_j(\mathbb{V}) & \text{otherwise,} \end{cases} \end{aligned} \tag{5.17}$$

where again, j denotes the (potential) unique child of i that can reach k. Next, observe that by definition $e^{\hat{V}_j(\mathbb{V})-\hat{V}_i(\mathbb{V})} = \Pr(i \rightsquigarrow j \mid \mathbb{V})$, using the maximum entropy policy induced by $\mathbb{V}$. Substituting into (5.17), we see that if k has only a single pivot action (or state) the lemma follows by induction on the path from i to k, where the base case is

$$\frac{\partial \hat{V}_k}{\partial_k \mathbb{V}_k} = \left(1 - \sum_{(y,z)\in E} \Pr(y \rightsquigarrow z \mid \mathbb{V})\right) \cdot \frac{\partial}{\partial_k \mathbb{V}_k}\mathbb{V}_k,$$

since probability of applying one of the pivot actions (leading to the forest of subtrees summarized by $\hat{V}_k$) as one minus the probability of traversing an edge in the subtree. □

Proof of Prop 5.3.2. Recall that the probability of traversing an environment edge is constant w.r.t $\mathbb{V}$. Thus, inspecting (5.4) we see that it suffices to prove that for any ego edge, (i, j),

$$\frac{\partial}{\partial \mathbb{V}_k} \ln \Pr(i \rightsquigarrow j \mid \mathbb{V}) = p_{ik} - p_{jk}.$$

Recall that by definition, if i is ego, then $\ln \Pr(i \rightsquigarrow j \mid \mathbb{V}) = \hat{V}_j(\mathbb{V}) - \hat{V}_i(\mathbb{V})$. Thus, the proposition follows directly from Lemma 5.7.1. □

5.8 Bibliographic Notes

The key difficulty for the task specification inference from demonstrations literature is how to search an intractably large (often infinite) representation class.[5] In particular, and in contrast to the reward setting, the discrete nature of automata and logic, combined with the assumed a-priori ignorance of the relevant memory required to describe the task, makes existing gradient based approaches either intractable or inapplicable. Instead, current literature either (syntactically) enumerates tasks [149, 37, 131, 161] or hill climbs via simple probabilistic (syntactic) mutations [84, 24].

The major contribution of this chapter was to systematically reduce the problem of learning from demonstrations into a series of supervised task specification identification problems, e.g., finding a small DFA that is consistent with a set of example strings [70], a problem more generally referred to as Grammatical Inference [45].

The structure of the resulting algorithm DISS, is inspired by two more established families of algorithms: simulated annealing (SA) [134] (for *guided* hill climbing) and Oracle-guided inductive synthesis[6](OGIS)[129, 81] (for gradually constraining a search based on previous guesses). While the connection with SA is made explicit in this chapter, the OGIS connection is largely left unexplored.

[5]This intractable size is arguably the reason that this space of problems has been underserved until now.

[6]OGIS being a generalization of the prior work on Counter Example Guided Inductive Synthesis (CEGIS) [135].

OGIS operates by alternating communication between a learner and a verifier. The learner proposes candidates and the verifier checks whether there is something wrong with the candidate, e.g., violates some specification. If so, the learner is provided with an explicit counterexample to take into account for its next candidate. At a distance, one observes that DISS operates in largely the same way. The identification algorithm acts like the learner and SGS acts like the verifier. The key difference is that the provided examples are not (provably) counterexamples. Nevertheless, these pseudo counterexamples ostensibly play the same role. They constrain the next candidate. The role of SA in DISS is to account for the fact that these are pseudo counterexamples, and thus must not be taken as facts.

OGIS also forms the basis for the relaxation of the Luce axiom violation. One can view the act of running DISS and checking for value-distinguishability as a form of verification (or conformance testing) and the process of creating a equivalence class as the learner.

Finally, the energy function minimized by DISS is equivalent to maximizing the data compression [3]. In this way, we can view DISS as looking for more and more succinct representations of the demonstrations - taking care to balance the description complexity of both the encoding and the encoded paths.

Chapter 6

Teaching Tasks using Demonstrations

The more I think about language, the more it amazes me that people ever understand each other at all.

Kurt Gödel, (Logician, 1906-1978)

In the last chapter, we argued that demonstrations offer a rich and ergonomic means for a teacher to communicate task specifications to a learner. In this chapter, we flip this on its head and ask how to generate demonstrations that help a learner infer a given task.

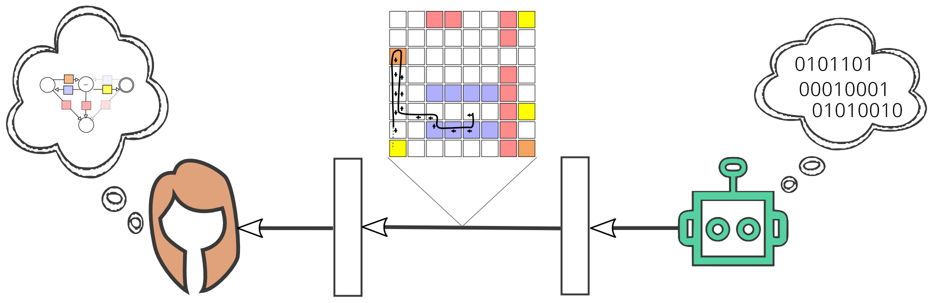


Figure 6.1: Illustration of providing pedagogic demonstrations to help a human learn a task.

There are a number of benefits of changing perspective from a learner to a teacher. Firstly, formal task specifications can be difficult to understand, even for domain experts. Thus, being able to supplement formal or natural language descriptions of a task with carefully selected demonstrations can be invaluable. Second, alternating teaching and learning forms a foundation for future work on organic team-based collaboration, where team members automatically specialize given inferred assumptions about other agents. Finally, teaching provides further academic insight into what kind of demonstrations are helpful for learning models.

A natural question then is how to characterize and algorithmically generate pedagogic demonstrations. As we'll discuss at the end of the chapter, the perspectives in this chapter are inspired by the literature on pragmatics [62], legibility [48], and showing vs doing [73]. The common idea across these works is for the teacher to simulate the learner in order to provide demonstrations that make the true reward easier to infer.

Contributions

We will study how to make the demonstrations *surprising* under tasks we do not wish to teach and *unsurprising* under the ground truth task. This is made explicit by providing a re-writing of the showing model [73] in terms of description lengths. To handle large spaces of task specifications, e.g. automata, we provide a counterexample driven algorithm (using DISS as a surrogate learner) for finding pedagogic demonstrations. Using DISS as a proxy for the learner, offers several benefits:

1. Understanding how to provide pedagogic demonstrations to DISS will offer further insights into how effective DISS can be in practice.
2. DISS is designed to make minimal assumptions about the teacher, and thus teaching to DISS requires generating demonstrations that are pedagogic across a variety of learners.

The result will be an algorithm that automatically generates demonstrations that help the learner understand a task. Finally, we provide the results of a human study that provides empirical evidence for the efficacy of our teaching algorithm.

6.1 Pedagogic Demonstrations

To begin, we must define what it means for a demonstration to be pedagogic. For inspiration, we turn to the literature on showing versus doing [73], which offers empirical evidence that human teachers provide demonstrations, $\Xi = \xi^*_1, \ldots \xi^*_m$, proportional to the belief that the learner assigns to the ground truth task, i.e.,

$$P_{\text{teacher}}(\Xi \mid \varphi) \propto P(\varphi \mid \Xi) = \frac{P(\Xi \mid \varphi) \cdot P(\varphi)}{\sum_{\varphi'} P(\Xi \mid \varphi') \cdot P(\varphi')} \tag{6.1}$$

where P is the belief the learner assigned to task, φ. Concretely, P will take the form of a DISS learner from the previous chapter.

Remark 6.1.1. The name "showing vs doing" derives from contrasting (6.1) with the teacher that is just "doing" the task, e.g., sampling paths using π_φ. In our running example, a "doing" teaching might providing two demonstrations that start near and go directly to the bottom yellow tile. By contrast, as we shall see "showing" teaching is likely to provide one demonstration that illustrates that visiting 🟦 requires visiting 🟧 and another demonstration that shows that visiting 🟦 is optional.

Remark 6.1.2. The showing vs doing framework is actually defined for arbitrary sets of Markovian reward functions. The technique developed here is biased towards task specifications, but should in principle work with arbitrary rewards. As with learning, the key difficulty in the task specification setting is that task specifications are discrete combinatorial objects. Thus, it is unclear how to teach against the whole set simultaneously.

Description length perspective

Equation (6.1) implies that the teacher is biased towards demonstrations that are unsurprising under φ, but surprising under φ'. To make this explicit, we re-write (6.1) by taking the negative log and rearranging:

$$\begin{aligned}
-\ln P_{\text{teacher}}(\Xi \mid \varphi) &= -\ln \Big(P(\Xi \mid \varphi) \cdot P(\varphi)\Big) + \ln \sum_{\varphi'} e^{\ln(P(\Xi|\varphi')P(\varphi'))} \\
&= -\ln P(\Xi \mid \varphi) - \ln P(\varphi) + \operatorname*{LSE}_{\varphi'} \Big(\ln P(\Xi \mid \varphi') + \ln P(\varphi')\Big) \\
&\overset{\text{def}}{=} U(\varphi, \Xi) + \operatorname*{LSE}_{\varphi'} \Big(-U(\varphi', \Xi)\Big) \\
&= \operatorname*{LSE}_{\varphi'} \Big(U(\varphi, \Xi) - U(\varphi', \Xi)\Big),
\end{aligned} \tag{6.2}$$

where similar to the previous chapter, we denote by U the description length (or energy): $-\ln P(\Xi \mid \varphi) - \ln P(\varphi)$. Recalling that LSE is a smoothed version of the max function, we see that minimizing (6.2) (maximizing (6.1)) implies making energy of φ small and the energies of other tasks φ' large.

6.2 Generating Pedagogic Demonstrations

We now turn to the question of how to find the most probable teaching demonstrations given (6.1). In particular, we are interested in approximately solving the following optimization problem:

$$\min_{\Xi \in \text{Paths}_{\$}^{m}} \operatorname*{LSE}_{\varphi' \in \mathcal{R}} \Big(U(\varphi, \Xi) - U(\varphi', \Xi)\Big). \tag{6.3}$$

There are two obstacles to directly optimizing (6.3). First and foremost, the LSE is taken over the entire representation class.[1]Second, the transition system and paths are discrete objects with no obvious gradient. As we will see, this second obstacle is easily overcome by optimizers such as simulated annealing. To overcome the first obstacle, observe that if one fixes the demonstrations Ξ, then (6.3) reduces to:

$$\operatorname*{LSE}_{\varphi' \in \mathcal{R}} \Big(-U(\varphi', \Xi)\Big) \approx \min_{\varphi' \in \mathcal{R}} U(\varphi', \Xi), \tag{6.4}$$

[1]This LSE is well defined since the there cannot be a uniform distribution on countably infinite sets. Thus, the $-U(\varphi, \xi)$ tends to negative infinity.

where the approximation follows from LSE being a smooth approximation of max. Observing that this is exactly the energy minimization problem solved by DISS in the previous chapter, we arrive at a simple iterative algorithm for generating pedagogic demonstrations.

Generating Pedagogic Demonstrations:

1. Let n, k, n_{DISS}, and m be natural numbers.
2. Let $\hat{\mathcal{R}} = \{\varphi\}$ denote a set only containing the target task, φ.
3. Sample m demonstrations, Ξ_0, using π_φ.
4. For $i = 1 \ldots n$:
    a) Run DISS given Ξ_i for n_{DISS} iterations yielding a set $\mathcal{R}_i$.
    b) Add the k most likely tasks (under Ξ_i) from $\mathcal{R}_i$ to $\hat{\mathcal{R}}$.
    c) Find a set of demonstrations Ξ_i that approximately minimizes:

$$\operatorname*{LSE}_{\varphi' \in \hat{\mathcal{R}}} \Big(U(\varphi, \Xi_i) - U(\varphi', \Xi_i) \Big)$$

5. Return Ξ_n.

This algorithm operates by alternating between generating pedagogic demonstrations for the $\hat{\mathcal{R}}$ set and then expanding the $\hat{\mathcal{R}}$ set using DISS. Intuitively, $\hat{\mathcal{R}}$ is made up of specifications found by DISS that have non-trivial probability mass given a previous Ξ_i. Thus, it is the set of task specifications the teacher should try to keep in mind when generating demonstrations.

Remark 6.2.1. This algorithm can be viewed as a form of counter-example guided synthesis [135], where $\mathcal{R}_i$ acts "proof" that the current demonstration might imply another set of tasks.

Trajectory Optimization

The only component missing from the above algorithm is a means to implement step 4c, i.e., finding demonstrations that minimize:

$$U(\Xi) \overset{\text{def}}{=} \operatorname*{LSE}_{\varphi' \in \hat{\mathcal{R}}} \Big(U(\varphi, \Xi) - U(\varphi', \Xi) \Big).$$

Depending on the specific dynamical system under consideration, many techniques ranging from reinforcement learning [140] to convex programming [95] can be used. Below, we provide and motivate a simple approach, which like DISS is based on simulated annealing.

Recall that to define an instance of simulated annealing, one requires an energy U to minimize, a proposal distribution q, a cooling schedule, and a reset trigger. The energy in this case is $U(\Xi)$ as defined above. For simplicity the reset occurs periodically every 50 iterations and the cooling schedule linearly decays from temperature $T_{\max} = 100$ to 1 just like in DISS. Finally our proposal distribution, $q(\Xi' \mid \Xi)$ operates as follows:

Proposal Distribution:

1. Sample a path, ξ, uniformly at random from Ξ.
2. Sample a pivot, ρ, uniformly at random from the prefix tree of ξ.
3. Using π_φ sample a (positive) complete path, $\xi' \in \varphi$, that pivots at ρ.
4. Replace ξ with ξ', i.e., $\Xi' = (\Xi \setminus \{\xi\}) \cup \xi'$.

This proposal distribution was designed with the observation that, ignoring other tasks, $U(\varphi, \Xi)$ should be minimized. Thus, we propose demonstrations from π_φ which are biased towards small description lengths (U). We then rely on the hill climbing of simulated annealing to account for other specifications. The random pivots provide two types of changes. Small refinements by pivoting near the end of the demonstration and large changes by pivoting near the start. Note however that because we sample from π_φ, even large changes may often take similar paths, e.g., multiple ways to reach the same bottleneck. Finally, to allow changing the initial location, we apply the standard transformation on the MDP where a dummy start state is included and any action from this state uniformly transitions the agent to a location in the MDP. Thus, pivoting on the first state (environment action) becomes equivalent to changing starting locations.

Failure Modes

One should naturally wonder what pit-falls await practical implementations of the above algorithm. Below we briefly discuss two particularly problematic issues, along with potential remedies.

Competency Estimation

Recall that the showing distribution (6.2) biases towards demonstrations that have short *relative* description lengths, i.e., small under φ and large under other specifications. This decoupling from total description length can yield demonstrations where the teacher's behavior seems undirected, even bordering on incompetent. For instance, in our running example, the teacher could move back and forth within the corridor of blue tiles, 🟦, to emphasize that despite multiple opportunities to do so, it does not cross 🟦. Unfortunately, particularly in the context of human learners, the teacher's actions may seem undirected, and it may not be as obvious that the goal is to eventually reach a yellow tile, 🟨. This can be mitigated in a multitude of ways. Below we provide three examples.

1. In the literature on legibility [48], a predictability term is added which penalizes demonstrations that are unlikely under the expert policy. Under the lens of description lengths, the result is re-weighting our energy function by the introduction of a hyper-parameter $\alpha > 1$ as below:

$$U(\Xi) \overset{\text{def}}{=} \underset{\varphi' \in \hat{\mathcal{R}}}{\text{LSE}} \left(\alpha \cdot U(\varphi, \Xi) - U(\varphi', \Xi)\right).$$

Recalling the relative entropy (expected relative description length) is the cross-entropy minus the entropy, we see that α has the effect of interpolating between minimizing the total and relative number of nats needed to describe Ξ. This means the teacher is incentivized to produce directed behavior since directed behavior has smaller total description length.

2. An alternative mitigation is to directly model the learner as simultaneously estimating the competency (temperature) and the task specification. In this formulation, undirected behavior is explicitly penalized since the result is a larger inferred temperature. As the temperature rises, π_φ approaches the uniform distribution. Thus, all specifications approach the same description length which maximizes (rather than minimizes) the energy!

3. Finally, perhaps the simplest mitigation is to include a sufficiently rich class of task specifications. For instance, if there is a high a priori weight on a task specification that is trivially false or true, the teacher needs to distinguish itself *against* uniformly random actions. Furthermore, if the concept class is sufficiently expressive, then the learning algorithm will return many distinct tasks that the previous demonstration failed to distinguish. Together, the teacher is incentivised to not overly optimize teaching against any specific task specification, which effectively penalizes "undirected" behavior. Note that the success of this mitigation relies on multiple alternations between teaching against $\hat{\mathcal{R}}$ and expanding $\hat{\mathcal{R}}$ with DISS.

In our implementation, we rely on mitigation 3 since minDFA is sufficiently expressive and has a high prior on the two one state DFAs representing trivially true and false tasks.

Handling Similar Specifications[†]

An insidious issue arises when the teacher is asked to teach a task that is effectively equivalent to many of the tasks in the representation class. In such settings, $\hat{\mathcal{R}}$ will contain many tasks, $\varphi_1, \ldots, \varphi_j$, that are similar to φ. Assuming that $U(\varphi, \Xi) \approx U(\varphi_i, \Xi)$ and observing that $U(\varphi, \Xi) - U(\varphi_i, \Xi) \leq 0$, we see that the LSE over $U(\varphi, \Xi) - U(\varphi_i, \Xi)$ becomes less and less sensitive to the other tasks in $\hat{\mathcal{R}}$. The result is that the teacher loses its incentive to provide pedagogic demonstrations. Fortunately, if there is a way to detect that φ and φ_i are essentially equivalent, then one can quotient $\mathcal{R}$ by this (approximate) equivalence relation.

For example, let φ denote reaching 🟨 and let φ' denote reaching 🟨 at least twice. If reaching 🟨 requires visiting at least two such states, e.g., due to a one way door, then the policies for these two tasks are indistinguishable. One might then combine any states whose policies have negligible KL-divergence, i.e.,

$$\Big(\forall \xi \in \text{Paths}_\$ \;.\; D_{\text{KL}}\big(\pi_\varphi(\bullet \mid \xi) \,||\, \pi_{\varphi'}(\bullet \mid \xi)\big) \leq \epsilon\Big) \implies (\varphi \approx_\epsilon \varphi'), \tag{6.5}$$

for some $\epsilon \geq 0$.

6.3 Experiments

To get a qualitative sense of how our proposed algorithm works, we instantiated our teaching algorithm for variants of our running gridworld example.

Teaching Parameters

We took the number of demonstrations, m, and the number of DFAs added to $\hat{\mathcal{R}}$ per iteration, k, to be 2. The learner used was the same DFA based DISS implementation from Ch 5, with $\beta = 1/4$ and $n_{\text{DISS}} = 35$. This choice of hyperparameters was informed by the learning rates in Fig 5.9a. The trajectory optimizer (based on simulated annealing) was run for 350 iterations, and thus experienced 7 resets. Finally, the teaching algorithm was run for 3 iterations, with early stopping once the correct DFA was inferred or after 15 minutes had elapsed. As a baseline, we ran the teaching algorithm with $n = 0$, which results in m samples using π_φ. We refer to these cases as **showing** and **doing** respectively.

Tasks to Teach

We ran our teaching algorithm on 5 different gridworld-task pairs (shown in Fig 6.2.) The top row provides the tasks encoded as DFAs. The middle and bottom rows illustrate the generated showing and doing demonstrations for the DFA in their respective column. To avoid clutter, we only annotate the action (with a green arrow indicating intended direction) when the agent slips downward due to wind. Below we provide natural language descriptions of each task, enumerated from left to right.

1. Go to 🟨. But, if you visit 🟥 or 🟧, you must visit 🟦 before 🟨.
2. Go to 🟧 and then 🟥 (in that order) while avoiding 🟨.
3. Go to 🟥 and then 🟨 (in that order) while avoiding 🟧 and 🟦.
4. Go to 🟥. Avoid 🟨. If you visit 🟧 you must visit 🟦 before 🟨.
5. Go to 🟨. Avoid 🟥. If you visit 🟦 you must visit 🟧 before 🟨.

Qualitative Results

Analyzing the generated demonstrations, a number of qualitative differences between the doing and showing demonstrations become apparent. First and foremost, one observes that the doing demonstrations are generally able convey which colors must always be visited and avoided, but do not provide evidence for conditional logic. For example, our running example (shown in the right most column), the doing demonstrations clearly illustrate the need to go to 🟨, but provides little to no evidence that the agent needs to visit 🟦 after visiting 🟧. By contrast, the corresponding showing demonstrations illustrate this behavior. Moreover the showing demonstrations provide evidence that 🟥 needs to be avoided since both demonstrations ignore the shorter path to the top right 🟨. On the other hand, for tasks that

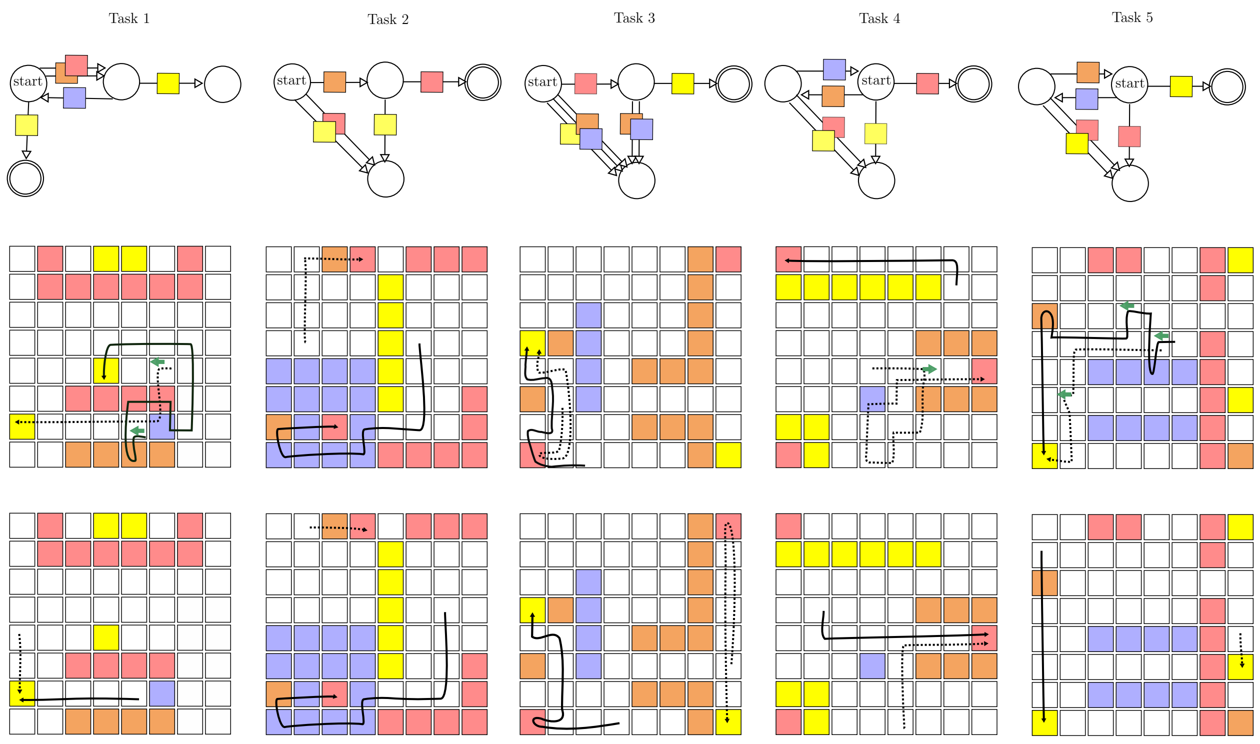


Figure 6.2:
Top row: Target tasks to teach encoded as DFAs.
Middle row: Pedagogic demonstrations for the task DFA in the same column.
Bottom row: Demonstrations from expert just performing the task in the same column.

only require visiting a fixed sequence of colors and avoiding other colors, e.g. the second and third tasks (from the left), the showing and doing demonstrations are qualitatively similar.

The next qualitative difference is that the showing examples actively exploit the slipping feature of the dynamics to encode additional information. This can be seen in the first, fourth and fifth tasks, where the conditional visitation behavior is triggered "by accident". This accidental triggering enables the action to hint at the intended path to an accepting state in the DFA. Again returning to our running example (fifth column), slipping is employed to illustrate that the agent did not intend to visit 🟦 and simultaneously that it did not intend to visit 🟥. A similar effect occurs in the fourth task, where the agent is forced to visit 🟦 and after slipping into 🟧. Slipping here makes it clear that the intention was to go through the corridor of 🟧 to reach 🟥. Finally, we see a similar trick being employed in the showing demonstrations for the first task to highlight that visiting 🟧 and 🟥 trigger conditional behavior.

Limitations

The first task also highlights some of the limitations of the approach as formulated. Here the demonstrations are fairly complex and require a non-trivial amount of thought to recover the intended task. In fact, one might argue that the fact that visiting 🟧 triggers visiting 🟦 is arguably obscured by the agent visiting 🟥. To address these concerns, one might consider adding additional demonstrations, but no principled way to determine the optimal number of demonstrations is provided in our algorithm. Similarly, one might try to enforce diversity in the way the demonstrations traverse through the task DFA, but it is unclear how to enforce this in general. Nevertheless, this experiment highlights that teaching against DISS enables generating demonstrations the better convey the nuances of target task than our doing baseline. Importantly, this is the case even when the representation class is too large to enumerate and only moderately structured.[2]

Human Study

To further test the efficacy of the above teaching algorithm, we ran an online human study using the Prolific platform [119]. We recruited 250 participants. The participants were recruited and told that:

1. They would be making judgments about the rules a robot follows.
2. The robot would be operating in a gridworld with the same dynamics as our running example.
3. There could be any number of rules.
4. There were three types of rules: (i) Go to color; (ii) Avoid color; (iii) If on color 1, then go to color 2.
5. The color white was not part of any rule.
6. A path is good if all the rules hold and bad if any of the rules do not hold.

Remark 6.3.1. The described representation class is expressive enough to capture many of the tasks in minDFA and in particular all DFAs in Fig 6.2. Further, note that this is an infinite representation class.

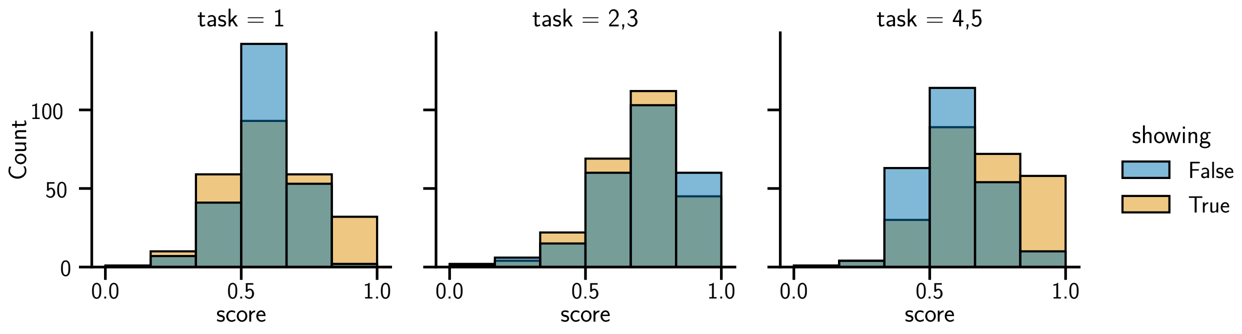
Next, each participant was blindly assigned to either be given showing demonstrations or doing demonstrations. The participants were then asked to perform a series of trials. Each trial corresponded to one of task specification gridworld pairs shown in Fig 6.2. Within the trial the participants where shown animations of either the showing demonstrations or the doing demonstrations - depending on their earlier assignment. After seeing the demonstrations, the participants were asked to describe the rules of the robot. The trial ended with the participants being asked to label five color sequences as "good" or "bad" under the inferred rules. We assigned the trial a score between zero and one denoting the fraction of correct answers. Finally, the color and order of the tasks were randomized. The colors used were:

[2]In particular, "similar" DFAs might have remarkably different languages.

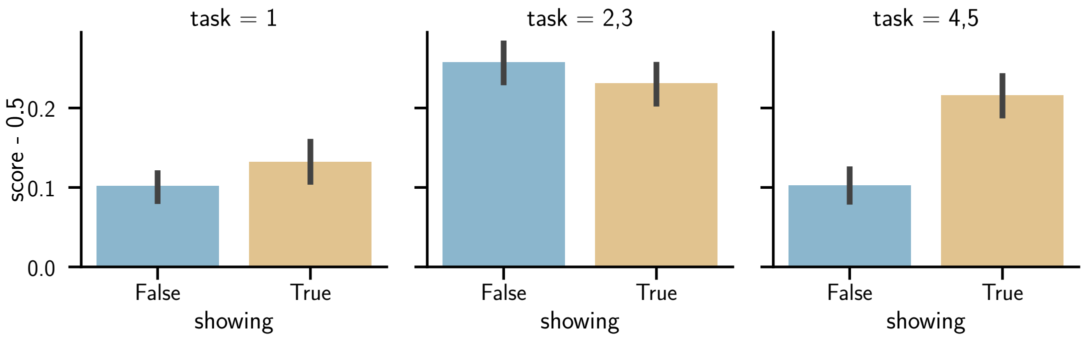
pink, blue, brown, and turquoise, and were selected to avoid a priori associations with colors, e.g., red is bad/good.

Human Study Results

To study the data generated by our human study, we partitioned the five trials into three groups: $\{1\}, \{2, 3\}, \{4, 5\}$. Again, the trial numbers correspond to the column in Fig 6.2, with the leftmost column being 1 and the rightmost column being 5. This partitioning of trials derived from our earlier analysis. In particular, recall that (i) trials 2 and 3 correspond to tasks with no conditional logic; (ii) trials 4 and 5 correspond to tasks with conditional logic; and (iii) trial 1 has conditional logic, but we argued not enough demonstrations were provided. In particular, there are two different triggers for visiting 🟪. Three demonstrations (or longer demonstrations) would have allowed exercising all three of these paths.



(a) Distribution of scores in human study by trial group and treatment.



(b) Average score by trial group and treatment.

Figure 6.3: Results of human study. Here, showing = False means the partipant was in the doing treatment and showing = True means the participant was in the showing treatment.

The distribution of scores is shown in Fig 6.3a and the average score (with annotated 95% confidence intervals) is shown in 6.3b. In both figures, we see confirmation of our earlier analysis. For example, the showing treatment did not significantly change the score distribution for trials without conditional logic (2 and 3). Conversely, the showing treatment did affect the distributions for trials with conditional logic, e.g., for trials 4 and 5, the overall distribution was shifted towards higher scores. For trial 1, we observe a larger variance. We suspect this variance is due whether or not the participant correctly inferred that visiting brown triggers a visit to blue as opposed to brown triggers a visit to red (as might be suggested by the demonstration.)

Remark 6.3.2. The key take away is that our teaching algorithm can indeed help the particpants learn task specifications; however, care should be taken to provide enough demonstrations. Further, we note that this was the case even when the representation classes differed.

6.4 Bibliographic Notes

As emphasized throughout the chapter, the teaching algorithm described builds upon previous work generating pedagogic demonstrations for teaching quantitative reward functions [73]. This and related frameworks such as legibility [48] rely on the ability for the teacher to simulate the way the learner would respond to a given demonstration. Of course, this process is naturally recursive, a phenomena codified in the Rational Speech Act (RSA) modeling framework [61]. In RSA, the teacher and learner recursively model each other. The teacher discussed in this chapter has two alternations between teacher and learner, i.e., the showing teacher simulates the learner and the learner simulates the doing teacher. The key novelty in this work is the ability to teach against a combinatorial representation class such as DFAs. This is facilitated by the innovations in the DISS algorithm, which simulated a learner that tries to minimize the description complexity of the demonstrations. Furthermore, like DISS, the final algorithm provided is again a OGIS [129, 81] style algorithm in which one component proposes a candidate (here the demonstrations) and the other component (here DISS) returns potentially problematic inferences made by the simulated learner. Finally, the similarity failure mode discussed in Sec 6.2 has been observed in the reward setting, with a similar re-weighting of the LSE being applied to account for similar rewards [18].

Part II

Prediction and Control

In Part I, we assumed access to a maximum causal entropy planner. This planner was used black-box to predict the actions of an agent trying to perform a given task. Using these predictions, we showed how to learn and teach specifications using demonstrations. In this part, we open the black-box and investigate how to efficiently realize such planners when the workspace can be modeled as a probabilistic sequential circuit.

This part is structured as follows. Ch 7 starts with a discussion of control improvisation - the decision problem corresponding to entropy regularized planning in Markov Decision Processes, Interval Markov Decision Processes, and more generally, Stochastic Games. The result will be planning algorithms that scale in the size of the time-unrolled stochastic game. To keep the size of the time-unrolled games tractable, we change perspective in Ch 8 and explore modeling probabilistic transition systems as probabilistic circuits. This sets the stage for Ch 9 in which we discuss storing compressed representations of probabilistic transition systems using Binary Decision Diagrams (BDDs). In particular, we shall see that the maximum causal entropy planning can be done directly on this compressed structure!

Chapter 7

Improvisation in Stochastic Games

Improvisation is too good to leave to chance.

Paul Simon (Musician, 1941-)

In this chapter, we study how to compute maximum causal entropy policies given a planning horizon and a task specification. Technically, this will be done by framing the problem as instance of control improvisation [55, 54] over stochastic games. In the control improvisation framework, one specifies a controller with three types of declarative constraints. (i) *Hard constraints* that, as in the classical setting, must hold on every execution, (ii) *soft constraints* that should hold on most executions, and (iii) *randomization constraints* that ensure that a synthesized policy does not overcommit to a particular action or behavior. The key challenge when solving control improvisation is that randomization and performance, in the form of soft constraints, constitute a natural trade-off.

Control improvisation was originally proposed for nondeterministic domains where uncertainty is resolved adversarially. This assumption is often too restrictive and leads (together with the soft/hard constraints) to conservative policies or common situations in which the synthesis algorithm cannot be employed at all. To overcome this weakness, in this chapter, we develop a theory of control improvisation in stochastic games which admit arbitrary *combinations* of nondeterministic and probabilistic uncertainty, including unknown or imprecise transition probabilities.

Technically, we formulate our problem on *simple stochastic games* [39], an extension of *Markov decision processes* (MDPs) that divides states between controllable states and uncontrollable (or adversarially controlled) states. *Soft constraints* are finite horizon temporal properties with a threshold on the worst-case probability of the property holding by the end of the episode. *Hard constraints* are soft constraints to be satisfied with probability 1. In contrast to other work on control improvisation, we adopt causal entropy as a natural means to formalize *randomness constraints.* In doing so, we are able to realize the entropy regularized planners required by the learning and teaching algorithms in Part I.

We refer to this variant of control improvisation as *Entropic Reactive Control Improvisation* (ERCI) and show that ERCI conservatively extends reactive control improvisation [55] to stochastic games. More precisely, entropy can be used in the non-stochastic setting and yields results analogous to reactive control improvisation. ERCI also extends classical policy synthesis in stochastic games, i.e. synthesis in absence of randomness constraints as, e.g., implemented in PRISM-games [89].

Contributions

In summary, this chapter provides Entropic Reactive Control Improvisation (ERCI), a decision variant of entropy regularized planning specialized for task specifications in stochastic games. ERCI provides an algorithmic way to trade performance and randomization in stochastic games. Stochastic games combine both adversarial and probabilistic behavior in an environment, enabling modeling flexibility, which facilitates model compression and applicability to new domains, e.g., using probability ranges rather point probabilities to combine similar states/actions or explicitly capture uncertainty in the dynamics model. Finally, this work contributes the necessary technical machinery and a prototype implementation. Combined, our theoretical and empirical analysis suggest that the ERCI framework contributes a tractable and flexible modeling formalism.

Overview

This chapter is structured as follows. We begin with a motivating example (Sec. 7.1). This motivating example will highlight the potential application of ERCI in domains outside behavior predication, and in particular, the generation of random behavior for testing systems. Furthermore, this example will demonstrate the modeling flexibility derived from supporting stochastic games. Next, we provide preliminaries and formalize the ERCI problem statement in Sec. 7.2, cast ERCI as a multi-objective optimization problem, and study properties of the solution set in Sec. 7.3. With this technical machinery developed, Sec. 7.4 re-frames existing literature on maximum causal entropy inference and control to derive an algorithm for MDPs. Then in Sec. 7.5, we provide an algorithm for the general case of stochastic games. We conclude with an empirical evaluation (Sec. 7.6) and a comparison with related work, e.g., other control improvisation formulations (Sec. 7.8). Proofs are provided in Sec. 7.7.

7.1 Motivating Example

We consider a scenario in which a regulatory agency wishes to certify the safety and performance of a new delivery drone D_{new}. As part of the process, the agency runs D_{new} through a series of tests. For example, given a certain delivery route, the agency investigates whether D_{new} successfully delivers packages while avoiding *other* delivery drones. To execute this test,

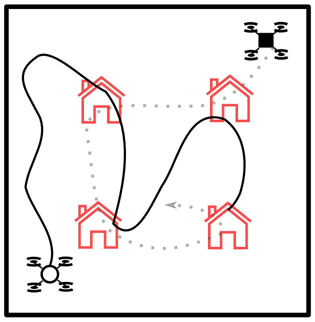


Figure 7.1: Illustration of delivery drone testing example. The goal is to synthesize a policy for the bottom left (white circle) drone to test the controller of the top right (black square) drone. Ideally, the synthesized policy should be as randomized as possible to avoid testing bias.

the agency decides to synthesize a controller for another delivery drone, D_{test}, to test if D_{new} can be certified.

Concretely, suppose we command D_{new} to continuously visit four houses in some workspace. We illustrate such a scenario in Fig. 7.1, in which D_{new} and D_{test} are shown as black square and white circle drones respectively. For this test scenario, the regulatory agency, wishes to examine how D_{new} responds to delivering packages to the red houses in the presence of D_{test}. In particular, it would like to let D_{test} also deliver packages while avoiding D_{new}. Importantly, to properly exercise D_{new}, D_{test} should show a *variety* of behaviors meeting the specification, and the behaviors should not be biased to any behavior beyond the given specification.

With the ERCI framework, the agency may formalize the above scenario with the following constraints on D_{test}:

1. (*hard constraint*) Ensure that the two drones *never* collide.
2. (*soft constraint*) With probability at least .8, visit all four houses within 10 minutes.
3. (*randomness constraint*) Perform this task as unpredictably as possible.

What remains is to synthesize a controller given the constraints *and* the world model. At this point, it is worth examining more closely how one models D_{new}'s controller when synthesizing D_{test}. We illustrate by examining three models. In all models, we capture the behaviors of D_{new} and D_{test}. We focus on D_{new}, but the ideas carry over to modeling the actuation of D_{test}.

Nondeterministic Model

The simplest approach to modeling is not to make any assumptions about D_{new} beyond what already has been established. Here, we model that the houses are visited either in clockwise or counter-clockwise order but that it may switch direction at *any time*. Such a model is too liberal and our assumptions under which we plan the behavior for D_{test} is too pessimistic, which leads to a bad test set. First, if D_{new} is unrestricted, then D_{test}'s behavior is severely limited, as it must behave conservatively to avoid collisions under all possible motions by D_{new} (even very unlikely motions). This limitation restricts the variance of its behavior, and it will not test D_{new}'s true behavior. A purely non-deterministic model for D_{new} thus may not lead to the synthesis of adequate behavior for D_{test}.

Stochastic Model

Rather than the pessimistic nondeterministic (or adversarial) assumption, we may collect data about D_{new} and construct a stochastic model, e.g., using inverse reinforcement learning [111]. Concretely (but simplified), after examining the data, one observes that D_{new} appears to flip a biased coin with fixed probability p whenever it reaches a house to decide whether or not to turn around. This models D_{new} much more precisely, and allows for more targeted test by D_{test}.

Nondeterministic and Stochastic Model

However, a natural criticism for stochastic models is the dependence on *fixed* probabilities. Obtaining such probabilities with confidence requires many tests which defeat the purpose of our test setup, and making point-estimates from little data may not create faithful models of the actual behavior. In absence of enough (or reliable) data, we can arbitrarily combine nondeterministic choices and stochastic behavior. We may use stochastic abstractions for parts that we can faithfully model, and nondeterministic behavior in absence of data. In particular, we support interval-valued transition probabilities. Consider the delivery-drone D_{new}. Rather than inferring a point-estimate from data, we may have inferred that the probability of turning around is in the interval $[p - \varepsilon, p + \varepsilon]$ for adequate values of p and ε. Furthermore the actual probability may even depend on aspects of the current state.

ERCI as a unifying framework

The strength of the (entropy-guided) control improvisation framework is that we can combine all these aspects into a single and thus flexible computational model. In particular, the models above are captured by a 2-player game, a 1.5-player game (MDP) and a 2.5-player game (stochastic game, SG), respectively. In all cases, the first player controls the behavior of D_{test} and this controller is to be synthesized. We contribute an algorithm that synthesizes a controller that maximally randomizes in all of the formalisms discussed above. In the coming sections, we shall formally define the ERCI problem, highlight that there is an implicit trade-off between performance of the soft constraint and unpredictability, and provide an algorithm solving ERCI for SGs.

7.2 Problem Statement

This section formalizes the novel Entropic Reactive Control Improvisation (ERCI) problem.

Stochastic Games

We start with some necessary definitions and notations on stochastic games which generalize the MDPs defined in Ch 3.

Definition 26. A (2.5-player) *stochastic game* (SG) is a tuple $\mathcal{G} = \langle S, s_0, A, P\rangle$. The finite set of *states* $S = S_{\mathsf{ego}} \cup S_{\mathsf{env}}$ is partitioned into a set S_{ego} of (controlled) ego-states and a set S_{env} of (uncontrolled) env-states. $s_0 \in S_{\mathsf{ego}}$ is the *initial state*, A is a finite set of *actions*, and $P\colon S \times A \to \mathit{Distr}(S)$ is the *transition function*. For simplicity of exposition, we assume w.l.o.g. that controlled and uncontrolled states alternate. Thus, P is defined by two *partial* transition functions: $P_{\mathsf{ego}}\colon S_{\mathsf{ego}} \times A \to \mathit{Distr}(S_{\mathsf{env}})$, $P_{\mathsf{env}}\colon S_{\mathsf{env}} \times A \to \mathit{Distr}(S_{\mathsf{ego}})$. We identify the available actions[a] as $A(s) \overset{\text{def}}{=} \{a \mid P(s,a) \neq \bot\}$. States without available actions, i.e., states with $A(s) = \emptyset$ are called *terminal states*. The *successor states* of a state s and an (enabled) action a is the set of states that are reached from s within one step with a positive transition probability, i.e., $\mathsf{Succ}(s,a) \overset{\text{def}}{=} \{s' \mid P(s,a)(s') > 0\}$, and $\mathsf{Succ}(s) \overset{\text{def}}{=} \bigcup_{a\in A(s)} \mathsf{Succ}(s,a)$.

[a]We use a partial function as we explicitly allow modeling unavailable actions, e.g., we can model that a door can only be opened when close enough to the door.

Example 7.2.1. *We introduce a six-state toy-example (Fig. 7.2) to illustrate the definitions. Terminal states are drawn with a rectangle,* ego*-states with a circle and* env*-states with a diamond. For every state s and action a, we draw transitions in the form of edges that connect all successors s', and label them with the associated probabilities $P(s,a)(s')$. For conciseness, we omit labelling probability* 1 *transitions.*

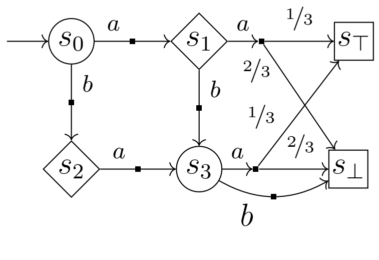


Figure 7.2: A running example.

Remark 7.2.2. SGs capture a variety of models. For example, if $|A(s)| = 1$ for all uncontrolled states, $s \in S_{\mathsf{env}}$, then $\mathcal{G}$ is a *Markov decision process* (MDP). If $|A(s)| = 1$ for all $s \in S$, then $\mathcal{G}$ is a *Markov chain.* If $P(s, a)$ is a Dirac distribution for every $s \in S$ and $a \in A$, then $\mathcal{G}$ is called *deterministic* or a *2-player game.*

Remark 7.2.3. Paths are defined just like in MDPs, except that that ego states are even indexed and env states are odd indexed - as we assume alternation. Additionally, it is helpful to partition paths based on their last state: $[\mathsf{Paths}]_{\mathsf{ego}} = \{\xi \in \mathsf{Paths} \mid \text{last}(\xi) \in S_{\mathsf{ego}}\}$ and $[\mathsf{Paths}]_{\mathsf{env}} = \mathsf{Paths} \setminus [\mathsf{Paths}]_{\mathsf{ego}}$.

Example 7.2.4. *In Fig. 7.2, there are two paths that end in s_3, $s_0 \xrightarrow{a} s_1 \xrightarrow{b} s_3$ and $s_0 \xrightarrow{b} s_2 \xrightarrow{a} s_3$, both of length 2. Both paths are in $[\mathsf{Paths}]_{\mathsf{ego}}$, as $s_3 \in S_{\mathsf{ego}}$.*

Policies and Schedules

Whenever some state s is reached, the corresponding player draws an action from $A(s)$. As standard, we capture this with the notion of a scheduler[1]. A *scheduler* is a tuple of *player policies* $\sigma = \langle \sigma_{\mathsf{ego}}, \sigma_{\mathsf{env}} \rangle$ with $\sigma_{\mathsf{i}} \colon [\mathsf{Paths}]_{\mathsf{i}} \to \mathit{Distr}(A)$ such that $\mathsf{support}(\sigma_i(\xi)) \subseteq A(\text{last}(\xi))$ for each ξ, i.e., for every history, the policy sets a distribution over the enabled successor actions. For a given path, ξ and a policy σ_{i}, we denote by $\sigma_{\mathsf{i}}(a \mid \xi)$ the distribution of actions induced by σ_{i} given the path ξ. To ease notation, we liberally use the notation $\sigma \colon \mathsf{Paths} \to \mathit{Distr}(A)$, where this function is given dependent on which player owns the last state.

Example 7.2.5. *An example for a ego-policy σ_{ego} is given by,*

$$\sigma_{\mathsf{ego}}(\alpha \mid \xi) = \begin{cases} 1/2 & \text{if } \alpha \in \{a, b\},\ \xi = s_0, \\ 1 & \text{if } \alpha = a,\ \xi = s_0 \xrightarrow{b} s_2 \xrightarrow{a} s_3, \\ 1 & \text{if } \alpha = b,\ \xi = s_0 \xrightarrow{a} s_1 \xrightarrow{b} s_3. \end{cases}$$

The probability $\Pr(\xi \mid \sigma)$ of a finite path ξ in an SG $\mathcal{G}$ conditioned on a policy σ is given by the product of the transition probabilities along a path. More precisely, we define the probability $\Pr(\xi \mid \sigma)$ recursively as:

$$\begin{aligned} \Pr(s \mid \sigma) &\overset{\text{def}}{=} 1 \\ \Pr(\xi \mid \sigma) &\overset{\text{def}}{=} \Pr(\xi' \mid \sigma) \cdot \sigma(a \mid \xi') \cdot P(\text{last}(\xi'), a)(s') \end{aligned} \tag{7.1}$$

where $\xi = \xi' \xrightarrow{a} s'$. Furthermore, just like with MDPs, the probability of a prefix-free set $X \subseteq \mathsf{Paths}$ of paths is the sum over the individual path probabilities, $\Pr(X \mid \sigma) = \sum_{\xi \in X} \Pr(\xi \mid \sigma)$.

[1]Also known as *strategy* or *policy.*

Entropic Reactive Control Improvisation

Finally, to simplify our problem statement, we now define causal entropy of a schedule in a stochastic game. Recall that a path alternates states and actions. The next state after observing a sequence of state-action pairs is a random variable. Formally, given $\mathcal{G}$ and a scheduler σ, let us denote by $\mathcal{A}^{\mathsf{ego}}_{1:i}$ and $\mathcal{S}_{1:i}$ random variable sequences for ego-player actions and states respectively. The causal entropy of controllable actions in τ-length paths under σ is then,

$$H_\tau(\sigma) \overset{\text{def}}{=} H(\mathcal{A}^{\mathsf{ego}}_{1:\tau'} \,¦\, \mathcal{S}_{1:\tau}), \tag{7.2}$$

where $\tau' = \lceil \frac{\tau}{2} \rceil$ is the number of ego-actions due to alternation.

Example 7.2.6. *Consider the uniform* ego *policy on Fig. 7.2. If* $\sigma_{\mathsf{env}}(a \mid \xi) = 1$. $H_\tau(\sigma) = \log(2) + {}^1\!/_2(\log(2))$. *Note, only* ego *can* add *entropy, while* env *and stochastic transitions yield convex combinations via expectation.*

We now formalize the problem statement.

The Entropic Control Improvisation (ERCI) Problem: Given a SG $\mathcal{G}$, τ-bounded path properties ψ and φ, and thresholds $\mathbf{p} \in [0, 1]$ and $\mathbf{h} \in [0, \infty)$, find a ego-policy σ_{ego} (or report that none exists) such that for every env-policy σ_{env},

1. (*hard constraint*) $\Pr(\psi \mid \sigma) \geq 1$
2. (*soft constraint)* $\Pr(\varphi \mid \sigma) \geq \mathbf{p}$
3. (*randomness constraint*) $H_\tau(\sigma) \geq \mathbf{h}$

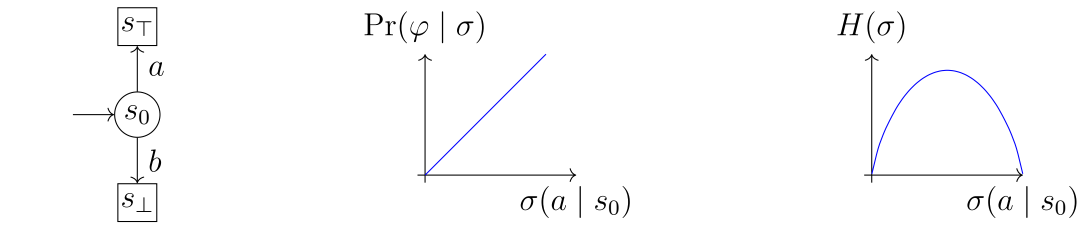
where $\sigma = \langle \sigma_{\mathsf{ego}}, \sigma_{\mathsf{env}} \rangle$.

We say that an instance of the ERCI problem is realizable, if an appropriate σ_{ego} exists and call such σ_{ego} an *improviser*. The problem is unrealizable otherwise.

Remark 7.2.7. The maximum causal entropy planners used in Part I correspond to the optimization variant of ERCI. As well shall see, our solution to ERCI actually creates improvisers that maximized causal entropy subject to a performance constraint (as desired).

7.3 ERCI as multi-objective optimization

We investigate the ERCI problem statement. Based on a sequence of observations, we reduce the ERCI problem to the Core ERCI problem which significantly eases the description (and implementation) of the algorithm afterwards.



(a) Minimal MDP **(b)** Probability to reach $s_\top$ **(c)** Causal Entropy

Figure 7.3: Minimal ERCI problem with $\varphi = (\mathrm{last}(\xi) = s_\top)$

Preprocessing

To ease the technical exposition, we make the following assumptions (without loss of generality): We assume the graph structure underlying the SG is finite and acyclic – and thus all paths are finite length. When considering τ-bounded path properties (monitorable by finite automata), this assumption is naturally realized by a τ-step unrolling of a monitor augmented SG, i.e., augmenting the state space with a counter from 0 to τ and the current property monitor state.

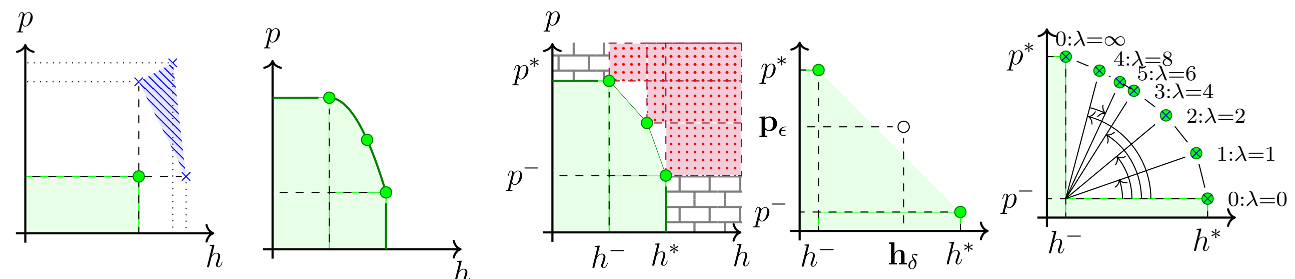
Next, in order to ensure the hard constraint, ψ, we calculate all states from which the env-player can enforce violating the hard constraint. Such states are identifiable using a single topologically ordered pass over $\mathcal{G}$ from the terminal states to the initial state. We remove such states along with their in- and outgoing transitions. Any ego-policy now satisfies the hard constraint. The remaining terminal states are all merged into two states $s_\top$ and $s_\bot$, based on membership in φ, i.e.,

$$\begin{array}{lcl} \mathrm{last}(\xi) = s_\top & \implies & \xi \in \varphi \\ \mathrm{last}(\xi) = s_\bot & \implies & \xi \notin \varphi \end{array}. \tag{7.3}$$

Example 7.3.1. *In Fig. 7.3a we show a (deterministic) MDP and we plot for all schedulers the induced probability to reach $s_\top$ and the induced causal entropy, in Fig. 7.3b and 7.3c, respectively. We see that taking action a with increasing probability yields a larger probability to reach $s_\top$, whereas taking action a and b uniformly at random is optimal for the entropy.*

Geometric Perspective

There is a natural trade-off between probability of generating paths in φ (from here onwards: *the performance*) and causal entropy induced by a policy (*the randomization*). In particular, with all other ingredients fixed, we are interested in understanding the combinations of $\mathbf{p}$ and $\mathbf{h}$ that yield a solvable instance of the (core) ERCI problem. To this end, we cast ERCI as an instance of a multi-objective optimization problem, and study its Pareto front. Some ideas are inspired by variants of multi-objective analysis of MDPs with multiple soft constraints, e.g. [32, 50, 53].



(a) Guaranteed points $\mathbb{S}_{\sigma_{\mathsf{ego}}}$ **(b)** Solutions $\mathbb{S}$ **(c)** Iterative construction **(d)** Regret-based ERCI **(e)** Rationality-based algorithm

Figure 7.4: Geometric interpretation of the ERCI problem for some fixed SG.

It is convenient to consider this front geometrically. To begin, given a fixed ERCI instance, a scheduler σ *induces* a point x_σ:

$$x_\sigma \overset{\text{def}}{=} \Big\langle \Pr(X_\varphi \mid \sigma), H(\sigma) \Big\rangle \in [0,1] \times [0,\infty). \tag{7.4}$$

To ease notation, for $x_\sigma = \langle p, h \rangle$ we use $p_\sigma \overset{\text{def}}{=} p$ and $h_\sigma \overset{\text{def}}{=} h$. Next, we partially order these points via the standard product ordering:

$$\langle p, h \rangle \preceq \langle p', h' \rangle \quad \text{iff} \quad p \le p' \wedge h \le h'. \tag{7.5}$$

We say that σ_{ego} *guarantees* a point $x_{\mathsf{ego}} \overset{\text{def}}{=} \langle p, h \rangle$, if for every policy σ_{env}, using $\sigma = \langle \sigma_{\mathsf{ego}}, \sigma_{\mathsf{env}} \rangle$, we have $p_\sigma \ge p$ and $h_\sigma \ge h$. Thus, a point is guaranteed if no matter what policy env uses, x_σ will induce a point no worse w.r.t. to either randomization or performance than x_{ego}. We define *the set of guaranteed points* for a scheduler σ_{ego}:

$$\mathbb{S}[\sigma_{\mathsf{ego}}] \overset{\text{def}}{=} \{\langle p, h \rangle \mid \sigma_{\mathsf{ego}} \text{ guarantees } \langle p, h \rangle\}. \tag{7.6}$$

We observe that guaranteed points are downward closed, i.e., if σ_{ego} guarantees x and $x' \preceq x$, then σ_{ego} guarantees x'.

Example 7.3.2. *Consider Fig. 7.4a. We fix σ_{ego} and in the blue hatched area draw all points induced by $\sigma = \langle \sigma_{\mathsf{ego}}, \sigma_{\mathsf{env}} \rangle$ when varying σ_{env}. We take the minimal randomness h and the minimal performance p. The points in the downward closure of $\langle p, h \rangle$ (green circle) are the guaranteed points for σ_{ego} in the green solid area. We notice the gap between both areas: While the performance and randomization may be better than the optimum that* ego *can guarantee, it cannot guarantee a higher randomization* and *performance simultaneously, as the* env*-player would have a counter-policy violating either the performance* or *the randomization.*

Points guaranteed by some σ_{ego} are called *achievable*. Thus, the achievable points are: $\mathbb{S} = \bigcup_{\sigma_{\mathsf{ego}}} \mathbb{S}[\sigma_{\mathsf{ego}}]$. Importantly, the ERCI problem is realizable iff $\langle \mathbf{p}, \mathbf{h} \rangle$ is achievable. Thus, to solve ERCI instances, we start by characterizing $\mathbb{S}$. We start by observing the $\mathbb{S}$ is convex[2] (proof in Sec 7.7).

Proposition 7.3.3. The set of achievable points, $\mathbb{S}$, is convex.

Next, because $\mathbb{S}$ is downward closed, it suffices to study the "maximal" or non-dominated points. Precisely, we say that a point x is *dominated* by x' if $x \prec x'$, i.e., if $x \preceq x' \wedge x \neq x'$. The Pareto front $\mathcal{F}_{\mathbb{S}}$ of $\mathbb{S}$ is then the set of non-dominated achievable points,

$$\mathcal{F}_{\mathbb{S}} \overset{\text{def}}{=} \{x \in \mathbb{S} \mid \forall x' \in \mathbb{S}, x \not\prec x'\}. \tag{7.7}$$

Importantly, it holds that the ERCI problem is satisfiable iff there exists a $x \in \mathcal{F}_{\mathbb{S}}$ such that $\langle \mathbf{p}, \mathbf{h} \rangle \preceq x$.

Example 7.3.4. *The set $\mathbb{S}$ illustrated in Fig. 7.4b is obtained by taking the union of guaranteed points, and can be characterized by the set of points on the Pareto front: This is the curved border between the green and white area, in particular the three green dots are on the Pareto front. Any ERCI instance with $\langle \mathbf{p}, \mathbf{h} \rangle$ in the green area is realizable.*

Approximating the Pareto front gives a natural approximation scheme for ERCI instances: For any subset $\mathcal{F} \subseteq \mathcal{F}_{\mathbb{S}}$,

1. If there exists an $x \in \mathcal{F}$ such that $\langle \mathbf{p}, \mathbf{h} \rangle \preceq x$, then the ERCI problem must be realizable and x is a witness to realizability.
2. If there exists an $x \in \mathcal{F}$ such that $x \prec \langle \mathbf{p}, \mathbf{h} \rangle$, then the ERCI problem is not realizable and x is a witness to unrealizability.

Due to convexity, we may speed up the search for realizability: If there exist $x_1, x_2 \in \mathcal{F}$ such that $\langle \mathbf{p}, \mathbf{h} \rangle \prec \Big(w \cdot x_1 + (1-w) \cdot x_2\Big)$, we call x_1, x_2 a witness-pair.

Remark 7.3.5. Given a witness (pair) to realizability, it is easy to extract the corresponding improviser. Let x_1, x_2 be a witness-pair to realizability, induced by σ_{λ_1} and σ_{λ_2} such that $\langle \mathbf{p}, \mathbf{h} \rangle \preceq w \cdot x_1 + (1-w) \cdot x_2$, then the policy described by

$$\sigma^*_{\mathsf{ego}}(a \mid s) \overset{\text{def}}{=} w \cdot \sigma_{\lambda_1}(a \mid s) + (1-w) \cdot \sigma_{\lambda_2}(a \mid s) \tag{7.8}$$

is an improviser solving the ERCI problem.

[2]That is, $x, x' \in \mathbb{S}$ implies for every $w \in [0,1]$ that $w \cdot x + (1-w) \cdot x \in \mathbb{S}$

Example 7.3.6. *Consider Fig. 7.4c. We have found three points on the Pareto front, and already have a good impression of the trade-off between randomization and performance. In particular, the green area is definitively a subset of $\mathbb{S}$: It exploits the downward closure and the convexity of $\mathbb{S}$. The red (dotted) parts contain the points on the Pareto front in their downward closure, thus they cannot be part of the Pareto front themselves. Furthermore, the topmost point on the Pareto front was obtained by maximizing performance (and optimizing randomization only as a secondary objective). Thus, by construction, the bricked area at the top is not realizable. Analogously, the bricked area at the right reflects non-achievable randomization.*

Remark 7.3.7. We notice that the multi-objective optimization perspective allows us to extend the set of witnesses for unrealizability. In particular, every point of the Pareto-curve can be described as optimizing some scalarization of the objectives. Geometrically, it optimizes along a particular direction. Whenever we know that a Pareto-optimal point $x = \langle p, h \rangle$ optimizes a weighted objective with weights $w = \langle w_1, w_2 \rangle$, then x and w *together* are a witness for unrealizability for $\langle \mathbf{p}, \mathbf{h} \rangle$ whenever $w_1 \cdot p + w_2 \cdot h < w_1 \cdot \mathbf{p} + w_2 \cdot \mathbf{h}$.

Thus a key algorithmic question in ERCI is how to efficiently explore the Pareto front $\mathcal{F}_{\mathbb{S}}$.

Regret-Based ERCI

To algorithmically explore the Pareto-curve, we re-parameterize the ERCI problem. First, we find the two special points induced by (1) optimizing performance and only then randomization (the topmost green point in the figures) and (2) optimizing randomization and only then performance (the rightmost green point). As we have seen, these restrict the domain in which we can actually trade performance for randomness. We define $h^* \overset{\text{def}}{=} \max\{h \mid \exists p \text{ s.t. } \langle p, h \rangle \in \mathbb{S}\}$, i.e., the largest randomness that can be guaranteed by any ego-policy. Likewise, we define $p^* \overset{\text{def}}{=} \max\{p \mid \exists h \text{ s.t. } \langle p, h \rangle \in \mathbb{S}\}$, i.e., the largest performance that can be guaranteed by any ego-policy. Then, we define $p^- \overset{\text{def}}{=} \max\{p \mid \langle p, h^* \rangle \in \mathbb{S}\}$, the best performance that ego can guarantee while guaranteeing optimal randomness. Likewise, we define the analogous $h^- \overset{\text{def}}{=} \max\{h \mid \langle p^*, h \rangle \in \mathbb{S}\}$. We thus obtain two points on the Pareto front: $\langle p^-, h^* \rangle$ and $\langle p^*, h^- \rangle$, and intuitively, we can trade between these two points following the Pareto front.

Now, rather that fixing $\mathbf{p}$ and $\mathbf{h}$ a priori, we seek to guarantee some percentage of the independently achievable soft constraint and causal entropy measure. We re-parameterize ERCI as follows:

$$\mathbf{p}_\epsilon \overset{\text{def}}{=} \epsilon \cdot (p^* - p^-) + p^- \quad \mathbf{h}_\delta \overset{\text{def}}{=} \delta \cdot (h^* - h^-) + h^- \tag{7.9}$$

where $\epsilon, \delta \in [0, 1]$. We call this version of ERCI *regret-based.* We remark that the reparameterization is not only beneficial from a usability point-of-view, but it also eases our exposition. Geometrically, after computing p^* and h^*, we know that the left triangle in Fig. 7.4d is definitively realizable, and the regret-based ERCI asks whether the white circle is also realizable (where the point of the white point is given by ϵ and δ). Together, we obtain the following (core) ERCI problem.

The Core ERCI Problem: Given an finite acyclic SG $\mathcal{G}$, with terminal states, $s_\top$ and $s_\bot$, and thresholds $\epsilon, \delta \in [0, 1]$, find a ego-policy σ_{ego} s.t. for every env-policy σ_{env}:

1. (*soft constraint*) $\Pr(\text{last}(\xi) = s_\top \mid \sigma) \geq \mathbf{p}_\epsilon$
2. (*randomness constraint*) $H(\sigma) \geq \mathbf{h}_\delta$

where $\sigma = \langle \sigma_{\mathsf{ego}}, \sigma_{\mathsf{env}} \rangle$.

Finally, it is helpful to think about the Pareto front as a function of randomization in this reparameterization. We define a characteristic function which given a target performance ratio, ϵ, yields the optimal randomness ratio, δ:

$$\begin{aligned} &f_{\mathbb{S}} \colon [0, 1] \to [0, 1] \\ &f_{\mathbb{S}}(\delta) = \max_{\epsilon} \{\mathbf{h}_\delta \mid \langle \mathbf{p}_\epsilon, \mathbf{h}_\delta \rangle \in \mathbb{S}\} \end{aligned} \tag{7.10}$$

Proposition 7.3.8. $f_{\mathbb{S}}$ is continuous and (strictly) decreasing.

We shall temporarily postpone the proof of Prop. 5.2.2. For now, one can observe that (non-strict) monotone decreasing follows directly from convexity and using the adequate domains. Finally, the set $\mathbb{S}$ is (in general) *not* a finite polytope – the MDP in Fig. 7.3a serves as an example. Nevertheless, $\mathbb{S}$ can be well approximated with finitely many vertices, see Ex. 7.3.6. With these facts, we are now well-equipped to develop the algorithms in Sec. 7.4 for MDPs and Sec. 7.5 for SGs.

7.4 The Control Improvisation Problem for MDPs

We present an algorithm for the control improvisation problem for MDPs, which in the next section, will serve as a subroutine for an algorithm on SGs. Our goal shall be to instantiate the approximation scheme from the previous section. In particular, we seek to find points on the Pareto curve $\mathcal{F}_{\mathbb{S}}$ and incrementally build up $\mathcal{F} \subseteq \mathcal{F}_{\mathbb{S}}$.

Rationality

To start, recall that an MDP is a stochastic game with no action choices for the environment, i.e., the environment is purely stochastic and the only degree of freedom is ego's policy. The key idea for finding points on the Pareto-curve is to rephrase the trade-off between randomization and performance as a degree in rationality λ of the policy. Formally, the rationality corresponds to the following scalarization of our multi-objective problem [96],

$$J_\lambda(\sigma) \overset{\text{def}}{=} \Big\langle 1, \lambda \Big\rangle \cdot \Big\langle h_\sigma, p_\sigma \Big\rangle. \tag{7.11}$$

As we discussed in Ch 3, for MDPs, the **unique** (ego-)policy that optimizes (7.11) is given by a smooth variant of the Bellman equations [162], i.e., for each rationality $\lambda \in [0, \infty)$, we define a policy σ_λ – using $s = \text{last}(\xi)$ – as follows:

$$\sigma_\lambda(a \mid s) \overset{\text{def}}{=} \exp(Q_\lambda(s,a) - V_\lambda(s)) \tag{7.12}$$

$$V_\lambda(s) \overset{\text{def}}{=} \begin{cases} \lambda \cdot [s = s_\top] & \text{if } s \in \{s_\top, s_\bot\}, \\ \text{smax}_{a \in A(s)} Q_\lambda(s,a) & \text{otherwise.} \end{cases} \tag{7.13}$$

$$Q_\lambda(s,a) \overset{\text{def}}{=} \sum_{s'} P(s,a,s') \cdot V_\lambda(s'). \tag{7.14}$$

Remark 7.4.1. As opposed to (3.25), here we have written the smooth Bellman backup (3.25) in its more traditional Q and V decomposition. This is to emphasize that the value is a function of the state (or state distribution) indexed by the prefix, ξ.

To ease notation, we denote $x_\lambda \overset{\text{def}}{=} x_{\sigma_\lambda}, p_\lambda \overset{\text{def}}{=} p_{\sigma_\lambda}, h_\lambda \overset{\text{def}}{=} h_{\sigma_\lambda}$. Intuitively, as $\lambda \to 0$, σ_λ approaches the uniform distribution over *all available actions*. Note that this policy maximizes (causal) entropy, and thus $h^* = h_0$. As $\lambda \to \infty$, this variant of the Bellman equations coincides with the standard Bellman equations [120], where σ_λ selects (uniformly) from actions *that maximize performance*. Furthermore, the monotonicity and smoothness of the above Bellman equations yields the following proposition.

Proposition 7.4.2. p_λ is continuously (and strictly) increasing in λ and h_λ is smoothly (and strictly) decreasing in λ.

In terms of $f_\mathbb{S}$, we can define:

$$\epsilon_\lambda \overset{\text{def}}{=} \frac{p_\lambda - p_0}{p_\infty} + p_0 \quad \text{and} \quad \delta_\lambda \overset{\text{def}}{=} \frac{h_\lambda - h_\infty}{h_0} + h_\infty. \tag{7.15}$$

Then, because σ_λ maximizes randomness given a target performance, one derives:

$$f_\mathbb{S}(\delta_\lambda) = \epsilon_\lambda. \tag{7.16}$$

What remains is to instantiate the approximation scheme for the Pareto front by varying the optimization direction $\langle \lambda, 1 \rangle$.[3] In particular, we construct $\mathcal{F} = \{x_\lambda \mid \lambda \in \{\lambda_1, \lambda_2, \ldots\}\}$ until $\mathcal{F}$ contains a witness to either realizability or unrealizability of the ERCI instance. We notice that the scalarization in (7.11) means that we may additionally exploit witnesses to unrealizability as outlined in Remark 7.3.7. In the remainder of this section, we improve upon randomly selecting values for λ.

[3]Assuming $p^*, h^* \neq 0$ (which would otherwise yield trivial $\mathbb{S}$ and $\mathcal{F}_\mathbb{S}$)

Targeted Pareto-exploration

The key ingredient to improve upon arbitrarily selecting $\lambda_1, \ldots \lambda_i$ is to exploit additional structure of the rationality. We propose a three staged sequence: (i) Compute x_λ for the end points $\lambda \in \{0, \infty\}$. (ii) Double λ (starting at $\lambda = 1$) until $h_\lambda \leq \mathbf{h}$, yielding $\lambda_1 \ldots \lambda_j$. (iii) Binary search for $\lambda \in [\lambda_{j-1}, \lambda_j]$. We illustrate the idea in Fig. 7.4e.

The algorithm terminates almost surely, that is: the algorithm halts if $\langle \mathbf{p}, \mathbf{h} \rangle$ is not on $\mathcal{F}_{\mathbb{S}}$ (or if we happen to exactly hit $\langle \mathbf{p}, \mathbf{h} \rangle$ by selecting some rationality λ). As the Pareto front has measure 0, we argue that not halting is thus merely a technical concern, as a small perturbation to the ERCI instance (i.e. a *smoothed analysis* [138]) on $\mathcal{G}$ admits decidability.

Our approximation scheme yields a semi-decision process which halts iff either (a) $\langle \mathbf{p}, \mathbf{h} \rangle$ is bounded away from $\mathcal{F}_{\mathbb{S}}$ *or* (b) $\langle \mathbf{p}, \mathbf{h} \rangle$ is dominated by x_{λ_i}.

Next, observe that if we terminate the binary search when the search region is smaller than Δ, this approximation scheme becomes linear in the MDP size and logarithmic in the final rationality, λ_*, and the resolution, Δ, i.e., the run-time is,

$$\mathcal{O}\Big(\underbrace{|\mathcal{G}|}_{\text{Evaluate } x_\lambda} \cdot \overbrace{\log(\lambda_*)}^{\text{Doubling Phase}} \cdot \underbrace{\log(1/\Delta)}_{\text{Binary Search}} \Big) \tag{7.17}$$

Finally, before generalizing to stochastic games, we observe that in practice, $\lambda = 100$ yields a nearly optimal policy, and thus one can often assume $\lambda_* \leq 100$ in our run-time analysis.

7.5 The Control Improvisation Problem for SGs

MDP algorithm in hand, we are now ready to provide an algorithm for stochastic games.

Environment Policies

We begin with three observations about the env-policies. First, for ERCI, we can assume an adversary for env that aims to foil ego achieving both the performance *and* randomization requirement. We call such a env-policy *violating*. For a policy to be violating, it suffices to violate, against every ego-policy independently, either performance *or* randomization. Second, if there is a violating env-policy, there is a deterministic env-policy that proves this. In particular, at every state, σ_{env} may choose to violate either constraint via the appropriate action with no incentive to randomize. Third, fixing an environment policy reduces $\mathcal{G}$ to a MDP $\mathcal{G}[\sigma_{\mathsf{env}}]$.

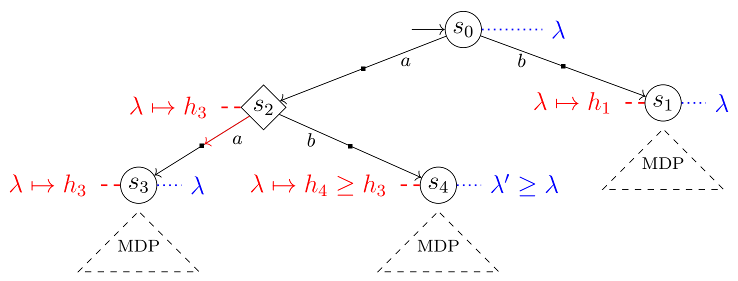


Figure 7.5: SG to illustrate entropy matching policies.

A Sufficient Class of Policies

For MDPs, we have seen that varying rationality is sufficient to explore the Pareto curve. We show that we can adapt that idea to a class we call *entropy matching policies*, which may be indexed by the (initial) rationality. In the initial state, we start by assuming that env selects a (deterministic) policy, $\sigma^{\lambda}_{\mathsf{env}}$, that lexicographically minimizes the guaranteed randomness, followed by performance. On the sub-graph, $\mathcal{G}[\sigma^{\lambda}_{\mathsf{env}}]$, ego employs the corresponding entropy maximizing policy for the MDP $\mathcal{G}[\sigma^{\lambda}_{\mathsf{env}}]$. Whenever env diverges from the entropy minimizing policy (to decrease the induced performance), we let ego increase its rationality such that it still induces the same guaranteed randomness. We refer to this idea as *entropy matching*. The idea is that the rationality at the initial state induces a worst-case entropy, and whatever env chooses to do, throughout the SG, we ensure that we indeed obtain this entropy. The policy thus tracks this entropy and if necessary adapts the rationality (which we call *replanning*). Replanning ensures we obtain the optimal performance from a particular point while still ensuring the required randomness.

Example 7.5.1. *We sketch an entropy matching policy in Fig. 7.5. In particular, we show part of a SG. For some fixed rationality λ, we annotate in red, on the left of the SG states, the entropy obtained when assuming that* env *plays an entropy-minimizing policy as outlined above. In particular, this means that in s_2,* env *selects action a. Now, our entropy-matching policy (in blue, on the right) will play with rationality λ, unless state s_4 is reached. As this ensures a higher entropy, we may now select a higher rationality, λ'.*

Soundness and Completeness

Importantly, observe that because fixing a policy for ego yields a verifiable point in $\mathbb{S}$, any witness for realizability we find is trivially sound. For completeness, we can restrict ourselves to the case in which our algorithm claims the ERCI instance unrealizable. Surprisingly, the class of policies we consider suffices, and the algorithm is thus sound and (whenever halting) complete (proof provided in Sec 7.7). That is, all guaranteed points are witnessed by an entropy matching policy!

Further, observe that as a corollary of the entropy matching family being complete, it must be the case that $f_{\mathbb{S}}(h_\lambda)$ inherits continuity and (strict) monotonicity from the MDP case. Namely, at each env state, the achievable points $\mathbb{S}$ are necessarily the intersection of the achievable points of the sub-graphs. By induction, (with the MDP base case), we obtain continuity and strict monotonicity.

Algorithm: Memoizing Pareto Fronts

We propose approximating the Pareto front using the same three staged sequence of exploring rationality coefficients (at the initial state) as the MDP case: (1) endpoints, (2) doubling, (3) binary search. To perform the these computations efficiently, we adopt a geometric perspective. Observe that each node of $\mathcal{G}$ indexes a sub-graph, which has a corresponding Pareto front for trading performance for randomness. Further, note that the Pareto front at an env node is the intersection of the Pareto fronts of its child nodes. Entropy matching corresponds to "switching" between Pareto fronts and adjusting the optimization direction by increasing the rationality. Thus, by traversing the graph from the terminal states to the initial state, approximating Pareto fronts along the way, one can memoize how to trade performance for randomness at any given node. This preprocessing enables determining the minimum entropy response for any optimization direction and quickly replanning via a convex combination of Pareto optimal policies.

Approximate Pareto Fronts

Of course, by varying λ, one can only construct approximate Pareto fronts $\hat{\mathcal{F}} \subseteq \mathcal{F}_{\mathbb{S}}$. We propose the following high-level algorithm to adapt the above algorithm to the case where each Pareto front approximation introduces at most κ error along the performance axis.

1. Let τ denote the length of the longest path in $\mathcal{G}$.
2. Let $0 < \kappa < 1$ be some arbitrary initial tolerance.
3. Recursively compute κ-close Pareto fronts for each successor state using replanning.
4. If the any minimum entropy action cannot be determined or $\mathbf{p}$ is within $\kappa \cdot \tau$ distance to (but outside of) $\hat{\mathcal{F}}$, halve κ and repeat.
5. Otherwise, perform the entropy matching algorithm (with initial entropy $\mathbf{h}$) using these Pareto fronts and return the resulting policy (if on exists).

The soundness of this algorithm relies on the following critical facts: (1) Given sufficient resolution, the minimum entropy env-actions can be determined. (2) The resulting entropy depends solely on the resulting sub-graph (and is independent of the current Pareto approximation). (3) Thus, when querying points on $\mathcal{F}_{\mathbb{S}}$, error can only accumulate for p. (4) Next, observe that p is computed using convex combinations of entropy matched points on Pareto approximations. (5) Convex combinations of an error interval cannot increase the error, i.e.,

$$q \cdot [x, x+\kappa] + \bar{q} \cdot [y, y+\kappa] = [z, z+\kappa], \tag{7.18}$$

where $z = q \cdot x + \bar{q} \cdot y$. Thus, so long as $\kappa \cdot \tau$ is enough resolution to answer $p_\lambda < \mathbf{p}$, one obtains a semi-decision procedure as in the MDP case.

Termination and Run Time

First, as in the MDP case, the algorithm terminates almost surely, with the exception occurring only for a subset of the Pareto front. Below, we give an output-sensitive analysis of the run time (assuming it does halt). If κ^* tolerance is required to terminate, then the κ search introduces $\mathcal{O}(\log(1/\kappa^*))$ iterations. Next, observe that each node need process a given rationality coefficient at most once. Further, looking up which pair of rationalities are need to upper and lower bound the performance for a given randomness can be done in logarithmic time via binary search on rationality coefficients. As the corresponding bounds and convex combinations can be computed in constant time, this means this algorithm runs in time:

$$\mathcal{O}\Big(\log(1/\kappa^*) \cdot N_\lambda \cdot \log(N_\lambda) \cdot |\mathcal{G}|\Big), \tag{7.19}$$

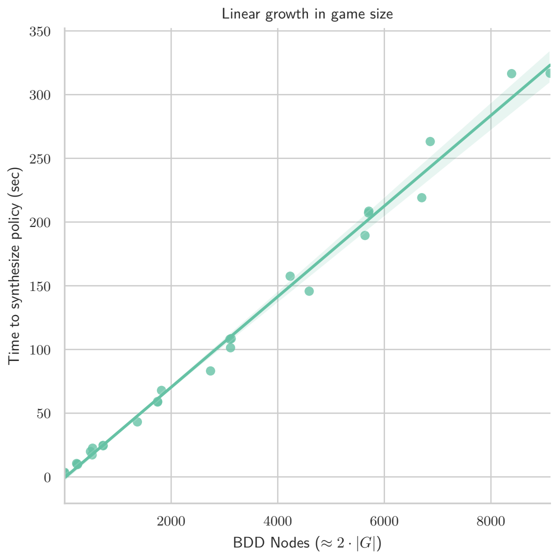
where, N_λ is the number of unique rationality coefficients processed. If, as in the MDP case, one assumes a maximum rationality coefficient λ^* and a minimum rationality resolution Δ, one obtains:

$$\mathcal{O}\Big(\underbrace{\log(1/\kappa^*)}_{\kappa\text{ search}} \cdot \underbrace{\lambda^*/\Delta \cdot \overbrace{\log(\lambda^*/\Delta)}^{\text{Replanning}} \cdot |\mathcal{G}|}_{\text{Evaluate }\lambda}\Big). \tag{7.20}$$

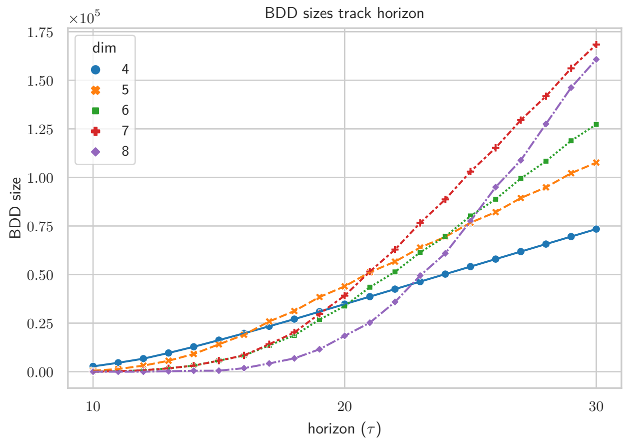
The above however is very conservative and empirically we observe N_λ bounded far away from λ^*/Δ.

7.6 Implementation and Empirical Evaluation

To experimentally validate the feasibility of our ERCI algorithm for SGs, we implemented [147] our algorithm in Python. The domain considered was the same as our motivating example. Specifically, our experiments used a $k \times k$ grid discretization of the workspace (cf. Fig. 7.1), for $k \in \{4, 5, 6, 7\}$ where the four target houses lie in $\{\lfloor k/3 \rfloor, \lfloor 2k/3 \rfloor\}^2$, and the drones D_{test} and D_{new} are initially at in the bottom left corner and top right house resp. Furthermore, for simplicity, we embedded the avoid crash condition as part of the soft constraint, rather than a hard constraint[4]. We took ego's dynamics to be deterministic and modeled env as visiting each house in either clock-wise or counter-wise order, where the orientation can switch with (a potentially state dependent) probability $p \in [1/100, 1/50]$ whenever a house is visited. Next, we considered an alternation between ego and env to be a single logical time step, and



(a) Experimental times for computing Pareto front of a variety delivery drone problems.



(b) BDD graph size as a function of horizon for the problems in our benchmark suite. Distribution of problems, non-uniform in horizon to avoid small horizon artifacts.

Figure 7.6: Plots to illustrate scalability

(non-uniformly) instantiate problem instances with horizons ranging from 6 to 18, i.e., paths ranged from length 12 to 36. Finally, as we shall expand on in the next two chapters, each SG was represented in a compressed form as a Binary Decision Diagram (BDD) [23].

Results

First and foremost, we succeed in synthesizing controllers in the mentioned setup. The controller randomizes its behavior while meeting the specification, which is not surprising as the algorithm yields a correct-by-construction policy.

Next, we consider the practical run time of our algorithm. As Fig 7.6a demonstrates, the empirical time to estimate the Pareto front seemed to increase linearly with our SG encoding – which is consistent with our complexity analysis. Moreover, our encoding seems to linearly track with the horizon for all k (Fig. 7.6b), suggesting that the overall run time grows linearly in the horizon within our parameterization. When combined with the potential to parallelize across the rationality coefficients, these results suggest that practical optimizations to our ERCI algorithm may admit usage on other more complicated benchmarks. Finally, we remark that the use of a decision diagram encoding did indeed dramatically decrease the size of the SG (with negligible overhead).[5]

[4]Note that counter-intuitively, only using soft constraints generally results in harder instances as the compressed SGs are larger.

[5]For example, the $(k = 8, \text{horizon} = 18)$ case is encoded using a 505,100 node BDD ($|\mathcal{G}| = 6,861$ nodes).

7.7 Proofs

Convexity of ERCI solution set

Proof Sketch Prop 7.3.3. Recall that a set is convex, if it is closed under convex-combinations[6]. Consider two points $\langle p, h\rangle, \langle p', h'\rangle \in \mathbb{S}$ achieved by σ_{ego} and σ'_{ego} respectively. Consider the new policy, π, defined by employing σ_{ego} with probability q and σ'_{ego} with probability $\bar{q} \overset{\text{def}}{=} 1-q$. Because each policy *guarantees* its corresponding performance, this new policy has performance at least $q \cdot p + \bar{q} \cdot p'$. Similarly, by viewing π as a random variable and applying chain rule yields,

$$\begin{aligned} H_\tau(\sigma) &\geq q \cdot H(\mathcal{A}^{\mathsf{ego}}_{1:\tau'} \mathrel{\vdots} \mathcal{S}_{1:\tau} \mid \pi = \sigma_{\mathsf{ego}}) + \\ &\quad \bar{q} \cdot H(\mathcal{A}^{\mathsf{ego}}_{1:\tau'} \mathrel{\vdots} \mathcal{S}_{1:\tau} \mid \pi = \sigma'_{\mathsf{ego}}) \\ &= q \cdot h + \bar{q} \cdot h'. \end{aligned} \tag{7.21}$$

Thus, any convex combination of guaranteed points is guaranteed by a convex combination of the corresponding ego policies. □

Completeness of Entropy Matching for SGs

Proof Sketch of SG Completeness. We prove the statement by induction over the (acyclic) SG. First, observe that on games with only terminal nodes, completeness follows directly. Next, suppose the entropy matching family is complete on all sub-graphs of $\mathcal{G}$. To simplify our proof, observe that w.l.o.g., we can restrict our attention to ERCI instances on the Pareto front, $\langle \mathbf{p}, \mathbf{h}\rangle \in \mathcal{F}_{\mathbb{S}}$. Next, for the sake of contradiction, we shall assume that no entropy matching policy achieves $\langle \mathbf{p}, \mathbf{h}\rangle$, but σ^*_{ego} does:

$$\forall \sigma_{\mathsf{ego}} \in \{\sigma^\lambda_{\mathsf{ego}}\}_\lambda \;.\; x_{\sigma_{\mathsf{ego}}} \prec \langle \mathbf{p}, \mathbf{h}\rangle \tag{7.22}$$

$$\exists \sigma^*_{\mathsf{ego}} \notin \{\sigma^\lambda_{\mathsf{ego}}\}_\lambda \;.\; \langle \mathbf{p}, \mathbf{h}\rangle \preceq x_{\sigma^*_{\mathsf{ego}}}. \tag{7.23}$$

Indeed, we may reformulate (7.23) to

$$\exists \sigma^*_{\mathsf{ego}} \notin \{\sigma^\lambda_{\mathsf{ego}}\}_\lambda \;.\; \langle \mathbf{p}, \mathbf{h}\rangle = x_{\sigma^*_{\mathsf{ego}}} \tag{7.24}$$

as we assumed that $\langle \mathbf{p}, \mathbf{h}\rangle$ is Pareto-optimal.

Note that because the entropy matching family contains the maximizers and minimizers of entropy ($\lambda = \infty$ and $\lambda = 0$ resp.), and because increasing rationality monotonically decreases entropy, there must exist some rationality, λ, such that $\sigma^\lambda_{\mathsf{ego}}$ induces entropy $\mathbf{h}$:

$$h_{\sigma^\lambda_{\mathsf{ego}}} = \mathbf{h} = h_{\sigma^*_{\mathsf{ego}}}, \tag{7.25}$$

Compare with the direct encoding $|\mathcal{G}| = |S| \cdot \tau \cdot |\text{monitor state}| = (8 \times 8)^2 \cdot (2 \cdot 18) \cdot (2^4 \cdot 2) \approx 37,000$.

where the second equality follows from (7.24). Next, let $\sigma^\lambda_{\mathsf{env}}$ denote the min-entropy env-policy given $\sigma^\lambda_{\mathsf{ego}}$, i.e., the policy that minimizes entropy in $\mathcal{G}[\sigma^\lambda_{\mathsf{ego}}]$. Because σ^*_{ego} witnesses $\langle \mathbf{p}, \mathbf{h}\rangle$, it must be the case that:

$$h_{\langle \sigma^*_{\mathsf{ego}}, \sigma^\lambda_{\mathsf{env}}\rangle} \geq \mathbf{h} \qquad \text{and} \qquad p_{\langle \sigma^*_{\mathsf{ego}}, \sigma^\lambda_{\mathsf{env}}\rangle} \geq \mathbf{p} \tag{7.26}$$

Recalling that for MDPs, the maximum entropy policies as defined in (7.12)–(7.14) are the unique maximizers of entropy (given p), it must be the case that:

$$\mathbf{h} = h_{\langle \sigma^\lambda_{\mathsf{ego}} \sigma^\lambda_{\mathsf{env}}\rangle} \geq h_{\langle \sigma^*_{\mathsf{ego}}, \sigma^\lambda_{\mathsf{env}}\rangle} \geq \mathbf{h}, \tag{7.27}$$

and thus,

$$h_{\langle \sigma^\lambda_{\mathsf{ego}} \sigma^\lambda_{\mathsf{env}}\rangle} = h_{\langle \sigma^*_{\mathsf{ego}}, \sigma^\lambda_{\mathsf{env}}\rangle}. \tag{7.28}$$

Thus, from uniqueness on MDPs, $\sigma^\lambda_{\mathsf{ego}}$ and σ^*_{ego} must exactly match on $\mathcal{G}[\sigma^\lambda_{\mathsf{env}}]$ and must differ on some other subgraph. Applying the inductive hypothesis, we know that the entropy matching family is complete on these subgraphs, and thus if σ^*_{ego} achieves a given $\langle \mathbf{p}, \mathbf{h}\rangle$ on this subgraph, there must be an entropy matching that does so as well. Thus,

$$x_{\sigma^*_{\mathsf{ego}}} \preceq x_{\sigma_{\lambda^*}}, \tag{7.29}$$

contradicting assumptions (7.22) and (7.23). Thus, entropy matching must be complete. □

7.8 Bibliographic Notes

Control Improvisation in the Literature

In this section, we briefly compare ERCI with other forms of control improvisation. Firstly, we observe that general Control Improvisation has been proposed in stochastic environments for lane changing [56] and imitating power usage in households [5]. However, in those both settings, the randomness constraint is phrased as an upper-bound on the probability of indefinitely-long paths. Consequently, those randomness constraints are trivially satisfied. In comparison, we consider the synthesis of policies that necessarily randomize in presence of stochastic behavior in the environment. The closest prior work is to ours is Reactive Control Improvisation (RCI) for (deterministic) 2-player games [55]. As in ERCI, RCI features three kinds of constraints; hard, soft, and randomness. As in ERCI, RCI can be preprocessed resulting in the following core problem.

The Core RCI Problem: Given a finite acyclic (deterministic) SG $\mathcal{G}$, with terminal states, $s_\top$ and $s_\bot$, and thresholds $\mathbf{p} \in (0, 1)$ and $\mathbf{h} \in [0, \infty)$, find a ego-policy σ_{ego} such that for every env-policy σ_{env}

1. (*soft constraint*) $\Pr(\text{last}(\xi) = s_\top \mid \sigma) \geq \mathbf{p}$,
2. (*randomness*) $\max_\xi \Pr(\xi \mid \sigma) \leq \mathbf{d}$,

where $\sigma = \langle \sigma_{\mathsf{ego}}, \sigma_{\mathsf{env}}\rangle$.

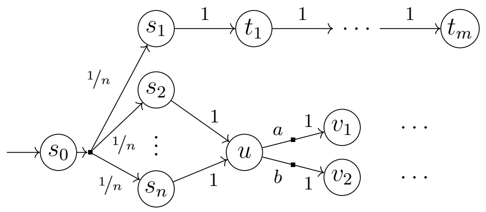


Figure 7.7: Example Illustrating the problem with RCI in stochastic environments.

While RCI is only applied to deterministic SGs in [55], there is nothing in the definition that prevents its application to the general class of SGs.[7] We observe that then, the only difference between ERCI and RCI is that we use causal entropy rather than an upper bound on the probability of a path to enforce randomness. Below we address two problems with bounding the maximum probability of a trace.

First, RCI fails to account for causality when measuring randomness. In deterministic systems, for which RCI was conceived, this distinction is unnecessary, but stochastic systems must deal with counter-factuals. In practice, RCI encodes an agent model that is systematically overly optimistic regarding the outcomes of dynamics transitions [96]. This results in policies with worse performance given a fixed randomness target. In the context of our motivating drone example, applying RCI thus results in a policy that is both quantitatively and qualitatively less random than the ERCI.

Second, RCI fails to enforce randomization if there exists *any* path with sufficiently high probability. The next (pathological) example illustrates.

Example 7.8.1. *Consider the SG (actually, an MDP where we omit the* env*-states) in Fig. 7.7. First consider that under each scheduler, the path from s_0 to t_m has probability $1/n$. In particular, this means that a feasible RCI instance (applied to an SG) must have $\mathbf{d} \geq 1/n$. At the same time, every path in the SG already has probability at most $1/n$, and thus, every scheduler that satisfies the randomness constraint for $\delta = 1$ satisfies it for any $\mathbf{d} \geq 1/n$. Thus, for this MDP, the RCI formulation fails to enforce any randomization in the* ego*-policy. By contrast, a causal entropy constraint from ERCI will continuously trade-off randomness for performance.*

On the other hand, one can observe that in reality, proposed algorithms for solving RCI equally distribute probability mass across the maximum number of paths that ego can guarantee [55]. We remark that because (1) causal entropy reduces to non-causal entropy in deterministic dynamics and (2) uniform distributions maximize entropy, our proposed entropy matching family exactly agrees with existing RCI algorithms on deterministic SGs. Thus, we observe the following proposition.

[7]However, this does not mean that the algorithm to compute a solution carries over to the general case

Proposition 7.8.2. There exists a computable function,

$$f\colon (\mathbf{d}, \mathcal{G}) \mapsto \mathbf{h},$$

such that, for any deterministic SG, $\mathcal{G}$, and performance threshold $\mathbf{p}$, there exists an ego-policy solving the RCI problem with threshold $\mathbf{d}$ iff there exists a ego-policy solving the ERCI problem with threshold $\mathbf{h} = f(\mathbf{d}, \mathcal{G})$.

Additional Related Work

Synthesis in MDPs with multiple hard and soft constraints (often over indefinite horizons) is a well-studied problem [32, 50, 53, 124]. In this setting, one generates deterministic policies and their convex combinations. Put differently, some degree of randomization is *not an objective*, but rather a consequence. Interestingly, in [46] the optimal policies in *absence* of randomization are investigated. Along similar lines, [20] trades average performance for less variance, thereby implicitly trading off the average and the worst-case performance. The original results sparked interest in different extension to MDPs and the type of soft constraints, such as continuous MDPs [65] and continuous-time MDPs [121], cost-bounded reachability [66], or mean-payoff properties [19]. The algorithms have also been extended towards stochastic games [34, 89]. Finally, notions of lexicographic multi-objective synthesis [31] – in which one optimizes a secondary criterion among all policies that are optimal with respect to a first criterion bare some resemblance with the algorithm we consider. The aforementioned algorithms have been put in a robotics context in [91]. Finding policies that optimize reward objectives is well-studied in the field of reinforcement learning, and has been extended to generate Pareto fronts for multiple objectives [106, 114].

Next, our core ERCI instance can be seen as a multi-objective path problem [9, 107, 159]. The literature on multi-object path finding differs prominently from ERCI in two aspects: they do not trade-off randomization and performance, and they do not trade-off declarative and formal constraints with the accompanying formal guarantees, but are more search-based.

Another related domain is the problem of (randomly) patrolling a perimeters and points of interest [4, 8, 118]. Closest to our work are formalisms rooted in game-theory, such as *Stackelberg games* [132, 115]. Stackelberg games have been extending to Stackelberg planning [137] in which a trade-off between the cost for the defender and the attacker can be investigated. Most related are the zero-sum *patrolling games* introduced in [6], which has led to numerous practical solutions [141]. Patrolling games are explicitly games between an intruder and a defender, and there is no stochastic environment. Adding additional objectives makes solving these problems harder [86] and in general, the obtained policies are no longer applicable. To overcome this, a specific set of fixed objectives has been added to these games recently [86]. The large common aspect in all of this work is that optimal strategies do randomize. As in the synthesis work above, this is a consequence of the objectives rather than an objective in itself. In comparison, we provide a general framework and in particular support stochastic environments.

Finally, entropy as an optimization objective for MDPs with fixed rewards has been well studied [127], particularly in the context of regularizing (robustifying) inverse and reinforcement learning [162, 57]. The primary distinction from our work (in the MDP setting) is the unspecified (performance/entropy) trade-off. Nevertheless, as previously discussed, the specification variant of this literature served as the basis for our MDP subroutine [153]. Beyond Markov models, the (uniform) randomization over languages in finite automata [71, 83] or over propositional formulae [80, 15, 29] has received quite some attention, however neither of those approaches support the notion of soft constraints or the related trade-offs.

Chapter 8

Stochastic Games as Circuits

We demand rigidly defined areas of doubt and uncertainty!

Douglas Adams (The Hitchhiker's Guide to the Galaxy, 1979)

The previous chapter developed algorithms for control improvisation (maximum causal entropy planning) in simple stochastic games. As the experiments and theoretical discussion showed, the performance of these algorithms depends linearly on the *size* of the game graphs. In the next two chapters, we will develop a systematic approach for representing compressed representations of these game graphs.

Contributions

In order to succinctly represent stochastic game for history dependent task specifications, we study modeling probabilistic transition systems as circuits. The result is a framework for analyzing queries about stochastic games, e.g., what is the probability of satisfying the task specification when applying actions uniformly at random, using tool such as Boolean Satisfiability (SAT) solvers [44] and Binary Decision Diagrams [23]. The first step of this approach will be to represent probabilistic systems as sequential circuits. We begin by defining bit-vector predicates.

Definition 27. A **bit-vector** is a string with alphabet $\{0, 1\}$. A **n-ary bit-vector predicate** maps n-bit vectors to $\{0, 1\} \subseteq \mathbb{R}$, e.g,

$$\varphi : \{0,1\}^n \to \{0,1\}. \tag{8.1}$$

If $\varphi(\boldsymbol{x}) = 1$, we additionally call $\boldsymbol{x}$ a **model** of φ. We define the **model count** of φ, denoted $\#(\varphi)$, as the number of models of φ, i.e,

$$\#(\varphi) \overset{\text{def}}{=} \sum_{\boldsymbol{x} \in \{0,1\}^n} \varphi(\boldsymbol{x}). \tag{8.2}$$

An illustration of such a predicate is shown in Fig 8.1. When the number of inputs of a predicate φ is unambiguous (or not important), we shall write $\varphi(\boldsymbol{x})$ as a logical sentence over $\boldsymbol{x}$, where True is mapped to 1 and False is mapped to 0.

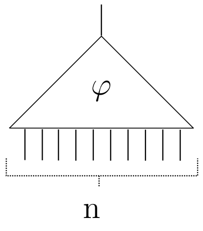


Figure 8.1: A n-ary bit-vector predicate as a circuit with n inputs.

Example 8.0.1. *Let $\varphi : \{0,1\}^{10} \to \{0,1\}$ denote the 10 input map,*

$$\varphi(\boldsymbol{x}) = x_1 \wedge x_7 \overset{\text{def}}{=} \begin{cases} 1 & \text{if } x_1 \wedge x_7 \\ 0 & \text{otherwise} \end{cases}. \tag{8.3}$$

Thus $\varphi(\boldsymbol{x}) = 1$ iff x_1 and x_7 are True (i.e., 1). Further observing that there are 8 other "don't care" inputs, each with two possible values, yields $\#(\varphi) = 2^8$.

Example 8.0.2. *Given $n \in \mathbb{N}$, let k be an integer between 0 and $2^n - 1$ and let $\varphi : \{0,1\}^n \to \{0,1\}$ be given by,*

$$\varphi(\boldsymbol{x}) = \boldsymbol{x} < k, \tag{8.4}$$

where $\boldsymbol{x}$ is interpreted as an integer between 0 and $2^n - 1$. Observe that $\#(\varphi) = k$ since there are only k unsigned integers less than k.

Observe that a circuit can be made probabilistic by feeding the results of random coin flips as inputs. To this end, we introduce notation for the process of generating a bit-vector using n unbiased coin flips.

Definition 8.0.3. Denote by $x_1x_2\dots x_n \sim \{0,1\}^n$ the act of creating an $n-$bit-vector by flipping n independent unbiased coins with:

$$\Pr_{\boldsymbol{x}\sim\{0,1\}^n}(x_i = 0) = \Pr_{\boldsymbol{x}\sim\{0,1\}^n}(x_i = 1) = \frac{1}{2}, \tag{8.5}$$

and thus, the probability of drawing any particular bit-vector, $\boldsymbol{x}^*$ is:

$$\Pr_{\boldsymbol{x}\sim\{0,1\}^n}(\boldsymbol{x} = \boldsymbol{x}^*) = \frac{1}{2^n}. \tag{8.6}$$

To study probabilistic statements, we make the following elementary observation.

Observation 1. Given an n-ary bit-vector predicate, φ, if one flips n independent unbiased coins, $\boldsymbol{x} \sim \{0,1\}^n$, the probability that $\varphi(\boldsymbol{x}) = 1$ is equal to the fraction of n-bit-vectors that are models of φ, i.e,

$$\Pr_{\boldsymbol{x}\sim\{0,1\}^n}(\varphi(\boldsymbol{x}) = 1) = \sum_{\boldsymbol{x}\in\{0,1\}^n} \frac{1}{2^n}\varphi(\boldsymbol{x}) = \frac{\#(\varphi)}{2^n}. \tag{8.7}$$

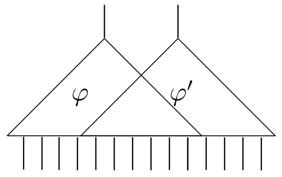
Therefore, if one wishes to compute $\Pr_{\boldsymbol{x}\sim\{0,1\}^n}(\varphi(\boldsymbol{x}) = 1)$ for some complicated φ, it suffices to use a model counter to compute (or approximate) $\#(\varphi)$. While straightforward, the power of this observation is only truly realized when one starts composing bit-vector predicates and reusing inputs. We illustrate this through two more observations.

Observation 2. Using (8.7), φ can be reinterpreted as a process to turn n unbiased coins into a *biased* coin.

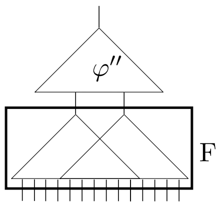
To emphasize Observation 2, we shall denote by $x \sim \varphi$ the process of drawing a biased coin, $x \in \{0,1\}$, using the distribution given in (8.7).

Observation 3. If the results of some coin flips are shared, $F : \boldsymbol{x} \mapsto (\varphi(\boldsymbol{x}), \varphi'(\boldsymbol{x}))$, then $F : \{0,1\}^n \to \{0,1\}^2$ models *correlated* coin flips.

Inspired by Observation 3 and letting F be a map between bit-vectors, $F : \{0,1\}^n \to \{0,1\}^m$, we denote by $\boldsymbol{x} \sim F$ the process of drawing m correlated biased coin flips. In particular, if $\varphi_i(\boldsymbol{x}) = F(\boldsymbol{x})_i$, then $\boldsymbol{x}$ is the concatenation of m bit-vectors such that, $x_i \sim \varphi_i$. Together, Observations 2 and 3 enable studying complex distributions via model counting.



(a) By sharing inputs, two bit-vector predicates, which model biased coins, can be used to model a pair of *correlated* coin flips.



(b) Feeding correlated biased coin flips into a bit-vector predicate yields a new bit-vector predicate, and thus models a biased coin flip.

Figure 8.2: Illustrations of Observations 2 and 3.

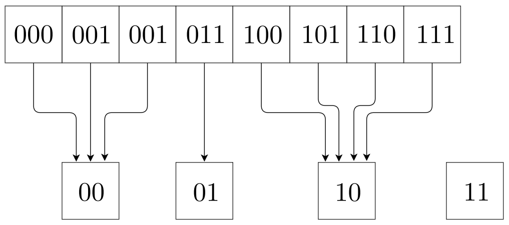


Figure 8.3: Visualization of $[\varphi \times \varphi'](\boldsymbol{x})$. The bit sequences have the most significant bit on the left and the least significant bit on the right, e.g., $011 = 3$.

Example 8.0.4. *Let φ and φ' denote the following 3-bit bit-vector predicates,*

$$\begin{aligned} &\varphi : \{0,1\}^3 \to \{0,1\} && \varphi' : \{0,1\}^3 \to \{0,1\} \\ &\varphi(\boldsymbol{x}) \overset{\text{def}}{=} \boldsymbol{x} = 3 && \varphi'(\boldsymbol{x}) \overset{\text{def}}{=} \boldsymbol{x} > 3 \end{aligned}, \tag{8.8}$$

where $\boldsymbol{x}$ is interpreted as an unsigned integer. Next, define $\varphi \times \varphi'$ as the product of φ and φ', i.e., $[\varphi \times \varphi'] : \boldsymbol{x} \mapsto \Big(\varphi(\boldsymbol{x}), \varphi'(\boldsymbol{x})\Big)$. The resulting map is illustrated in Fig. 8.3. Note that because φ and φ' share inputs, then the biased coins they model are correlated (Observation 3). In particular, using Fig. 8.3, we see that $\varphi \times \varphi'$ induces the following distribution over 2-bit-vectors:

$$\Pr_{\boldsymbol{y} \sim \varphi \times \varphi'}\Big(\boldsymbol{y} = k\Big) = \begin{cases} 3/8 & \textit{if } k = 0 \\ 1/8 & \textit{if } k = 1 \\ 1/2 & \textit{if } k = 2 \\ 0 & \textit{otherwise.} \end{cases} \tag{8.9}$$

Now suppose one wishes to compute the probability that $k > 0$ under (8.9). By (8.7), it suffices to compute the model count of $[\boldsymbol{x} > 0]$ composed with $\varphi \times \varphi'$,

$$\Pr_{\boldsymbol{y} \sim \varphi \times \varphi'}\Big(\boldsymbol{y} > 0\Big) = \frac{\#\Big([\boldsymbol{x} > 0] \circ [\varphi \times \varphi']\Big)}{2^3} = {}^5\!/\!{}_8. \tag{8.10}$$

Of course, in this case, it is easy to look at Fig. 8.3 to determine that $\#\Big([\boldsymbol{x} > 0] \circ [\varphi \times \varphi']\Big) = 5$. *However, in general, with bigger circuits and more complicated properties, this explicit reduction to model counting proves incredibly useful.*

Biased Coins and Bayes' Rule

So far we have focused on modeling probability distributions where the probability masses are (integer) multiples of $\frac{1}{2^n}$. Of course, many examples violate this assumption, e.g, a coin with a $\frac{1}{3}$ bias towards heads. We shall post-pone an explicit handling of biased coins until Ch 9 and instead note that probabilistic queries over unbiased coins can be answered by two model counting queries.

Proposition 8.0.5. Let $\varphi : \{0,1\}^n \to \{0,1\}$ and $\psi : \{0,1\}^n \to \{0,1\}$ denote any two bit-vector predicates. Then,

$$\Pr_{\boldsymbol{x} \sim \{0,1\}^n}(\varphi(\boldsymbol{x}) = 1 \mid \psi(\boldsymbol{x}) = 1) = \frac{\#(\varphi \wedge \psi)}{\#(\psi)}. \tag{8.11}$$

Proof. By the chain rule,

$$\Pr_{\boldsymbol{x} \sim \{0,1\}^n}(\varphi(\boldsymbol{x}) = 1 \mid \psi(\boldsymbol{x}) = 1) \cdot \Pr_{\boldsymbol{x} \sim \{0,1\}^n}(\psi(\boldsymbol{x}) = 1) = \Pr_{\boldsymbol{x} \sim \{0,1\}^n}([\varphi \wedge \psi](\boldsymbol{x}) = 1). \tag{8.12}$$

Replacing the unconditioned probabilities using (8.7) gives,

$$\Pr_{\boldsymbol{x} \sim \{0,1\}^n}(\varphi(\boldsymbol{x}) = 1 \mid \psi(\boldsymbol{x}) = 1) \cdot \frac{\#(\psi)}{2^n} = \frac{\#(\varphi \wedge \psi)}{2^n}. \tag{8.13}$$

Multiplying both sides by 2^n and rearranging yields (8.11). □ □

To avoid notational clutter, we shall frequently write (8.11) using the sampling notation, $y \sim \varphi$, previously introduced, but additionally condition on ψ,

$$\Pr_{y \sim \varphi}(y \mid \psi) \overset{\text{def}}{=} \Pr_{\boldsymbol{x} \sim \{0,1\}^n}\Big(\varphi(\boldsymbol{x}) = 1 \mid \psi(\boldsymbol{x}) = 1\Big) \tag{8.14}$$

Example 8.0.6. *Again, suppose we seek to find a pair* φ, ψ *that encodes a biased coin with probability* $1/3$ *of coming up* 1. *Observe that this can be accomplished by letting* $\varphi(\boldsymbol{x}) \overset{\text{def}}{=} (\boldsymbol{x} = 0), \psi(\boldsymbol{x}) \overset{\text{def}}{=} (\boldsymbol{x} < 3)$, *such that,*

$$\Pr_{y \sim \varphi}(y_i \mid \psi) = \frac{\#(\boldsymbol{x} = 0 \wedge \boldsymbol{x} < 3)}{\#(\boldsymbol{x} < 3)} = \frac{1}{3}, \tag{8.15}$$

where $\boldsymbol{x} \in \{0,1\}^2$ *is encoded as an unsigned integer.*

Remark 8.0.7. In many contexts, $\#(\psi)$ can be precomputed, sometimes even without the use of a model counting algorithm.

Remark 8.0.8. Computing the model count of a circuit is known to be $\#P$-complete [13]. Nevertheless, in practice, moderately efficient methods for computing or approximating $\#(\bullet)$ exist including Monte Carlo simulation [104] and weighted model counting [33] via Binary Decision Diagrams (BDDs) [21] or repeated SAT queries [28].

Encoding Rational Coins

Prop 8.0.5 provides guidance on how Ex 8.0.6 can be generalized to encode an arbitrary coin with a rational bias. Consider a coin, y, such that $\Pr(y = 1) = \frac{k}{m}$, for some $k, m \in \mathbb{N}$. Letting n be the smallest integer such that $m \leq 2^n$, and recalling that $\boldsymbol{x} < k$ has exactly k models (Ex. 8.0.2), observe that y corresponds to feeding n unbiased coins into $\varphi(\boldsymbol{x}) \overset{\text{def}}{=} \boldsymbol{x} < k$ and conditioning on $\psi(\boldsymbol{x}) \overset{\text{def}}{=} \boldsymbol{x} < m$. Finally, observing that $\boldsymbol{x} < k$ implies that $\boldsymbol{x} < m$ yields,

$$\Pr_{y \sim \varphi}(y = 1 \mid \psi) = \frac{\#(\boldsymbol{x} < k)}{\#(\boldsymbol{x} < m)} = \frac{k}{m}. \tag{8.16}$$

Finally, since $\Pr_{\boldsymbol{y} \sim F}(\varphi(\boldsymbol{y}) \mid \psi, \psi') = \Pr_{\boldsymbol{y} \sim F}(\varphi(\boldsymbol{y}) = 1 \mid \psi \wedge \psi')$ then (8.16) naturally extends to modeling multiple input conditioned coin flips. Of course, biased coins are not very interesting by themselves. Nevertheless, as illustrated in Ex. 8.0.4, feeding multiple correlated coin flips into another circuit enables studying more sophisticated objects. Further, we will later incorporate a notion of state which will enable answering non-trivial queries about probabilistic systems via model counting.

8.1 Distributions over Finite Sets

We now return to the topic of modeling distributions over finite sets Formally, we first seek to systematically solve the following problem:

Problem 8.1.1. Let Y be a finite set whose elements, $\hat{y}_i$, are numbered from 1 to $|Y|$, and associate to Y the following rational valued probability distribution:

$$\Pr(\hat{y}_i) = {}^{a_i}\!/_{m}, \tag{8.17}$$

where $a_i, m \in \mathbb{N}$ such that $\sum_{i=1}^{|Y|} a_i = m > 0$. Further, denote by $\boldsymbol{y} \in \{0,1\}^{|Y|}$ the 1-hot encoding of elements of Y, e.g. $y_i = 1$ iff $\boldsymbol{y}$ corresponds to $\hat{y}_i$. Find an $n \in \mathbb{N}$, an n-bit-vector function $F : \{0,1\}^n \to \{0,1\}^{|Y|}$, and a n-bit-vector predicate ψ, such that:

$$\Pr(\hat{y}_i) = \Pr_{\boldsymbol{y} \sim F}(y_i = 1 \mid \psi) \tag{8.18}$$

Note that the use of a 1-hot encoding is without loss of generality, since one can always feed this encoding into a circuit that transforms it into another encoding.

Common Denominator Method

Many techniques can be used to solve Prob 8.1.1. Here we outline a straightforward generalization of encoding a biased coin (8.16). The key idea is to encode $|Y|$ mutually exclusive biased coins, which together, form a 1-hot encoding of $\hat{y}_i$. To begin, let n be the smallest integer such that $m \leq 2^n$. For convenience, define

$$b_0 \overset{\text{def}}{=} 0 \qquad b_{i+1} \overset{\text{def}}{=} b_i + a_i. \tag{8.19}$$

Now, let $\varphi_i : \{0,1\}^n \to \{0,1\}$ denote the circuit,

$$\varphi_i(\boldsymbol{x}) \overset{\text{def}}{=} b_i \leq \boldsymbol{x} < b_i + a_i \tag{8.20}$$

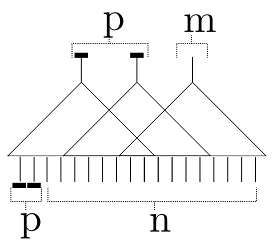
where $\boldsymbol{x}$ is interpreted as an unsigned integer. Further, note that by construction, $\#(\varphi_i) = a_i$ and the φ_i are mutually exclusive. Thus, the product of all φ_i results in a 1-hot encoding. Namely, letting $F : \{0,1\}^n \to \{0,1\}^{|Y|}$ denote $\varphi_1 \times \ldots \times \varphi_{|Y|}$ and $\psi(\boldsymbol{x}) \overset{\text{def}}{=} \boldsymbol{x} < m$ yields,

$$\Pr_{\boldsymbol{y} \sim F}(y_i = 1 \mid \psi) = {}^{a_i}/_{m}, \tag{8.21}$$

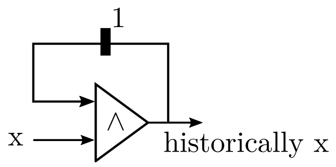
as desired.

8.2 Sequential Circuits

Ultimately, we want study sequential probabilistic systems, e.g., Stochastic Games. We begin by formalizing sequential circuits.



(a) A sequential circuit where the first p bits of input and output are marked with black rectangles to indicate that they represent the previous and next state, respectively.



(b) Sequential circuit testing if x is currently 1 *and* ($\wedge$) if x has historically been 1. In this illustration, the latch ($\boldsymbol{s}_0 = 1$) is shown cutting the cyclic dependency of the circuit.

Figure 8.4: Sequential circuit illustration and example.

Definition 28. Let n, m, and p denote natural numbers. A **sequential circuit** is a tuple, $C = (\boldsymbol{s}_0, F)$, where $F : \{0,1\}^{p+n} \to \{0,1\}^{p+m}$ is the transition function, and $\boldsymbol{s}_0 \in \{0,1\}^p$ is the initial state. To every sequence of inputs $\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots \in \{0,1\}^n$, associate a sequence of states $\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots \in \{0,1\}^p$ and outputs $\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots \in \{0,1\}^m$ by:

$$\boldsymbol{s}_i \mathbin{.} \boldsymbol{y}_i = F(\boldsymbol{s}_{i-1}, \boldsymbol{a}_i) \tag{8.22}$$

Finally, if $m = 1$, we refer to C as a **monitor**.

Example 8.2.1. *Figure 8.4b illustrates a sequential circuit that checks if x has been constantly 1. Formally if $\varphi(\boldsymbol{x}) \overset{\text{def}}{=} x_0 \wedge x_1$, then $F(\boldsymbol{x}) \overset{\text{def}}{=} \varphi(\boldsymbol{x}).\varphi(\boldsymbol{x})$ and $\boldsymbol{s}_0 = 1$. Note that as circuits can reuse outputs, in Fig. 8.4b $\varphi(\boldsymbol{x})$ is only computed once.*

Now observe that we can reduce the execution of a fixed number of steps of a sequential circuit, $C = (\boldsymbol{s}_0, F)$, back to a bit-vector function simply by composing F with itself, akin to bounded model checking [17].

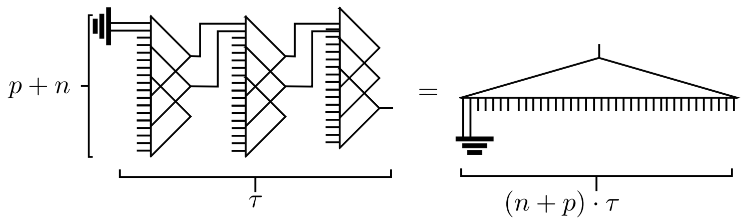


Figure 8.5: The sequential circuit of Fig. 8.4a unrolled for 3 steps. As in Fig. 8.4a, the first two inputs and outputs of each copy of F denote the state. Note that the first copy of F has its state inputs grounded to denote that $\boldsymbol{s}_0 = (0, 0)$.

Definition 29. Let $C = (\boldsymbol{s}_0, F)$ denote a sequential circuit with n inputs, p states, and m outputs and let $\text{last}_m : \{0,1\}^{p+m} \to \{0,1\}^m$ denote the bit-vector function that returns the last m bits of input. For all times $\tau \in \mathbb{N}$, define the **τ-unrolling** of C, to be the map:

$$\begin{aligned} &U_C^\tau : \{0,1\}^{\tau \cdot n} \to \{0,1\}^m \\ &U_C^\tau(a_1.a_2.\ldots.a_\tau) \overset{\text{def}}{=} \text{last}_m \circ F(\ldots F(F(F(x_0, a_1), a_2), a_3), \ldots, a_\tau) \end{aligned} \tag{8.23}$$

where each a_i denotes a bit-vector in $\{0,1\}^n$.

Since unrolling a monitor results in a Boolean predicate, Observations 1, 2, and 3 naturally extend to the sequential circuits. This suggests extending our notation for sampling a coin to

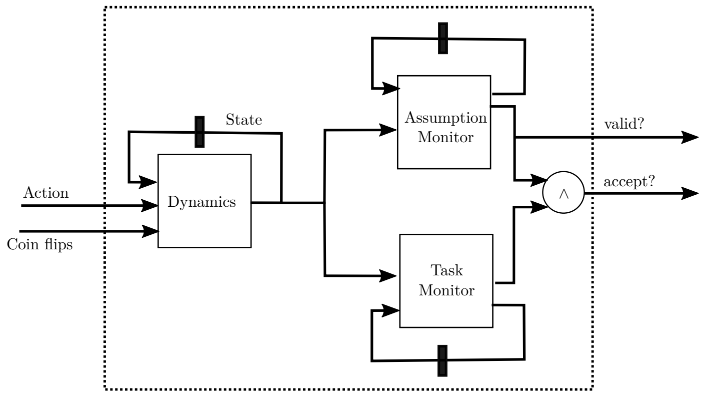


Figure 8.6: Recipe for modeling dynamical systems as sequential circuits.

sequential circuits. Given a sequential circuit C and a monitor ψ to condition on, we define $\boldsymbol{x} \overset{\tau}{\sim} C$ so that:

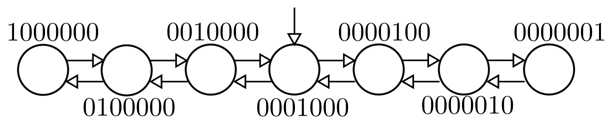
$$\Pr_{\boldsymbol{x} \overset{\tau}{\sim} C}(\boldsymbol{x} \mid \psi) \overset{\text{def}}{=} \Pr_{\boldsymbol{x} \sim U_C^\tau}(\boldsymbol{x} \mid U_\psi^\tau) \tag{8.24}$$

Encoding Stochastic Games

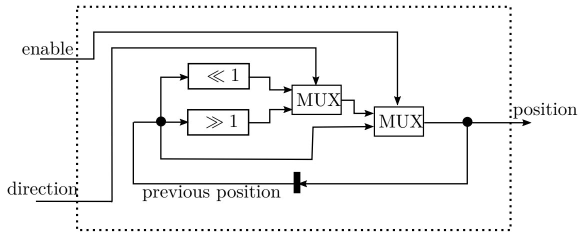
The key utility of Eq. (8.24), is that it enables studying stochastic games via tools like SAT solvers and Binary Decision Diagrams (Ch 9). For example, as shown in Fig 8.6, the combination of a task φ and a stochastic game, $M = \langle S, s_0, A, P\rangle$, can be modeled as three sequential circuits: the dynamics (P), a task monitor, and an assumption monitor. The dynamics circuit corresponds models the distribution P. As such, the actions inputs are partitioned into ego and env inputs. The task monitor takes as input the current state and outputs 1 iff sequence of states seen would be accepting if the episode ended. Similarly, the assumption monitor checks if the sequence of states would satisfy the assumption ψ if the episode ended. If the dynamics circuit is independent of the current action inputs, then the circuit corresponds to a Markov chain. Similarly, if the dynamics circuit is independent of the env inputs, then the circuit corresponds to a MDP.

Remark 8.2.2. Recall that when maximum causal entropy coincides with maximum entropy (4.11), the forecasting policy π_φ and likelihood of the demonstrations depend entirely (Prop 4.4.1, (4.16)) on the estimated competency of the agent, p_φ, and the competency of an agent applying actions uniformly at random, q_φ. Using a circuit representation of the dynamics, computing q_φ reduces to a model counting query.

Example 8.2.3. *Consider the 1-d variant of the classic drunken sailor random walk. A sailor walks along a pier, where with each step, the sailor either stumbles forward by one plank, backward by one plank, or remains on the same plank. Further, suppose the pier is 11 planks long and that if the sailor visits the central plank more than 3 times, the plank will break and the sailor will fall into the water. If the sailor starts on the middle plank and the* *probability of moving forward is* $2/6$*, moving backward is* $1/6$*, and not moving is* $3/6$*, what is the probability the sailor breaks a plank after* 10 *steps?*



(a) Illustration of "1-hot" encoding of a chain graph. The right and left arrows represent arithmetic right ($\gg 1$) and left ($\ll 1$) shifts of the state respectively.

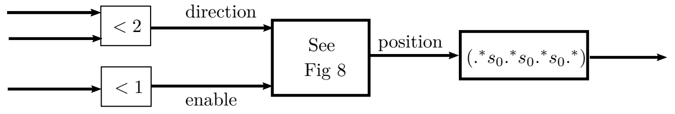


(b) "1-hot" encoding of a chain graph as a sequential circuit. The MUX gates use their top input to select which of their two other inputs to output.

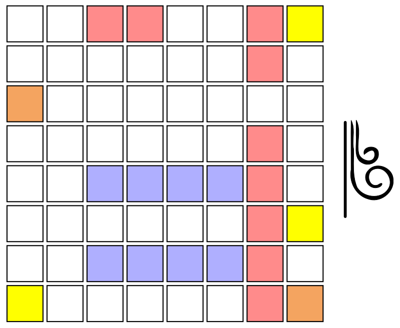
Within the above framework, the dynamics corresponds the a 1-d finite chain (see Fig. 8.7a). An example encoding of such a chain as a sequential circuit is given in Fig. 8.7b. Similarly, the monitor is a sequential circuit for the regular language $(.^*\boldsymbol{s}_0.^*\boldsymbol{s}_0.^*\boldsymbol{s}_0.^*)$ *which can be efficiently compiled into a sequential circuit [145]. Finally, the policy corresponds to some circuit that models the probability distribution over actions which, using the inputs in Fig. 8.7b), corresponds to modeling two independent biased coin flips, namely,* $\Pr(enable = 0) = \frac{1}{2}$ *and* $\Pr(direction = 0) = \frac{2}{3}$*. As shown in the previous section, namely (8.16), we can model the direction coin by feeding the output of* $\varphi_{direction}(\boldsymbol{x}) = \boldsymbol{x} < 2$ *into the direction input shown in Fig. 8.7b, and conditioning on* $\boldsymbol{x} < 3$*, where* $\boldsymbol{x} \in \{0,1\}^2$*. Similarly, using a disjoint set of inputs, on can encode the enable coin by feeding the output of* $\varphi_{enable}(\boldsymbol{x}') = \boldsymbol{x}' < 1$ *into the enable input of Fig. 8.7b. The resulting sequential circuit,* C_{ds} *is summarized in Fig. 8.8a.*

Letting C_{valid} *denote the monitor that for all time steps,* $\boldsymbol{x} < 3$*, then the probability of the sailor falling into the water within the first 50 steps is given via:*

$$\Pr\left(\textit{sailor falls into water}\right) = \Pr_{\substack{50\\x \sim C_{ds}}}\left(x = 1 \mid C_{valid}\right) = \frac{25398396}{6^{10}} \approx 0.42 \tag{8.25}$$



(a) Drunken Sailor policy circuit, C_{ds}.



(b) Running example gridworld.

Example 8.2.4. *Consider modeling the dynamics of the 8×8 gridworld of our running example. As with the previous example, we use a 1-hot encoding of position. In particular let states $s \in S$ be represented as a bit-vector $s = x \centerdot y$ where $x, y \in \{0,1\}^8$. Next, we represent the action input as bit-vector, $a = a_1 \centerdot a_2 \in \{0,1]^2$ where a_1 determines along which coordinate, x or y, the agent moves and a_2 determines the direction, forward or backwards. Finally, let c denote a coin flip inputs $c \in \{0,1\}$ with probability $1/32$ to being 1. This will encode that there is a $1/32$ chance of slipping and moving down in the gridworld. The dynamics circuit can then be modeled as the following function:*

$$\hat{P}(x \centerdot y, a, c) = \begin{cases} \hat{P}(x \centerdot y, 1 \centerdot 0, 0) & \textit{if } c = 1 \\ (x \ll 1) \centerdot y & \textit{else if } a = 0 \centerdot 0 \\ (x \gg 1) \centerdot y & \textit{else if } a = 0 \centerdot 1 \\ x \centerdot (y \ll 1) & \textit{else if } a = 1 \centerdot 0 \\ x \centerdot (y \gg 1) & \textit{else if } a = 1 \centerdot 1 \end{cases}. \tag{8.26}$$

To enforce the Luce axiom and keep the agent from leaving the grid, one can introduce an assumption circuit that monitors that x and y never become 0. Asserting that the left-shift ($\ll$) and right-shift ($\gg$) logical shift implies that $1 \centerdot 0 \ll 1 = 0$ and $0 \centerdot 1 \gg 1 = 0$, we have that the this condition enforces that $A(s)$ only contains actions that will not leave the grid.

8.3 Bibliographic Notes

With an eye towards realizing a maximum entropy planner, we systematically developed a framework for modeling probabilistic systems as model counting problems. Starting from unbiased coins, we constructed biased coins, correlated coins, and conditional probabilities. We discussed how to model arbitrary distributions over finite sets and how to combine our building blocks into sequential systems. While the building blocks discussed in this work have been used in previous works (e.g. [149, 122]), we believe that the explicit discussion of the modeling techniques in this work will enable future case studies on probabilistic systems with SAT-based model counting algorithms.

This work connects with literature in two primary ways. First and foremost, we provide an encoding of probabilistic systems as sequential circuits, which when unrolled, are suited to be analyzed with model counting algorithms. Numerous encodings of probabilistic systems as *weighted* model counting problems have been proposed [126, 33, 43], which has then spurred the adaptation of a number of unweighted model counting algorithms to solve weighted model counting problems [126, 35, 26]. Unfortunately, such adaptations require expert knowledge of the inner workings of model counters, making the transfer of advancements from unweighted model counting to weighted model counting difficult. Hence, techniques for efficient and automated reductions from weighted model counting to unweighted model counting have been proposed [27]. In many ways, this chapter continues in this direction, by providing a framework for encoding probabilistic systems and inferences on said systems *directly* into unweighted model counting problems. Such reductions are particularly appealing given that, to our knowledge, there is no major algorithmic advantage in using weighted over unweighted model counting algorithms.

Further, this work is intimately related to the work on simulating discrete distributions using a stream of random bits. This framework, called the random bit model and first introduced by von Neumann [110], has gone on to spawn numerous techniques (for a more detailed survey, we point the reader to [100]). One of the goals in this chapter has been to illustrate how to draw from this vast literature to create new encodings of probabilistic circuits.

Chapter 9

Stochastic Games as BDDs

BDDs are somehow special in the "Art of Computer Programming" because they weren't any where near on the scene in 1962. It's really one of the only really fundamental data structures that came out in the last 25 years.

Donald Knuth (Computer Scientist, June 5 2008)

In the previous chapter, we studied modeling probabilistic systems, and particularly Stochastic Games, as abstract circuits. In this chapter, we study using Binary Decision Diagrams (BDDs) [22, 23] to represent the directed acyclic game graphs encoded in these circuits. The structure of this chapter then is to first introduce BDDs, their relevant properties, and operations.[1]

Contributions

With this machinery in hand, we study how BDDs realize compressed representations of the directed acyclic graphics used when constructing improvisers (Ch 7). This serves as the final piece in the description of the maxEntPlanner (4.6). Furthermore, as we saw in the previous chapter, model counting provides a means to compute q_φ, even for very small values. Here we shall see how BDDs admit transforming model counting to probabilistic reachability, and support computing q_φ directly.

Next, given these BDDs, the natural question is how large they can be? To this end, we provide conservative size bounds that shows these BDDs grow at most linearly in the horizon. Furthermore, this compressed encoding of time-unrolled stochastic games can be repeated expansion of the time horizon. This keeps the dynamics model compressed throughout the entire planning process. Finally, we conclude with the observation that BDDs provide a natural means to detect and enforce of the Luce axiom as per Sec 5.6. Without further ado!

[1]For a more thorough introduction to BDDs, we point the reader to [87].

Let n be a natural number. A **Binary Decision Diagram** (BDD) over n variables is a finite rooted directed acyclic graph, $G = (V, E)$ satisfying four conditions:

1. Nodes and edges are **labeled** with a natural number between 0 and $n + 1$, i.e.,
$$\text{label} : V \cup E \to [0 \ldots n + 1].$$
2. A path, ξ, in G never visits a label twice, i.e.,
$$v, v' \in \xi \implies \text{label}(v) \neq \text{label}(v').$$
3. Non-sink nodes have exactly two edges, one labeled 0 and the other labeled 1.
4. A node is a sink iff its label is 0 or 1.

A BDD is said to be **ordered** if along any path, $v_1 \to \ldots \to v_m$, nodes have strictly decreasing labels, i.e., $\text{label}(v_1) > \ldots > \text{label}(v_m)$. A BDD is said to be **reduced** if it contains no isomorphic subgraphs. Unless otherwise stated, we shall assume BDDs are ordered and reduced. Finally, let f be a n-bit bit-vector predicate. A BDD is said to **recognize** f if for all $x_n, \ldots, x_2 \in \{0, 1\}$, $f(x_n, \ldots, x_2) = 1$ iff there exists a path $v_1 \to \ldots \to v_m$ from the root to a sink such that:
$$\Big(\text{label}(v_i) = j > 1\Big) \implies \Big(\text{label}(v_i, v_{i+1}) = x_j\Big).$$

Example 9.0.1. *The BDDs for the predicates* $0 : x \mapsto 0$ *and* $1 : x \mapsto 1$ *have exactly one node labeled* 0 *and* 1 *resp. These are denoted* $\mathcal{B}_0$ *and* $\mathcal{B}_1$*.*

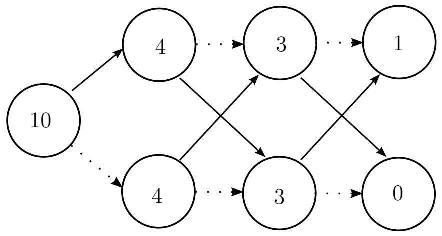


Figure 9.1: The BDD for the predicate $x_{10} + x_4 + x_3 \equiv 1 \mod 2$. Nodes are annotated with their label. Edges are solid and dotted if they have label 1 or 0 resp.

Example 9.0.2. *An example 9 variable BDD for the predicate,* $x_{10} + x_4 + x_3 \equiv 1 \mod 2$*, is shown in Fig 9.1. Nodes are explicitly labeled and edges labeled 1 are represented as solid lines and edges labeled 0 are represented by dotted lines. Observe that any path to* 1 *traverses an even number of solid lines.*

Properties of BDDs

BDDs have a number of useful properties which make them a popular and powerful representation of Boolean predicates.

Connection to automata and formal languages

In many ways, the structure of a BDD mirrors that of a DFA. This connection can be made precise by another related structure called a Quasi Reduced Binary Decision Diagram (QDD) [87]. QDDs are equivalent to BDDs except that variables cannot be skipped, i.e., every variable must appear on paths from the root to a sink. Logically, this is the same as the traversal of edges to consuming inputs for decision variables that are skipped over. Thus, a QDD is a DFA, where the node labeled 1 is the only accepting state! The language of this DFA is exactly $\{x \centerdot y \in \{0,1\}^n \centerdot \{0,1\}^* \mid f(x) = 1\}$. Hence why we say that a BDD recognizes f. Furthermore, note that since the BDD is assumed reduced, each state in this DFA must access a unique residual language. Thus the DFA corresponding to a given BDD (QDD) is also minimal. Finally, most of the subsequent properties of BDDs follow directly from this connection. As such, we shall often use terminology from DFA when discussing BDDs, e.g., access strings.

Operations on BDDs

BDDs support a number of operations such as conjunction, disjunction, exclusive or, negation and function composition. These operations can all be implemented by simple graph manipulations - each of which has polynomial time complexity (approximately linear or quadratic) in the size of the BDDs. Thus, for many Boolean predicates, BDDs offer a succinct and efficient form for analysis.

Reduced Ordered BDDs are canonical

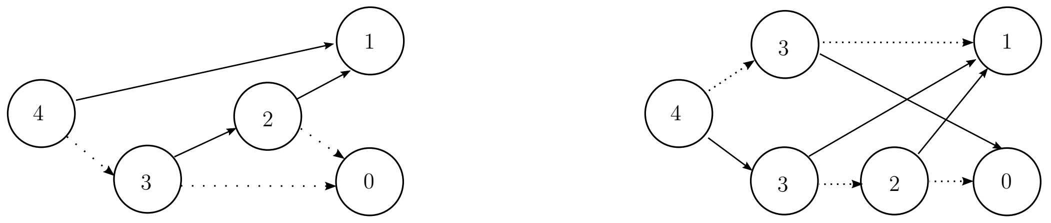
Following from correspondence with minimal DFAs, reduced ordered BDDs are canonical, i.e., given predicate $f(x_n, \ldots, x_2)$, there is exactly one BDD that recognizes f. We shall denote this BDD by $\mathcal{B}_f$. A useful consequence is that the Boolean Satisfaction query :

$$\exists x \,.\, f(x) = 1,$$

is equivalent to checking if $\mathcal{B}_f = \mathcal{B}_0$. Recalling that $\mathcal{B}_0$ has exactly one node, we see that this check takes constant time (after constructing the BDD!) Similarly, checking if two predicates f and g are equivalent becomes the same as checking if $\mathcal{B}_{f \oplus g} = \mathcal{B}_0$, where $\oplus$ denotes exclusive or. Since $\oplus$ can be realized by graph manipulation, we see that equivalence checks are also simple graph manipulations.

BDDs depend on variable ordering

Note that BDD uniqueness is only true given the ordering of variables in f. For example, let f be the predicate for $(x_2 \wedge x_3) \vee x_4$ and g be the predicate for $(x_2 \wedge x_4) \vee x_3$. As Fig 9.2 illustrates the f and g have different BDDs. Of note is that the size of the BDD depends on the variable ordering.



(a) BDD for $(x_2 \wedge x_3) \vee x_4$. **(b)** BDD for $(x_2 \wedge x_4) \vee x_3$.

Figure 9.2

Model Counting, Probabilistic Reachability, and BDDs

In the previous chapter, model counting served as a central primitive for analyzing probabilistic circuits. Below, we show how model counting of a predicate $f : \{0,1\}^n \to \{0,1\}$ can be efficiently performed by probabilistic reachability in $\mathcal{B}_f$. While slightly non-standard, this perspective will also make clear that the random action competency, q_φ, can be directly computed using the same algorithm.

To start, interpret $\mathcal{B}_f$ as a MDP, M, where actions are selected by flipping biased coins associated with each input. That is, the states of M are the nodes of $\mathcal{B}_f$, the actions of M have $A(s) = \{0,1\}$ if s is not a sink node. The transition probabilities follow: $P(s' \mid s, a)$ is 0 if s and s' are not connected and $b_i \in [0,1]$ if $\text{label}(s) = i$ and $a = 1$. Observe that the probability of reaching the state s^* with label 1 in M can be expressed as the following dynamic programming scheme [90],

$$\Pr(\text{reach}(s^*) \mid s) = \begin{cases} 1 & \text{if label(s)} = 1 \\ 0 & \text{if label(s)} = 0 \\ b_s s'_1 + (1 - b_s) s'_0 & \text{otherwise,} \end{cases} \tag{9.1}$$

where b_s denotes $b_{\text{label}(s)}$ and s'_0, s'_1 denote the child of s accessed by 0 and 1 resp. Next, noting that the probability of reaching label 1 from the root corresponds to that satisfaction probability. Thus if $b_s = 1/2$ for all interior states, then:

$$\Pr(\text{reach}(s^*) \mid s) = \Pr_{\boldsymbol{x} \sim \{0,1\}^n}(f(\boldsymbol{x}) = 1) = \frac{\#(f)}{2^n}. \tag{9.2}$$

Thus, if $b_i = 1/2$ then $\#(f) = 2^n \Pr(\text{reach}(s^*) \mid s)$, as desired. Further, note that if the f encoded the unrolled probabilistic circuit testing satisfying the task in a workspace MDP, then

$$\Pr(\text{reach}(s^*) \mid s) = q_\varphi,$$

enabling efficient maximum entropy planning when (4.11) holds.

Multi-Terminal BDDs

So far we have focused on the view of BDDs modeling Boolean predicates,

$$f : \{0,1\}^n \to \{0,1\}.$$

Now suppose one wants to model a multi-valued function, e.g.,

$$g : \{0,1\}^n \to \{0,1,\ldots,m\}.$$

There exist several variants of BDDs to directly encode such functions. Observe however that multi-terminal BDDs can also encoded as BDDs [139]. In particular, introduce auxiliary variables, $y_1,\ldots,y_m$, and create a function $f = h(g(x), y_1,\ldots,y_m)$ where:

$$h(z, y_1,\ldots,y_m) = \begin{cases} 1 & \text{if } z = i \wedge y_i = 1 \wedge \forall j \neq i \ . \ y_i = 0 \\ 0 & \text{otherwise.} \end{cases} \tag{9.3}$$

Note that in $\mathcal{B}_h$, the new auxiliary effectively act as new terminals. In particular, if $g(x) = i$, then the final non-terminal node in the BDD must be y_i since the other output variables have no effect.

Remark 9.0.3. Several other encodings of the same idea exist, e.g., using a base 2 encoding of $g(x)$ instead of the 1-hot encoding implicit above. For our purposes, the important property is that the BDD accessed by setting the input variables accesses a sub-BDD that maps directly to one of the outputs.

9.1 Encoding Stochastic Games

As mentioned at the start of the chapter, our goal will be to represent game graphs of SGs using BDDs. In this section, we describe the encoding and construction of such BDDs (with an overview provided in Fig 9.3). To start, assume that our planning horizon is τ and all complete paths are of lengths $\tau \in \mathbb{N}$ (using the padding trick from Ch 3). Next, recall from the previous chapter that any finite SG can be expressed as a sequential circuit, C, with n inputs and m. Let the first q inputs of C correspond to coins with bias $(b_1,\ldots b_q)$ and the remaining $n-q$ inputs encode the action. Similarly, the first output bit will denote the output of the assumption monitor and the remaining $m-1$ bits encode the state S. Next, observe that because the planning horizon is finite, any task restricted to this horizon must

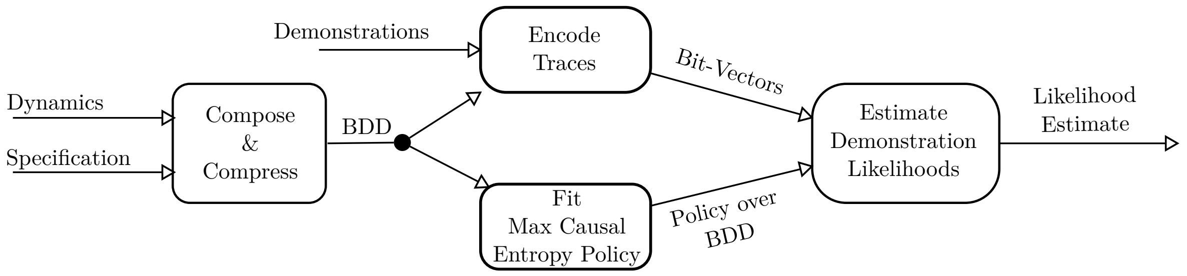


Figure 9.3: High-level likelihood estimation procedure described in this chapter.

be regular - and thus monitorable by a sequential circuit[2]. Unrolling the composition of C and the task monitor circuit τ-steps, yields a Boolean predicate:

$$\widehat{\varphi} : \{0,1\}^{\tau \cdot n} \to \{0,1\}. \tag{9.4}$$

Causal Orderings

When building the BDD for $\widehat{\varphi}$, denoted $\mathcal{B}_{\widehat{\varphi}}$, we assert that the order of inputs of $\widehat{\varphi}$ satisfies three conditions to reflect the decision process being modeled. First, the inputs should be temporally ordered, i.e., inputs corresponding to earlier time steps have larger labels. Second, within a given time step, the ego-action bits appear before the env-action bits. Third, within a given time step, the action bits appear before the coin flip decisions. We refer to such an orderings as **causal** since they respect the way in which information is revealed to the ego-agent operating the SG represented by $\mathcal{B}_{\widehat{\varphi}}$.

Paths to bits

Viewing the BDDs as automata, the causal orderings take the form:

$$\Big(\{0,1\}^{n_1} \centerdot \{0,1\}^{n_2} \centerdot \{0,1\}^{q}\Big)^{\tau}, \tag{9.5}$$

i.e., repeatedly process the first n_1 ego-decision variables, then the n_2 env-decision variables, and then q-coin flips. This defines map then from paths to bit-vectors:

$$\text{encode} : \text{Paths}_{\$} \to \Big(\{0,1\}^{n_1} \centerdot \{0,1\}^{n_2} \centerdot \{0,1\}^{q}\Big)^{\tau}. \tag{9.6}$$

This implies that $\mathcal{B}_{\widehat{\varphi}}$ is a compression of an SG where a single action now corresponds to $n - q$ actions and a probabilistic transition corresponds to flipping q coins.

[2]In particular, the DFA of a regular language is easily converted into a sequential circuit since both the transition function and accepting set can be realized as lookup tables.

Enforcing hard constraints

At this point, we have described how to encode a SG and a task specification as a BDD. Recall, however, that the ERCI problem required handling both a hard constraint, ψ, and soft constraint, φ. To do this, the pre-processing step on stochastic game graphs in Ch 7 removed all ego-actions where it was impossible for ego to avoid the possibility of violating the hard constraint. Importantly, this requires summarizing which paths violate the hard constraint. Observe then that three terminal BDDs offer a direct means to encode whether a particular (i) satisfies φ, (ii) does not satisfy φ, or (iii) violates an assumption (ψ) monitored by the assumption circuit. Thus, the BDD can be post-processed to enforce satisfying ψ by a single pass starting at terminal (iii) and removing ego decisions where there is a set of ego and coin flip decisions that lead to (iii).

Smooth Bellman Backups

Finally, before discussing the construction and size of $\mathcal{B}_{\widehat{\varphi}}$, we address the question of how to adjust the smooth Bellman backup for the fact that:

1. The BDD game graph skips over certain decisions.

2. Selected actions and states outcomes are revealed over multiple decisions.

To this end, recall that in the variable ordering, nodes alternate between ego, env, and coin flip decisions. Viewed as a computation graph, the nodes of the (uncompressed) stochastic game would alternate between taking a LSE, min, and expectation of child values resp. To address point two above, recall that these operations are all associative (and commutative) in their arguments, i.e., $\max(\max(x, y), z) = \max(x, \max(y, z))$ and $\text{LSE}(\text{LSE}(x, y), z) = \text{LSE}(x, \text{LSE}(y, z))$. Thus, the smooth value is well defined when split over consecutive decisions.

Similarly, to address the first point, observe that, by definition, if a node is skipped in $\mathcal{B}_{\widehat{\varphi}}$, then it must have been inconsequential, i.e., the state or state-distribution indexed is the same either way. Thus the path's reward must have been independent of the decision made at that node. Therefore, the eliminated $\text{LSE}, \min$, and expectations must have been over a constant value - otherwise the eliminated sequences would be distinguishable w.r.t φ (or ψ). The result is summarized in the following identities, where α denotes the value of an eliminated node's children.

$$\text{softmax}(\overbrace{\alpha, \ldots, \alpha}^{|A|}) = \log(e^\alpha + \ldots + e^\alpha) = \ln(|A|) + \alpha, \tag{9.7}$$

$$\mathop{\mathbb{E}}_x[\alpha] = \sum_x p(x)\alpha = \alpha, \tag{9.8}$$

and

$$\min(\alpha, \ldots, \alpha) = \alpha. \tag{9.9}$$

Of course, it could also be the case that a sequence of nodes is skipped in $\mathcal{B}_{\widehat{\varphi}}$. Using (9.7), one can compute the change in value, Δ, that the eliminated sequence of k ego-nodes and any number of env and coin flip nodes would have contributed:

$$\Delta(k,\alpha) = \ln(2^k) + \alpha = k\ln(2) + \alpha \tag{9.10}$$

Crucially, evaluation of this compressed computation graph is linear in $|\mathcal{B}_{\widehat{\varphi}}|$.

Remark 9.1.1. One might wonder if skipped decisions point to a violation of the Luce axiom since the action makes no difference. Note that this is not necessarily the case as there need only be a single task, φ' that distinguishes the decisions. A related question then might be if this correction is necessary for the results from Part I. Empirically, running the experiments from Part I, with or without this correction made no qualitative difference in the results. Thus, in practice, it may make sense to not include the correction if it makes the implementation simpler.

Size of causal BDDs

Next, we return to the question of how big the compressed decision diagram can actually be. To this aim, we cite the following (conservative) bound on the size of an BDD given an encoding of the corresponding Boolean predicate in the linear model computation illustrated in Fig 9.4 (for more details, we refer the reader to [87]).

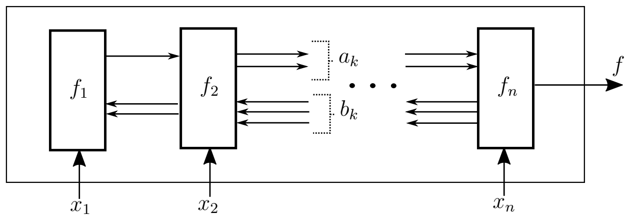


Figure 9.4: Generic network of Boolean modules for which Theorem 1 holds.

In particular, consider an arbitrary Boolean predicate

$$f : \{0,1\}^n \to \{0,1\} \tag{9.11}$$

and a sequential arrangement of n Boolean modules, $f_1, f_2, \ldots, f_n$ where each f_i has shape:

$$f_i : \{0,1\} \times \{0,1\}^{a_{i-1}} \times \{0,1\}^{b_i} \to \{0,1\}^{a_i} \times \{0,1\}^{b_{i-1}}, \tag{9.12}$$

and takes as input x_i as well as a_{i-1} outputs of its left neighbor and b_i outputs of the right neighbor ($b_0 = 0, a_n = 1$). Further, assume that this arrangement is well defined, e.g. for each assignment to $x_1, \ldots, x_n$ there exists a unique way to set each of the inter-module wires. We say these modules compute f if the final output is equal to $f(x_1, \ldots, x_n)$.

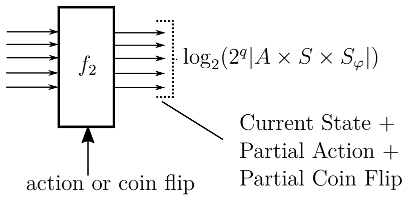

f_2
$\log_2(2^q|A \times S \times S_\varphi|)$
Current State + Partial Action + Partial Coin Flip
action or coin flip

Figure 9.5: Generic module in linear model of computation for $\mathcal{B}_f$. Note that backward edges are not required.

Theorem 1. If f can be computed by a linear arrangement of such modules, ordered $x_1, x_2, \ldots, x_n$, then the size, $S \in \mathbb{N}$, of $\mathcal{B}_f$ (in the same order), is upper bounded [21] by:

$$S \leq \sum_{k=1}^{n} 2^{a_k \cdot \left(2^{b_k}\right)}. \tag{9.13}$$

To apply this bound to our problem, recall that $\mathcal{B}_f$ computes a Boolean function where the decisions are temporally ordered and alternate between sequences of agent and environment decisions. Next, observe that because the traces are bounded (and all finite sets are regular), there exists a finite state machine which can monitor the satisfaction of the specification.

Remark 9.1.2. Ultimately, BDDs are a compression of the truth table of a Boolean predicate, which itself is simply a string of 2^n bits. As one might then expect, randomly selected predicates, i.e., a random sequence of 0's and 1's, typically have intractably large BDDs. From the theorem, we can view this as a consequence of modules necessarily needing back edges. Even without this theorem though, this result is inevitable since the set of small BDDs is much smaller than the number of all n-bit Boolean predicates (2^{2^n}). Furthermore, observe that decision problems and their negation (tautology detection) take constant time given a BDD, i.e., check if BDD has a single node and that node's label. Since these are NP-complete and co-NP-complete problems respectively, one sees that constructing the BDD can potentially take exponential time, even for predicates with small formula.

Specializing to SGs

Further, note that because this composed system is causal, no backward wires are needed, e.g., $\forall k \ . \ b_k = 0$. In particular, observe that because the composition of the dynamics and the monitor is Markovian, the entire system can be uniquely described using the monitor/dynamics state and agent/environment action (see Fig. 9.5). This description can be encoded in $\log_2(2^q|A \times S \times S_\varphi|)$ bits, where q denotes the number of coin flips tossed by the environment and S_φ denotes the monitor state. Therefore, a_k is upper bounded by $\log_2(2^q|A \times S \times S_\varphi|)$. Combined with (9.13) this results in the following bound on the size of $\mathcal{B}_f$.

Corollary 1. Let $M = (S, s_0, A, P)$ be a stochastic game whose probabilistic transitions can be approximated using q coin flips and let φ be a specification defined for horizon τ and monitored by a finite automaton with states S_φ. The corresponding BDD, $\mathcal{B}_{\widehat{\varphi}}$, has size bounded by:

$$|\mathcal{B}_{\widehat{\varphi}}| \leq \overbrace{\tau \cdot \Big(\log(|A|) + q\Big)}^{\#\text{ inputs}} \cdot \overbrace{\Big(2^q |A \times S \times S_\varphi|\Big)}^{\text{bound on } 2^{a_k}} \tag{9.14}$$

Notice that the above argument implies that as the episode length grows, $|\mathcal{B}_{\widehat{\varphi}}|$ grows linearly in the horizon/states and quasi-linearly in the agent/environment actions!

Remark 9.1.3. Note that this bound actually holds for the minimal representation of the composed dynamics/monitor (even if it's unknown a-priori!). For example, if the property is *true*, the BDD requires only one state (always evaluate true). This also illustrates that the above bound is often very conservative. In particular, note that for $\varphi = \textit{true}$, $|\mathcal{B}_{\widehat{\varphi}}| = 1$, independent of the horizon or dynamics. However, the above bound will always be linear in τ. In general, the size of the BDD will depend on the particular symmetries compressed.

Remark 9.1.4. With hindsight, corollary 1 is not too surprising. In particular, if the monitor is known, then one could explicitly compose the dynamics MDP with the monitor, with the resulting MDP having at most $|S \times S_\varphi|$ states. If one then includes the time step in the state, one could perform the soft-Bellman Backup directly on this automaton. In this composed automaton each (action, state) pair would need to be recorded. Thus, one would expect $O(|S \times S_\varphi \times A|)$ space to be used. In practice, this explicit representation is much bigger than $\mathcal{B}_{\widehat{\varphi}}$ due to the BDDs ability to skip over time steps and automatically compress symmetries.

Remark 9.1.5. In the worst case, the monitor could be the unrolled decision tree. This monitor would have exponential number of states. In practice, the composition of the dynamics and the monitor is expected to be much smaller.

Constructing causal BDDs

One of the biggest benefits of the BDD representation of a Boolean function is the ability to build BDDs from a Boolean combinations of other BDDs. Namely, given two BDDs with n and m nodes respectively, it is well known that the conjunction or disjunction of the BDDs has at most $n \cdot m$ nodes. Thus, in practice, if the combined BDD's remain relatively small, Boolean combinations remain efficient to compute and one does not construct the full binary decision tree! Further, note that BDDs support function composition. Namely, given predicates $f(x_1, \ldots, x_n)$ and n predicates $g_i(y_1, \ldots, y_k)$ the function

$$f\Bigg(g_1(y_1, \ldots, y_k), \ldots, g_n(y_1, \ldots, y_k)\Bigg) \tag{9.15}$$

can be computed in time [87]:

$$O(n \cdot |\mathcal{B}_f|^2 \cdot \max_i |\mathcal{B}_{g_i}|), \tag{9.16}$$

where $\mathcal{B}_f$ is the BDD for f and $\mathcal{B}_{g_i}$ are the BDDs for g_i. Now, suppose $\hat{\delta}_1, \ldots \hat{\delta}_{\log(|S|)}$ are Boolean predicates such that:

$$\hat{\delta}(\mathbf{s}, \mathbf{a}, \mathbf{c}) = (\hat{\delta}_1(\mathbf{s}, \mathbf{a}, \mathbf{c}), \ldots, \hat{\delta}_{\log(|S|)}(\mathbf{s}, \mathbf{a}, \mathbf{c})). \tag{9.17}$$

Thm 1 and an argument similar to that for Corr 1 imply then that constructing $\mathcal{B}_{\widehat{\varphi}}$, using repeated composition, takes time bounded by a low degree polynomial in $|A \times S \times S_\varphi|$ and the horizon. Moreover, the space complexity before and after composition are bounded by Corr 1.

Evaluating Demonstrations:

Next let us return to the question of how to evaluate the likelihood of a concrete demonstration in our compressed BDD. The key problem is that the BDD can only evaluate (binary) sequences of actions/coin flips, where as demonstrations are given as sequences of action/state pairs. That is, we need to algorithmically perform the following transformation.

$$s_0 \centerdot \mathbf{a}_0 \centerdot \mathbf{s}_1 \centerdot \ldots \centerdot \mathbf{a}_n \centerdot \mathbf{s}_{n+1} \mapsto \mathbf{a}_1 \centerdot \mathbf{c}_1 \centerdot \ldots \centerdot \mathbf{a}_n \centerdot \mathbf{c}_n \tag{9.18}$$

Given the random bit model assumption, this transformation can be rewritten as a series of Boolean Satisfiability problems:

$$\exists\ \mathbf{c}_i\ .\ \hat{\delta}(\mathbf{s}_i, \mathbf{a}_i, \mathbf{c}_i) = \mathbf{s}_{i+1} \tag{9.19}$$

While potentially intimidating, in practice such problems are quite simple for modern SAT solvers, particularly if the number of coin flips used is small. Furthermore, many systems are translation invariant. In such systems, the results of a single query (9.19), can be reused on other queries, e.g., in our gridworld example slipping can always be detected by examining if the agent moved in the direction they intended. Nevertheless, in general, if q coin flips are used, encoding all m demonstrations takes at most $O(m \cdot \tau \cdot 2^q)$, in the worst case.

Luce axiom relaxation with BDDs[†]

We close this section with a brief discussion on realizing the Luce axiom relaxation (Sec 5.6) using BDDs. The key observation is that if two actions (or states) are equivalent, then they must access the same nodes in a BDD. Thus, when the maximum causal entropy planner (4.6) is represented as a BDD, two actions are distinguished when the visit different nodes of the BDD. This has the (slight) advantage of being independent of the particular rationality coefficient used when planning.

9.2 Bibliographic Notes

Binary Decision Diagrams served as a key tool in the early days of model checking [38] and are frequently used in symbolic value iteration for Markov Decision Processes [74] and reachability analysis for probabilistic systems[88]. However, the literature has largely relied on Multi-Terminal BDDs to encode the transition probabilities for a **single** time step. In contrast, this work introduces a two-terminal (or three-terminal) encoding based on the finite unrolling of a probabilistic circuit. The key difference is the focus on symbolically representing paths rather than the set of reachable states. Historically, this latter perspective was taken within the model checking community - presumably due to being a simple extension of non-deterministic state reachability. The key difference with non-deterministic model checking is that state must be associated with the probability of reaching them. Representing all of these probabilities effectively shatters symmetries and reduces the efficacy of state-based perspective (compared to the non-deterministic setting).

Thus, in many settings, despite being a much larger set, it is often more compact the factorized representation of the set of paths. For example, after appearing in [153] (and independently in [75]), this encoding was later utilized to perform finite horizon model checking on Markov Chains, providing noticeable improvements compared to state-of-the-art model checkers on a number of standard benchmarks [76].

Finally, many of the results from this chapter can easily be adapted to the zoo of variations of Binary Decision Diagrams, e.g., Zero Suppressed Decision Diagrams [105], which employ different rules for compressing the truth table of a Boolean predicate. For an overview on these variants, we point the reader to [158].

Chapter 10

Final Words

I have discovered a truly remarkable proof of this theorem which this margin is too small to contain.

Pierre de Fermat (Mathematician, 1607-1665)

Motivated by the problem of learning and teaching tasks from demonstrations, this thesis contributed a collection of algorithms and theoretical machinery for systematically mitigating combinatorial explosions inherent in (1) finding specifications that explain an agent's behavior (2) finding pedagogic demonstrations that help humans infer the specification (3) robustly predicting the behavior of an agent adhering to a specification.

In the context of concept learning, specification mining, and grammatical inference, we saw that demonstrations provide an ergonomic and sample-efficient means to communicate formal languages and specifications. This dissertation enables a user to partially specify the desired behavior of a system as example demonstrations and then find an automata or program that explains the user's behavior.

In the context of inverse reinforcement learning, Boolean task specifications are a class of sparse memory augmented rewards with explicit support for temporal and Boolean composition. We argued that these properties make task specifications immune to certain classes of reward hacking bugs that emerge from ad-hoc composition or perturbations to the dynamics. Unfortunately, the discrete nature of task specifications combined with an a-priori ignorance of what historical features are needed to encode the demonstrated task make existing approaches to learning rewards from demonstrations inapplicable.

In either case, after adapting the theory of maximum causal entropy inverse reinforcement learning to address task specifications, the fundamental hurdle remaining was a way to efficiently search through countably infinite representation classes to find task specifications that explain (compress) a multi-set of demonstrations. To address this, Ch 5 provided the Demonstrations Informed Specification Search (DISS) algorithm. Leveraging DISS, we were able to efficiently learn tasks from demonstrations and adapt many of the ideas from the algorithmic teaching literature. For example, we showed in Ch 6 that synthesizing pedagogic demonstrations enabled communicating the nuances of a specification through demonstrations.

10.1 Future work

With an eye towards the future, we enumerate a number of directions that this dissertation left unexplored.

Hybrid and Continous Dynamics

The work on learning and teaching only assumed black-box access to the MaxEntPlanner. A natural question then is if DISS would work well in more complex domains, e.g. continuous dynamics and hybrid systems. This was indeed one of the main motivations for developing the local prefix-tree perspective in Ch 5. The key observation is that so long as a form of the Luce axiom can be made to hold (at least in the visited state / actions of the prefix-tree), then any entropy regularized planner [109] could be used for the Surprisal Guided Sampler. In particular, one might consider adapting the counterexample driven algorithm in Sec 5.6 to maintain approximate value distinguishability. That is actions (and states) are considered equivalent if their values never deviate by more than $\delta \in \mathbb{R}$ for any task. This would enable automatically adapting any (black-box) entropy regularized planner to work with DISS - even if the actions (and state space is continuous).

Approximate and Model-Free MaxEntPlanners

In many settings, one might not have a world model and may wish to approximate (or learn) the MaxEntPlanner, e.g. using a Neural Network [64] or with a variant of Monte Carlo Tree Search [63]. In either case, Ch 5 provides a good deal of evidence that DISS is not particularly sensitive to these changes. First and foremost, focusing on the prefix tree and pivoting made the algorithm agonistic to the size of the action and state space. Second, empirical evidence points that one should set SGS temperature, β, much smaller than one. The result is that one only needs to identify the pivot with the largest gradient (in absolute value). Empirically, we also saw that the (absolute value) in the pivot gradient was concentrated only on a few points. Since this gradient is a function of the difference in transition probabilities, this implies that, so long as the most surprising actions don't change, the pivot decision will remain unchanged when using approximate planners.

Hindsight Experience Replay

A natural follow up question then is how to create such approximate planners. At the time of this dissertation, a promising direction seems to be to leverage the extensive literature on model-free reinforcement learning (RL) with neural network policies [64, 57, 67]. Unfortunately, when learning from sparse rewards like task specifications, one is presented with a bootstrapping problem. It is difficult to receive any feedback since applying actions uniformly at random typically yields reward 0. To address this, recent works have proposed instead focusing on multi-task RL. Multi-task RL aims to learn a universal policy that takes as input the current task and state and yields a distribution over actions. The key insight, called Hindsight Experience Replay (HER) [11], is that even if a episode fails to perform the target task, it may perform a related task. Thus, by counter-factually relabeling the episode with an alternative task, one can receive non-zero reward. This enables learning how to perform common motion and state estimation primitives and has been shown to even outperform hand-crafted rewards. Furthermore, in the case of entropy regularized control, it has been shown that the optimal way to relabel tasks is through maximum entropy inverse reinforcement learning [51]. Thus the DISS algorithm presents an opportunity to realize HER on very large concept classes, e.g. DFAs.

Beyond DFAs

This leads us to the next unexplored direction: How do DISS and related algorithms empirically perform on different representation classes? For example, one might consider learning conjunctions of DFAs to capture the compositional nature of many tasks. One could try syntactic concept classes such as temporal logic fragments. One could move beyond regular languages and consider context free grammars or register automata. Or perhaps, one could try to learn over a large alphabet by learning invariants as decision trees or symbolic automata. Importantly, while DISS is formulated to be agnostic to the underlying representation class, increasing the expressiveness of the representation class necessarily increases the sample complexity [85] of the concept identifiers. As such, one may need stronger inductive biases (say from data derived priors or the structure of the concept class) or more iterations of DISS to achieve similar results. Nevertheless, such extensions would open the door to many more applications of this thesis.

Fixed Rewards and Learning Constraints

One of the main academic exercises of this dissertation was to see how far one could go without assuming the existence of a Markovian reward. That said, something not discussed in Ch 5 is that the Prop 5.3.2 holds even in the presence of a *fixed* Markovian reward. In particular, since the reward is fixed, it is independent of the pivot values. Thus, the gradient makes this term 0.

It may then be worth exploring applying the machinery from Part I to settings where the agent trades off satisfying the task specification and optimizing some objective. Furthermore, as alluded to in the bibliographic notes of Ch 4, a parallel literature [36, 128, 103] has emerged to study the limiting case where the agent is constrained to always satisfy the task specification while optimizing some objective, e.g., run as fast as possible while avoiding obstacles. Again, the key conceptual difference is that rather than changing the reward induced by the task specification, this literature (implicitly) changes the MDP to make violating the task specification impossible. Nevertheless, the monotonicity in the change in pivot value remains, e.g., removing a high-value path using a constraint lowers the corresponding pivot value. Using the above observations, one might consider using DISS as a means to explore combinatorially large constraint spaces.

From DISS to Markov Chain Monte Carlo (MCMC)

The learning problem (and the DISS algorithm) considered in this thesis were formulated as maximum a-posteriori (MAP) inference. However, MAP inferences fail to capture the entire distributional structure [14] - a problem exacerbated in our setting due to the discrete nature of representation classes. For example, it may be the case that two conflicting hypotheses explain the demonstrations locally, but differ in a unvisited parts of the workspace. To get a more global perspective, one may wish to marginalize over the entire belief distribution to answer queries such as: "Do the demonstrations imply that the agent should avoid red tiles?" Importantly, by itself this sub-task may not have much explanatory power, but answering this query may be safety critical, e.g. the red tiles are lava. To answer such queries, it may be interesting to investigate transforming DISS into a Monte Carlo Markov Chain (MCMC) method [104]. Such methods are related to simulated annealing, but provide a popular means to sample from the belief distribution - as opposed to finding the most probable elements. Several obstacles must be overcome to perform MCMC however. For example, in the Metropolis-Hastings algorithm [68], the canonical MCMC algorithm, requires the proposal distribution that has a non-zero (known) probability of undoing the previous proposal. This is decidedly not the case in DISS as aggregating constraints often makes the previous task candidate impossible to sample. Further, the SAT-based DFA identification algorithms make computing the transition probabilities of the proposal distribution non-trivial. As such, more investigation is required to adapt DISS to the MCMC setting.

Assume-Guarantee and Multi-Agent Planning

Finally, we close this dissertation with the observation that the Boolean nature of task specifications allows them to encode assume-guarantee contracts [30]. An interesting future application of DISS then would be to learn the implicit assumption of black-box components with known "guarantees", e.g., through the spec sheet.

An alternative direction is to use such contracts as a means to enable organic collaboration with other systems (and humans) on complex tasks. Viewed as a multi-player game, one might hope to automatically infer the *assumptions* each agent is making about the other agent's future behavior, as well as what *guarantees* each agent aims to provide under these assumptions.

Consider for example the collaborative cooking game Overcook, a benchmark for human-robot collaboration [25]. In Overcook, players work together to prepare meals. To meet deadlines, the players specialize, e.g., one player preps onions while another player cooks and serves them. For successful collaboration, it is imperative that that each player's specialization be compatible and complementary with other player's specializations. Particularly in high stakes domains such as industrial manufacturing, failure to operate properly in human-robot teams raises serious fiscal and safety concerns. Within this context, formal specifications, and particularly the well developed theory of contracts [112, 116, 125], seem to be promising means to represent such interactions. A key question then is how to adapt DISS and its agent model to the multi-agent setting.

10.2 Bibliographic Notes

Mirroring the introduction, below I would like to provide historical notes and attributions for the future directions discussed. I would like to thank Markus Rabe and Christian Szegedy for introducing me to the idea of hindsight experience replay. The potential connection of hindsight experience replay and maximum entropy inverse reinforcement learning is thanks to discussions with Benjamin Eysenbach. The problem of learning constraints was introduced to me by Dexter Scobee. The observation that DISS should apply to learning constraints derives from discussions with Kaylene Stocking and David McPherson. Regarding the adaptation of DISS for MCMC, this was the initial plan for DISS. Various directions and variants were discussed with Gil Lederman and Mark Ho before ultimately deciding to focus on simulated annealing. Finally, the ideas behind collaborative assume guarantee reasoning derive from working on the NSF VeHICaL project's hand-off challenge problem [113] and from conversations with Niklas Lauffer, Ameesh Shah, Yash Pant, Dexter Scobee, Bala Thoravi, and Inigo Incer. More generally this thesis can be seen as contributing to the VeHICal project's mission of "developing the foundations of verified co-design of interfaces and control for human cyber-physical systems (h-CPS) — cyber-physical systems that operate in concert with human operators." The theory and algorithms developed in this dissertation were steps towards providing a principled means for human and autonomous systems to communicate and interact in a manner that is auditable, formally verifiable, and ergonomic to the human.

Bibliography

[1] Pieter Abbeel and Andrew Y Ng. “Apprenticeship learning via inverse reinforcement learning”. In: *Proceedings of the twenty-first international conference on Machine learning*. ACM. 2004, p. 1.

[2] David Abel, Will Dabney, Anna Harutyunyan, Mark K. Ho, Michael L. Littman, Doina Precup, and Satinder Singh. “On the Expressivity of Markov Reward”. In: *NeurIPS*. 2021.

[3] Pieter W. Adriaans. “Learning as Data Compression”. In: *CiE*. Vol. 4497. Lecture Notes in Computer Science. Springer, 2007, pp. 11–24.

[4] Noa Agmon, Sarit Kraus, and Gal A. Kaminka. “Multi-robot perimeter patrol in adversarial settings”. In: *ICRA*. IEEE, 2008, pp. 2339–2345.

[5] Ilge Akkaya, Daniel J. Fremont, Rafael Valle, Alexandre Donzé, Edward A. Lee, and Sanjit A. Seshia. “Control Improvisation with Probabilistic Temporal Specifications”. In: *IoTDI*. IEEE Computer Society, 2016, pp. 187–198.

[6] Steve Alpern, Alec Morton, and Katerina Papadaki. “Patrolling Games”. In: *Oper. Res.* 59.5 (2011), pp. 1246–1257.

[7] Rajeev Alur, Rastislav Bodík, Garvit Juniwal, Milo M. K. Martin, Mukund Raghothaman, Sanjit A. Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. “Syntax-guided synthesis”. In: *FMCAD*. IEEE, 2013, pp. 1–8.

[8] Francesco Amigoni, Nicola Basilico, and Nicola Gatti. “Finding the optimal strategies for robotic patrolling with adversaries in topologically-represented environments”. In: *ICRA*. IEEE, 2009, pp. 819–824.

[9] Francesco Amigoni and Alessandro Gallo. “A Multi-Objective Exploration Strategy for Mobile Robots”. In: *ICRA*. IEEE, 2005, pp. 3850–3855.

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

[11] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. “Hindsight experience replay”. In: *Advances in neural information processing systems*. 2017, pp. 5048–5058.

[12] Dana Angluin. “Learning Regular Sets from Queries and Counterexamples”. In: *Inf. Comput.* 75.2 (1987), pp. 87–106. DOI: `10.1016/0890-5401(87)90052-6`. URL: `https://doi.org/10.1016/0890-5401(87)90052-6`.

[13] Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. “Algorithms and complexity results for #SAT and Bayesian inference”. In: *Foundations of computer science, 2003. proceedings. 44th annual ieee symposium on.* IEEE. 2003, pp. 340–351.

[14] Robert Bassett and Julio Deride. “Maximum a posteriori estimators as a limit of Bayes estimators”. In: *Math. Program.* 174.1-2 (2019), pp. 129–144.

[15] Mihir Bellare, Oded Goldreich, and Erez Petrank. “Uniform Generation of NP-Witnesses Using an NP-Oracle”. In: *Inf. Comput.* 163.2 (2000), pp. 510–526.

[16] Richard E Bellman et al. “Dynamic Programming, ser”. In: *Rand Corporation research study.* Princeton University Press, 1957.

[17] Armin Biere, Alessandro Cimatti, Edmund M Clarke, Ofer Strichman, Yunshan Zhu, et al. “Bounded model checking.” In: *Advances in computers* 58.11 (2003), pp. 117–148.

[18] Andreea Bobu, Dexter RR Scobee, Jaime F Fisac, S Shankar Sastry, and Anca D Dragan. “Less is more: Rethinking probabilistic models of human behavior”. In: *Proceedings of the 2020 ACM/IEEE International Conference on Human-Robot Interaction.* 2020, pp. 429–437.

[19] Tomás Brázdil, Václav Brozek, Krishnendu Chatterjee, Vojtech Forejt, and Antonín Kucera. “Two Views on Multiple Mean-Payoff Objectives in Markov Decision Processes”. In: *Log. Methods Comput. Sci.* 10.1 (2014).

[20] Tomás Brázdil, Krishnendu Chatterjee, Vojtech Forejt, and Antonín Kucera. “Trading performance for stability in Markov decision processes”. In: *J. Comput. Syst. Sci.* 84 (2017), pp. 144–170.

[21] Randal E Bryant. “Symbolic Boolean manipulation with ordered binary-decision diagrams”. In: *ACM Computing Surveys (CSUR)* 24.3 (1992), pp. 293–318.

[22] Randal E. Bryant. “Graph-Based Algorithms for Boolean Function Manipulation”. In: *IEEE Trans. Computers* 35.8 (1986), pp. 677–691.

[23] Randal E. Bryant. “Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams”. In: *ACM Comput. Surv.* 24.3 (1992), pp. 293–318.

[24] Estefany Carrillo. “Controller synthesis and Formal Behavior Inference in Autonomous Systems”. PhD thesis. University of Maryland, College Park, MD, USA, 2021.

[25] Micah Carroll, Rohin Shah, Mark K. Ho, Tom Griffiths, Sanjit A. Seshia, Pieter Abbeel, and Anca D. Dragan. “On the Utility of Learning about Humans for Human-AI Coordination”. In: *NeurIPS.* 2019, pp. 5175–5186.

[26] Supratik Chakraborty, Daniel J Fremont, Kuldeep S Meel, Sanjit A Seshia, and Moshe Y Vardi. “Distribution-aware sampling and weighted model counting for SAT”. In: *Twenty-Eighth AAAI Conference on Artificial Intelligence.* 2014.

[27] Supratik Chakraborty, Dror Fried, Kuldeep S Meel, and Moshe Y Vardi. "From Weighted to Unweighted Model Counting". In: *Proceedings of IJCAI*. 2015, pp. 689–695.

[28] Supratik Chakraborty, Kuldeep S Meel, and Moshe Y Vardi. "Algorithmic improvements in approximate counting for probabilistic inference: From linear to logarithmic SAT calls". In: *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI-16)* (2016).

[29] Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. "Balancing Scalability and Uniformity in SAT Witness Generator". In: *DAC*. ACM, 2014, 60:1–60:6.

[30] Krishnendu Chatterjee and Thomas A. Henzinger. "Assume-Guarantee Synthesis". In: *TACAS*. Vol. 4424. Lecture Notes in Computer Science. Springer, 2007, pp. 261–275.

[31] Krishnendu Chatterjee, Joost-Pieter Katoen, Maximilian Weininger, and Tobias Winkler. "Stochastic Games with Lexicographic Reachability-Safety Objectives". In: *CAV (2)*. Vol. 12225. LNCS. Springer, 2020, pp. 398–420.

[32] Krishnendu Chatterjee, Rupak Majumdar, and Thomas A. Henzinger. "Markov Decision Processes with Multiple Objectives". In: *STACS*. Vol. 3884. LNCS. Springer, 2006, pp. 325–336.

[33] Mark Chavira and Adnan Darwiche. "On probabilistic inference by weighted model counting". In: *Artificial Intelligence* 172.6-7 (2008), pp. 772–799.

[34] Taolue Chen, Vojtech Forejt, Marta Z. Kwiatkowska, Aistis Simaitis, and Clemens Wiltsche. "On Stochastic Games with Multiple Objectives". In: *MFCS*. Vol. 8087. LNCS. Springer, 2013, pp. 266–277.

[35] Arthur Choi and Adnan Darwiche. "Dynamic minimization of sentential decision diagrams". In: *Twenty-Seventh AAAI Conference on Artificial Intelligence*. 2013.

[36] Glen Chou, Dmitry Berenson, and Necmiye Ozay. "Learning Constraints from Demonstrations". In: *WAFR*. Vol. 14. Springer Proceedings in Advanced Robotics. Springer, 2018, pp. 228–245.

[37] Glen Chou, Necmiye Ozay, and Dmitry Berenson. "Explaining Multi-stage Tasks by Learning Temporal Logic Formulas from Suboptimal Demonstrations". In: *Robotics: Science and Systems*. 2020.

[38] Edmund M. Clarke, Orna Grumberg, Daniel Kroening, Doron A. Peled, and Helmut Veith. *Model checking, 2nd Edition*. MIT Press, 2018.

[39] Anne Condon. "On Algorithms for Simple Stochastic Games". In: *Advances In Computational Complexity Theory*. Vol. 13. DIMACS Series in Discrete Mathematics and Theoretical Computer Science. DIMACS/AMS, 1990, pp. 51–71.

[40] Richard H. Connelly and F. Lockwood Morris. "A Generalization of the Trie Data Structure". In: *Math. Struct. Comput. Sci.* 5.3 (1995), pp. 381–418.

[41] François Coste and Jacques Nicolas. “Regular inference as a graph coloring problem”. In: *In Workshop on Grammatical Inference, Automata Induction, and Language Acquisition (ICML’97*. Citeseer. 1997.

[42] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[43] Nilesh Dalvi and Dan Suciu. “Efficient query evaluation on probabilistic databases”. In: *The VLDB Journal—The International Journal on Very Large Data Bases* 16.4 (2007), pp. 523–544.

[44] Martin Davis and Hilary Putnam. “A Computing Procedure for Quantification Theory”. In: *J. ACM* 7.3 (1960), pp. 201–215.

[45] Colin De la Higuera. *Grammatical inference: learning automata and grammars*. Cambridge University Press, 2010.

[46] Florent Delgrange, Joost-Pieter Katoen, Tim Quatmann, and Mickael Randour. “Simple Strategies in Multi-Objective MDPs”. In: *TACAS (1)*. Vol. 12078. LNCS. Springer, 2020, pp. 346–364.

[47] François Denis. “Learning Regular Languages from Simple Positive Examples”. In: *Mach. Learn.* 44.1/2 (2001), pp. 37–66.

[48] Anca D. Dragan, Kenton C. T. Lee, and Siddhartha S. Srinivasa. “Legibility and predictability of robot motion”. In: *HRI*. IEEE/ACM, 2013, pp. 301–308.

[49] Paul Ehrenfest. “Welche Züge der Lichtquantenhypothese spielen in der Theorie der Wärmestrahlung eine wesentliche Rolle?” In: *Annalen der Physik* (1911).

[50] Kousha Etessami, Marta Z. Kwiatkowska, Moshe Y. Vardi, and Mihalis Yannakakis. “Multi-objective Model Checking of Markov Decision Processes”. In: *TACAS*. Vol. 4424. LNCS. Springer, 2007, pp. 50–65.

[51] Ben Eysenbach, Xinyang Geng, Sergey Levine, and Ruslan Salakhutdinov. “Rewriting History with Inverse RL: Hindsight Inference for Policy Improvement”. In: *NeurIPS*. 2020.

[52] Benjamin Eysenbach and Sergey Levine. “Maximum Entropy RL (Provably) Solves Some Robust RL Problems”. In: *CoRR* abs/2103.06257 (2021).

[53] Vojtech Forejt, Marta Z. Kwiatkowska, and David Parker. “Pareto Curves for Probabilistic Model Checking”. In: *ATVA*. Vol. 7561. LNCS. Springer, 2012, pp. 317–332.

[54] Daniel J. Fremont, Alexandre Donzé, Sanjit A. Seshia, and David Wessel. “Control Improvisation”. In: *FSTTCS*. Vol. 45. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015, pp. 463–474.

[55] Daniel J. Fremont and Sanjit A. Seshia. “Reactive Control Improvisation”. In: *CAV (1)*. Vol. 10981. LNCS. Springer, 2018, pp. 307–326.

[56] Jin I. Ge and Richard M. Murray. “Voluntary lane-change policy synthesis with control improvisation”. In: *CDC*. IEEE, 2018, pp. 3640–3647.

[57] Matthieu Geist, Bruno Scherrer, and Olivier Pietquin. “A Theory of Regularized Markov Decision Processes”. In: *ICML*. Vol. 97. PMLR. PMLR, 2019, pp. 2160–2169.

[58] Shalini Ghosh, Susmit Jha, Ashish Tiwari, Patrick Lincoln, and Xiaojin Zhu. “Model, Data and Reward Repair: Trusted Machine Learning for Markov Decision Processes”. In: *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE. 2018, pp. 194–199.

[59] E. Mark Gold. “Complexity of Automaton Identification from Given Data”. In: *Inf. Control.* 37.3 (1978), pp. 302–320.

[60] E. Mark Gold. “Language Identification in the Limit”. In: *Inf. Control.* 10.5 (1967), pp. 447–474.

[61] Noah D Goodman and Michael C Frank. “Pragmatic language interpretation as probabilistic inference”. In: *Trends in cognitive sciences* 20.11 (2016), pp. 818–829.

[62] Herbert P Grice. “Logic and conversation”. In: *Speech acts*. Brill, 1975, pp. 41–58.

[63] Jean-Bastien Grill, Omar Darwiche Domingues, Pierre Ménard, Rémi Munos, and Michal Valko. “Planning in entropy-regularized Markov decision processes and games”. In: *NeurIPS*. 2019, pp. 12383–12392.

[64] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor”. In: *ICML*. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 1856–1865.

[65] Sofie Haesaert, Petter Nilsson, and Sadegh Soudjani. “Formal Multi-Objective Synthesis of Continuous-State MDPs”. In: *IEEE Control. Syst. Lett.* 5.5 (2021), pp. 1765–1770.

[66] Arnd Hartmanns, Sebastian Junges, Joost-Pieter Katoen, and Tim Quatmann. “Multi-cost Bounded Tradeoff Analysis in MDP”. In: *J. Autom. Reason.* 64.7 (2020), pp. 1483–1522.

[67] Hado van Hasselt. “Double Q-learning”. In: *NIPS*. Curran Associates, Inc., 2010, pp. 2613–2621.

[68] W Keith Hastings. “Monte Carlo sampling methods using Markov chains and their applications”. In: (1970).

[69] Holger Hermanns, Jan Krcál, and Gilles Nies. “Recharging Probably Keeps Batteries Alive”. In: *CyPhy*. Vol. 9361. LNCS. Springer, 2015, pp. 83–98.

[70] Marijn Heule and Sicco Verwer. “Exact DFA Identification Using SAT Solvers”. In: *ICGI*. Vol. 6339. Lecture Notes in Computer Science. Springer, 2010, pp. 66–79.

[71] Timothy J. Hickey and Jacques Cohen. “Uniform Random Generation of Strings in a Context-Free Language”. In: *SIAM J. Comput.* 12.4 (1983), pp. 645–655.

[72] Mark K. Ho, Michael L. Littman, Fiery Cushman, and Joseph L. Austerweil. "Teaching with Rewards and Punishments: Reinforcement or Communication?" In: *Proceedings of the 37th Annual Conference of the Cognitive Science Society*. Ed. by D.C. Noelle, R. Dale, A. S. Warlaumont, J. Yoshimi, T. Matlock, C. D. Jennings, and P. P. Maglio. Austin, TX: Cognitive Science Society, 2015, pp. 920–925.

[73] Mark K. Ho, Michael L. Littman, James MacGlashan, Fiery Cushman, and Joseph L. Austerweil. "Showing versus doing: Teaching by demonstration". In: *NIPS*. 2016, pp. 3027–3035.

[74] Jesse Hoey, Robert St-Aubin, Alan Hu, and Craig Boutilier. "SPUDD: Stochastic planning using decision diagrams". In: *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc. 1999, pp. 279–288.

[75] Steven Holtzen, Guy Van den Broeck, and Todd D. Millstein. "Scaling exact inference for discrete probabilistic programs". In: *Proc. ACM Program. Lang.* 4.OOPSLA (2020), 140:1–140:31.

[76] Steven Holtzen, Sebastian Junges, Marcell Vazquez-Chanlatte, Todd D. Millstein, Sanjit A. Seshia, and Guy Van den Broeck. "Model Checking Finite-Horizon Markov Chains with Probabilistic Inference". In: *CAV (2)*. Vol. 12760. Lecture Notes in Computer Science. Springer, 2021, pp. 577–601.

[77] Matanya B. Horowitz, Eric M. Wolff, and Richard M. Murray. "A compositional approach to stochastic optimal control with co-safe temporal logic specifications". In: *IROS*. IEEE, 2014, pp. 1466–1473.

[78] Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Anthony Valenzano, and Sheila A. McIlraith. "Using Reward Machines for High-Level Task Specification and Decomposition in Reinforcement Learning". In: *ICML*. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 2112–2121.

[79] Edwin T Jaynes. "Information theory and statistical mechanics". In: *Physical review* 106.4 (1957), p. 620.

[80] Mark Jerrum, Leslie G. Valiant, and Vijay V. Vazirani. "Random Generation of Combinatorial Structures from a Uniform Distribution". In: *Theor. Comput. Sci.* 43 (1986), pp. 169–188.

[81] Susmit Jha and Sanjit A. Seshia. "A theory of formal synthesis via inductive learning". In: *Acta Informatica* 54.7 (2017), pp. 693–726.

[82] Rudolf Emil Kalman. "When is a linear control system optimal". In: (1964).

[83] Sampath Kannan, Z. Sweedyk, and Stephen R. Mahaney. "Counting and Random Generation of Strings in Regular Languages". In: *SODA*. ACM/SIAM, 1995, pp. 551–557.

[84] Daniel Kasenberg and Matthias Scheutz. "Interpretable apprenticeship learning with temporal logic specifications". In: *CDC*. IEEE, 2017, pp. 4914–4921.

[85] Michael J Kearns and Umesh Virkumar Vazirani. *An Introduction to Computational Learning Theory.* MIT press, 1994.

[86] David Klaska, Antonín Kucera, and Vojtech Rehák. "Adversarial Patrolling with Drones". In: *AAMAS*. IFAAMAS, 2020, pp. 629–637.

[87] Donald Ervin Knuth. *The Art of Computer Programming: Vol. 4, No. 1: Bitwise Tricks and Techniques-Binary Decision Diagrams.* Addison Wesley Professional, 2009.

[88] M. Kwiatkowska, G. Norman, and D. Parker. "PRISM 4.0: Verification of Probabilistic Real-time Systems". In: *Proc. 23rd International Conference on Computer Aided Verification (CAV'11).* Ed. by G. Gopalakrishnan and S. Qadeer. Vol. 6806. LNCS. Springer, 2011, pp. 585–591.

[89] Marta Kwiatkowska, David Parker, and Clemens Wiltsche. "PRISM-games: verification and strategy synthesis for stochastic multi-player games with multiple objectives". In: *Int. J. Softw. Tools Technol. Transf.* 20.2 (2018), pp. 195–210.

[90] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. "Probabilistic Symbolic Model Checking with PRISM: A Hybrid Approach". In: *TACAS*. Vol. 2280. Lecture Notes in Computer Science. Springer, 2002, pp. 52–66.

[91] Bruno Lacerda, Fatma Faruq, David Parker, and Nick Hawes. "Probabilistic planning with formal performance guarantees for mobile service robots". In: *Int. J. Robotics Res.* 38.9 (2019).

[92] Kevin J. Lang, Barak A. Pearlmutter, and Rodney A. Price. "Results of the Abbadingo One DFA Learning Competition and a New Evidence-Driven State Merging Algorithm". In: *ICGI*. Vol. 1433. Lecture Notes in Computer Science. Springer, 1998, pp. 1–12.

[93] Edward Ashford Lee and Sanjit Arunkumar Seshia. *Introduction to embedded systems: A cyber-physical systems approach.* Mit Press, 2016.

[94] Jan Leike, Miljan Martic, Victoria Krakovna, Pedro A. Ortega, Tom Everitt, Andrew Lefrancq, Laurent Orseau, and Shane Legg. "AI Safety Gridworlds". In: *CoRR* abs/1711.09883 (2017).

[95] Claude Lemaréchal. "S. Boyd, L. Vandenberghe, Convex Optimization, Cambridge University Press, 2004 hardback, ISBN 0 521 83378 7". In: *Eur. J. Oper. Res.* 170.1 (2006), pp. 326–327.

[96] Sergey Levine. "Reinforcement Learning and Control as Probabilistic Inference: Tutorial and Review". In: *CoRR* abs/1805.00909 (2018). arXiv: `1805.00909`. URL: `http://arxiv.org/abs/1805.00909`.

[97] Wenchao Li. "Specification Mining: New Formalisms, Algorithms and Applications". PhD thesis. EECS Department, University of California, Berkeley, Mar. 2014. URL: `http://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-20.html`.

[98] Michael L. Littman, Ufuk Topcu, Jie Fu, Charles Isbell, Min Wen, and James MacGlashan. "Environment-Independent Task Specifications via GLTL". In: *arXiv:1704.04341 [cs]* (Apr. 2017). arXiv: 1704.04341. URL: `http://arxiv.org/abs/1704.04341`.

[99] R Duncan Luce. "Individual choice behavior." In: (1959).

[100] J Lumbroso. *Optimal discrete uniform generation from coin flips, and applications. CoRR abs/1304.1916 (2013).* 2013.

[101] Sharad Malik and Lintao Zhang. "Boolean satisfiability from theoretical hardness to practical success". In: *Commun. ACM* 52.8 (2009), pp. 76–82.

[102] Miguel A. Martínez-Prieto, Nieves R. Brisaboa, Rodrigo Cánovas, Francisco Claude, and Gonzalo Navarro. "Practical compressed string dictionaries". In: *Inf. Syst.* 56 (2016), pp. 73–108.

[103] David Livingston McPherson, Kaylene C. Stocking, and S. Shankar Sastry. "Maximum Likelihood Constraint Inference from Stochastic Demonstrations". In: *CCTA*. IEEE, 2021, pp. 1208–1213.

[104] Nicholas Metropolis and Stanislaw Ulam. "The monte carlo method". In: *Journal of the American statistical association* (1949).

[105] Alan Mishchenko. "An introduction to zero-suppressed binary decision diagrams". In: *Proceedings of the 12th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning*. Vol. 8. Citeseer. 2001, pp. 1–15.

[106] Sriraam Natarajan and Prasad Tadepalli. "Dynamic preferences in multi-criteria reinforcement learning". In: *ICML*. Vol. 119. ACM International Conference Proceeding Series. ACM, 2005, pp. 601–608.

[107] Milad Nazarahari, Esmaeel Khanmirza, and Samira Doostie. "Multi-objective multi-robot path planning in continuous environment using an enhanced genetic algorithm". In: *Expert Syst. Appl.* 115 (2019), pp. 106–120.

[108] Daniel Neider and Ivan Gavran. "Learning Linear Temporal Properties". In: *FMCAD*. IEEE, 2018, pp. 1–10.

[109] Gergely Neu, Anders Jonsson, and Vicenç Gómez. "A unified view of entropy-regularized Markov decision processes". In: *CoRR* abs/1705.07798 (2017).

[110] John von Neumann. "Various techniques used in connection with random digits". In: *John von Neumann, Collected Works* 5 (1963), pp. 768–770.

[111] Andrew Y. Ng and Stuart J. Russell. "Algorithms for Inverse Reinforcement Learning". In: *ICML*. Morgan Kaufmann, 2000, pp. 663–670.

[112] Pierluigi Nuzzo, Antonio Iannopollo, Stavros Tripakis, and Alberto L. Sangiovanni-Vincentelli. "Are interface theories equivalent to contract theories?" In: *MEMOCODE*. IEEE, 2014, pp. 104–113.

[113] Yash Vardhan Pant, Balasaravanan Thoravi Kumaravel, Ameesh Shah, Erin Kraemer, Marcell Vazquez-Chanlatte, K Kulkarni, Bjoern Hartmann, and Sanjit A Seshia. *Model-based Formalization of the Autonomy-to-Human Perception Hand-off*. Tech. rep. Technical Report UCB/EECS-2021-8, EECS Department, UC Berkeley, 2021.

[114] Simone Parisi, Matteo Pirotta, Nicola Smacchia, Luca Bascetta, and Marcello Restelli. "Policy gradient approaches for multi-objective sequential decision making: A comparison". In: *ADPRL*. IEEE, 2014, pp. 1–8.

[115] Praveen Paruchuri, Jonathan P. Pearce, Milind Tambe, Fernando Ordóñez, and Sarit Kraus. "An efficient heuristic approach for security against multiple adversaries". In: *AAMAS*. IFAAMAS, 2007, p. 181.

[116] Roberto Passerone, Íñigo Íncer Romeo, and Alberto L. Sangiovanni-Vincentelli. "Coherent Extension, Composition, and Merging Operators in Contract Models for System Design". In: *ACM Trans. Embed. Comput. Syst.* 18.5s (2019), 86:1–86:23.

[117] Joelle Pineau, Geoff Gordon, Sebastian Thrun, et al. "Point-based value iteration: An anytime algorithm for POMDPs". In: *IJCAI*. Vol. 3. 2003, pp. 1025–1032.

[118] David Portugal, Charles Pippin, Rui P. Rocha, and Henrik I. Christensen. "Finding optimal routes for multi-robot patrolling in generic graphs". In: *IROS*. IEEE, 2014, pp. 363–369.

[119] Prolific. *Prolific*. `https://www.prolific.co`. 2022.

[120] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. Wiley, 1994.

[121] Tim Quatmann, Sebastian Junges, and Joost-Pieter Katoen. "Markov Automata with Multiple Objectives". In: *CAV (1)*. Vol. 10426. LNCS. Springer, 2017, pp. 140–159.

[122] Markus N Rabe, Christoph M Wintersteiger, Hillel Kugler, Boyan Yordanov, and Youssef Hamadi. "Symbolic approximation of the bounded reachability probability in large Markov chains". In: *Proceedings of QEST*. Springer. 2014, pp. 388–403.

[123] Deepak Ramachandran and Eyal Amir. "Bayesian inverse reinforcement learning". In: *IJCAI* (2007).

[124] Mickael Randour, Jean-François Raskin, and Ocan Sankur. "Percentile queries in multi-dimensional Markov decision processes". In: *Formal Methods Syst. Des.* 50.2-3 (2017), pp. 207–248.

[125] Íñigo Íncer Romeo, Alberto L. Sangiovanni-Vincentelli, Chung-Wei Lin, and Eunsuk Kang. "Quotient for Assume-Guarantee Contracts". In: *MEMOCODE*. IEEE, 2018, pp. 67–77.

[126] Tian Sang, Paul Beame, and Henry A Kautz. "Performing Bayesian inference by weighted model counting". In: *AAAI*. Vol. 5. 2005, pp. 475–481.

[127] Yagiz Savas, Melkior Ornik, Murat Cubuktepe, Mustafa O. Karabag, and Ufuk Topcu. “Entropy Maximization for Markov Decision Processes Under Temporal Logic Constraints”. In: *IEEE Trans. Autom. Control.* 65.4 (2020), pp. 1552–1567.

[128] Dexter R. R. Scobee and S. Shankar Sastry. “Maximum Likelihood Constraint Inference for Inverse Reinforcement Learning”. In: *ICLR*. OpenReview.net, 2020.

[129] Sanjit A. Seshia. “Combining Induction, Deduction, and Structure for Verification and Synthesis”. In: *Proc. IEEE* 103.11 (2015), pp. 2036–2051.

[130] Sanjit A. Seshia, Dorsa Sadigh, and S. Shankar Sastry. “Towards Verified Artificial Intelligence”. In: *ArXiv e-prints* (2016). arXiv: `1606.08514`.

[131] Ankit Shah, Pritish Kamath, Julie A. Shah, and Shen Li. “Bayesian Inference of Temporal Task Specifications from Demonstrations”. In: *NeurIPS*. 2018, pp. 3808–3817.

[132] Marwaan Simaan and Jose B Cruz. “On the Stackelberg strategy in nonzero-sum games”. In: *Journal of Optimization Theory and Applications* 11.5 (1973), pp. 533–555.

[133] Michael Sipser. “Introduction to the Theory of Computation”. In: *SIGACT News* 27.1 (1996), pp. 27–29.

[134] Christopher C. Skiscim and Bruce L. Golden. “Optimization by simulated annealing: A preliminary computational study for the TSP”. In: *WSC*. ACM, 1983, pp. 523–535.

[135] Armando Solar-Lezama, Liviu Tancau, Rastislav Bodík, Sanjit A. Seshia, and Vijay A. Saraswat. “Combinatorial sketching for finite programs”. In: *ASPLOS*. ACM, 2006, pp. 404–415.

[136] Ray J. Solomonoff. “A Formal Theory of Inductive Inference. Part I”. In: *Inf. Control.* 7.1 (1964), pp. 1–22.

[137] Patrick Speicher, Marcel Steinmetz, Michael Backes, Jörg Hoffmann, and Robert Künnemann. “Stackelberg Planning: Towards Effective Leader-Follower State Space Search”. In: *AAAI*. AAAI Press, 2018, pp. 6286–6293.

[138] Daniel A. Spielman and Shang-Hua Teng. “Smoothed analysis: an attempt to explain the behavior of algorithms in practice”. In: *Commun. ACM* 52.10 (2009), pp. 76–84.

[139] Arvind Srinivasan, Timothy Kam, Sharad Malik, and Robert K. Brayton. “Algorithms for Discrete Function Manipulation”. In: *ICCAD*. IEEE Computer Society, 1990, pp. 92–95.

[140] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction.* Adaptive computation and machine learning. MIT Press, 1998.

[141] Milind Tambe. *Security and Game Theory - Algorithms, Deployed Systems, Lessons Learned.* Cambridge University Press, 2012.

[142] Richard Taylor. “Purposeful and non-purposeful behavior: A rejoinder”. In: *Philosophy of Science* (1950).

[143] Joshua Brett Tenenbaum. “A Bayesian framework for concept learning”. PhD thesis. Massachusetts Institute of Technology, 1999.

[144] Hazem Torfah, Shetal Shah, Supratik Chakraborty, S. Akshay, and Sanjit A. Seshia. “Synthesizing Pareto-Optimal Interpretations for Black-Box Models”. In: *FMCAD*. IEEE, 2021, pp. 153–162.

[145] Dogan Ulus. “Sequential Circuits from Regular Expressions Revisited”. In: *CoRR* abs/1801.08979 (2018).

[146] Vladimir Ulyantsev, Ilya Zakirzyanov, and Anatoly Shalyto. “BFS-Based Symmetry Breaking Predicates for DFA Identification”. In: *LATA*. Vol. 8977. Lecture Notes in Computer Science. Springer, 2015, pp. 611–622.

[147] Marcell Vazquez-Chanlatte. *Improvisers: A Python library for synthesizing Entropic Reactive Control Improvisers for stochastic games.* 2021. URL: `https://github.com/mvcisback/improvisers`.

[148] Marcell Vazquez-Chanlatte, Mark K Ho, Thomas L Griffiths, and Sanjit A Seshia. “Communicating Compositional and Temporal Specifications by Demonstrations, Extended Abstract”. In: *Symposium on Cyber-Physical Human Systems (CPHS)*. 2018.

[149] Marcell Vazquez-Chanlatte, Susmit Jha, Ashish Tiwari, Mark K. Ho, and Sanjit A. Seshia. “Learning Task Specifications from Demonstrations”. In: *NeurIPS*. 2018, pp. 5372–5382.

[150] Marcell Vazquez-Chanlatte, Sebastian Junges, Daniel J. Fremont, and Sanjit Seshia. “Entropy-Guided Control Improvisation”. In: *Robotics: Science and Systems*. 2021.

[151] Marcell Vazquez-Chanlatte, Markus N Rabe, and Sanjit A Seshia. “A Model Counter’s Guide to Probabilistic Systems”. In: *arXiv preprint arXiv:1903.09354* (2019).

[152] Marcell Vazquez-Chanlatte, Markus N. Rabe, and Sanjit A. Seshia. “A Model Counter’s Guide to Probabilistic Systems”. In: *CoRR* abs/1903.09354 (2019).

[153] Marcell Vazquez-Chanlatte and Sanjit A. Seshia. “Maximum Causal Entropy Specification Inference from Demonstrations”. In: *CAV (2)*. Vol. 12225. LNCS. Springer, 2020, pp. 255–278.

[154] Marcell Vazquez-Chanlatte and Ameesh Shah. *Demonstration Informed Specification Search.* Version 0.2.10. URL: `https://github.com/mvcisback/DISS`.

[155] Marcell Vazquez-Chanlatte, Ameesh Shah, Gil Lederman, and Sanjit A. Seshia. “Demonstration Informed Specification Search”. In: *CoRR* abs/2112.10807 (2021).

[156] Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. “Programmatically Interpretable Reinforcement Learning”. In: *ICML*. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 5052–5061.

[157] Kandai Watanabe, Nicholas Renninger, Sriram Sankaranarayanan, and Morteza Lahijanian. “Probabilistic Specification Learning for Planning with Safety Constraints”. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 6558–6565.

[158] Ingo Wegener. *Branching Programs and Binary Decision Diagrams*. SIAM, 2000.

[159] Jie Xu, Yunsheng Tian, Pingchuan Ma, Daniela Rus, Shinjiro Sueda, and Wojciech Matusik. “Prediction-Guided Multi-Objective Reinforcement Learning for Continuous Robot Control”. In: *ICML*. Vol. 119. PMLR. PMLR, 2020, pp. 10607–10616.

[160] Yuan Yang and Steven T Piantadosi. “One model for the learning of language”. In: *Proceedings of the National Academy of Sciences* 119.5 (2022).

[161] Hansol Yoon and Sriram Sankaranarayanan. “Predictive Runtime Monitoring for Mobile Robots using Logic-Based Bayesian Intent Inference”. In: *ICRA*. IEEE, 2021, pp. 8565–8571.

[162] Brian D Ziebart. “Modeling purposeful adaptive behavior with the principle of maximum causal entropy”. PhD thesis. CMU, 2010.

[163] Brian D Ziebart, J Andrew Bagnell, and Anind K Dey. “Modeling interaction via the principle of maximum causal entropy”. In: (2010).

[164] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. “Maximum Entropy Inverse Reinforcement Learning.” In: *AAAI*. Vol. 8. Chicago, IL, USA. 2008, pp. 1433–1438.