Computationally Bounded Rationality From Three Perspectives: Precomputation, Regret Tradeoffs, And Lifelong Learning

Thomas Orton

St Hugh's College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2023

Abstract

What does it mean for a computer program to be optimal? Many fields in optimal decision making, from game theory to Bayesian decision theory, define optimal solutions which can be computationally intractable to implement or find. This is problematic, because it means that sometimes these solutions are not physically realizable. To address this problem, bounded rationality studies what it means to behave optimally subject to constraints on processing time, memory and knowledge. This thesis contributes three new models for studying bounded rationality in different contexts.

The first model considers games like chess. We suppose each player can spend some time before the game precomputing (memorizing) strong moves from an oracle, but has limited memory to remember these moves. We show how to analytically quantify how randomly optimal strategies play in equilibrium, and give polynomial-time algorithms for computing a best response and an ε-Nash equilibrium. We use the best response algorithm to empirically evaluate the chess playing program Stockfish.

The second model takes place in the setting of adversarial online learning. Here, we imagine an algorithm receives new problems online, and is given a computational budget to run B problem solvers for each problem. We show how to trade off the budget B for a strengthening of the algorithm's regret guarantee in both the full and semi-bandit feedback settings. We then show how this tradeoff implies new results for Online Submodular Function Maximization (OSFM) (Streeter and Golovin, 2008) and Linear Programming. We use these observations to derive and benchmark a new algorithm for OSFM.

The third model approaches bounded rationality from the perspective of lifelong learning (Chen and Liu, 2018). Instead of modelling the final solution, lifelong learning models how a computationally bounded agent can accumulate knowledge over time and attempt to solve tractable subproblems it encounters. We develop models for incrementally accumulating and learning knowledge in a domain agnostic setting, and use these models to give an abstract framework for a lifelong reinforcement learner. The framework attempts to make a step towards making the best of analytical performance guarantees, while still being able to make use of black box techniques such as neural networks which may perform well in practice.

Acknowledgements

Thank you Varun, Rahul, and Standa. You gave me the freedom to explore my own research topics. You were consistently supportive, positive, and patient as I learned to become a better researcher during the past 4 years.

Varun: I still remember the time I was a visiting student at Oxford, taking your computational learning theory course while being tutored by Francisco. At the time, I certainly hadn't anticipated I would come back for another 4 years. Thank you for bringing me back to Oxford, for being a mentor on how to do academic research, and for being a good friend throughout. Thank you for the memories of high table and several years of interviewing candidates; I think these will be with me for a long time. Thank you for encouraging me to apply for the FHI scholarship; sometimes the small acts of help you offered ended up being exceptionally valuable in hindsight.

Rahul: Thank you for sharing your wealth of knowledge in TOC. I always enjoy your perspectives on the problems we discuss, and the way you are able to relate concepts which come up to work done by others. Thank you for always being willing to offer help whenever it was needed. Thank you for encouraging me to explore my own research interests, and for giving candid feedback when asked.

Standa: Thank you for consistently making the effort to have supervisor meetings, even during the times when the problems being discussed were not directly related to your research, and even during the times when you were exceptionally busy. You were always especially reliable with responding quickly to emails, and expressing a general degree of confidence in whichever research problem was being worked on. Thank you for your help in developing my knowledge as a researcher: whether that be the gradually assimilated knowledge of how to approach a research problem, the exposure to domain knowledge like CSPs and polymorphisms, or even the applied knowledge of department politics.

Michael Wooldridge and Shimon Whiteson: Thank you for assessing my transfer and confirmation of status, and for the helpful literature and research direction ideas you shared.

Boaz Barak: Thank you for your encouragement and mentorship in TCS research during my undergraduate degree at Harvard. We first met after we noticed a bug in a proof during a CS127 lecture. True to your enthusiasm to teach, you then spent 45 minutes with me after class until we figured out a fix on one of the Maxwell Dworkin whiteboards. Over the next few years, you helped me present at graduate student research groups, take research classes, encouraged me to apply for a PhD, and helped me with applications. It's quite possible this PhD wouldn't have happened without

your help; thank you for your support!

Marcus Bizony, Vernon Wood, and my teachers who supported me during high school: Thank you for tutoring me in Further Maths (Marcus), allowing me to enter dual-enrollment (Vernon), and being so accommodating and open minded with my last years of high school (John, Sean, Willem, Graham, and the list goes on). Without your support, I may not have had the same academic opportunities later on in life; therefore, you played a role in helping me to write this thesis too. Thank you.

Damon: Thank you for a very fun and fruitful collaboration. Your attention to detail and refusal to accept a result until every step is carefully checked served as inspiration during a time I became interested in improving my own problem solving processes. I hope our paths will cross again, even if it's just to give a reference for your PhD application.

Alex: Thank you for being a great friend throughout my PhD. From hiking in the misty mountains of La Gomera, to gossiping about supervisors and internships in St Hugh's, you coloured the experience with happy memories. I'm looking forward to more memories to come.

My parents: Thank you for never hesitating to offer your support during these last 4 years. Thank you for booking me flights to see family for Christmas each year. Thank you for always nurturing and supporting my academic interests.

Brigita: Thank you for helping me to get outside more often. Thank you for reminding me of how to be more carefree and happy in the moment. Thank you for all the adventures we've had together, and for the ones still to come.

FHI and the FHI community: Thank you for funding my PhD. Thank you Owen for helping me to meet others and get involved in my first few years at Oxford; I really enjoyed ESPR, and meeting the people you organised conversations with. Thank you Lewis for being a friend at FHI, and for the interesting stories of everything you've been working on.

Open Philanthropy: Thank you for funding the final year of my PhD; the lifelong learning chapter may have been much shorter without your support. I hope the ideas developed lead to further research which aligns with the institution's goals.

Declaration

I certify that I have written this thesis by myself. Chapter 3 is based on my own work published in IJCAI'21 (Orton, 2021). Chapter 4 is based on co-authored work published in NeurIPS'22 (Orton and Falck, 2022). This is an equal-contribution paper with Damon Falck. All theoretical results presented in Chapter 4 are a product of collaborative work or my own contribution. The benchmarking experiments in Section 4.4 were implemented solely by Falck. Chapter 5 is based on a collection of my own currently unpublished work.

Contents

Chapter 1

Introduction

Consider the following two problems:

1. What is the optimal way for a computer program to play chess?
2. How should a robot act in the world in order to maximize positive feedback?

Both of these problems have mathematically precise, "optimal" solutions in established models of Artificial Intelligence (AI). For problem 1, game theory tells us that chess can be modelled as a mini-max game where the optimal chess playing strategy can be recursively defined. For problem 2, Hutter's AIXI (Hutter, 2005) gives an optimal algorithm for maximizing positive feedback in general environments.

Besides being optimal, both of these solutions share another common feature: they are both computationally intractable to implement on a computer. In the case of chess, the optimal game theory strategy requires evaluating at least 10^{120} chess board states.[1] In the case of robot learning, AIXI is uncomputable, meaning that no computer could implement it even if it were given an unlimited amount of computational resources. Even computable forms of AIXI are computationally intractable.

The problem of models giving intractable optimal solutions is not restricted to these two examples. Indeed, a number of well established fields, from Bayesian decision theory (Berger, 1985) to computational learning theory (Kearns and Vazirani,

[1] 10^{120} is also known as the Shannon number. It is a lower bound on the number of board states which can be reached during a chess game.

1994), define "optimal" solutions to problems where the solutions themselves are often computationally intractable to implement or find. Needless to say, in the most pronounced cases, this can be a fairly significant problem. While the theory tells us what is optimal, it doesn't tell us what to actually do. As Russell and Wefald put it in "Do the Right Thing", many models "abstract away a crucial aspect of the process of doing the right thing, by associating 'rightness' with the action finally taken, rather than with the whole process of deliberating and acting" which itself may be restricted by computational and informational constraints (Russell and Wefald, 1991).

In 1957, the political scientist Herbert Simon published "Models of Man", an influential work which would go on to shape research in computer science, economics and cognitive psychology (Simon, 1957). In it, he criticized the prevailing models in economics, which modelled people as perfectly rational agents. In contrast, Simon argued that people are limited by their information processing ability, and that more accurate models of rationality would need to take this into account. Following this work, the field of bounded rationality began to emerge. Within the field of computer science, bounded rationality became concerned with understanding what it meant for computer programs to be optimal subject to limitations on their computational capabilities. These limitations could include restrictions on processing time, memory usage, or availability of information. Despite considerable interest, 24 years following Simon's original publication, the economist Ariel Rubinstein argued that "although Simon's ideas have received worldwide recognition [...] it is difficult to embed the procedural aspects of decision making in formal economics models" (Rubinstein, 1986). In hindsight, we now know there is a deep reason why researchers find building models of bounded rationality so challenging. It is no coincidence that fields concerned with optimal decision making frequently model their solutions as if computation were an unrestricted resource. Following the development of computational complexity theory (see Arora and Barak (2009)), it became clear that proving specific computational

tasks require a certain level of computational resources is an extremely difficult problem. Perhaps the most famous example is the P vs NP Millennium Prize problem, which offers $1 million to whomever resolves whether the class of problems which have polynomial-time solution verifiers also have polynomial-time solvers. Are there computationally efficient computer programs which can factor large integers? Despite decades of research, we still don't know.

What does this mean for the study of bounded rationality in computer science? It means that unless one carefully tailors their model to avoid implying consequences to these very difficult problems, their model is unlikely to give an explicit account of how to behave optimally under computational constraints. For example, suppose one wanted to build a general theory which makes explicit predictions about how computer programs with limited computational resources play games optimally. For a game whose goal is to factor large integers, such a theory must then say something specific about the optimal computationally bounded program for integer factorization. But saying something nontrivial about this program is widely believed to be a very challenging problem in complexity theory. Hence it is unlikely such a theory would be able to make explicit predictions for games in general in the first place. There is therefore a careful balancing act between giving general models of bounded rationality, and avoiding models which have implications for these very challenging problems in complexity theory. We argue that this fundamental issue has played a key role in shaping bounded rationality research in computer science. Because of it, researchers have often needed to demonstrate ingenuity in their models. Rather than tackling the problem head on, one often needs to create specific models for specific aspects of the problem. Over time, the cumulative sum of different modelling approaches sheds light on the overall problem.

Early work in bounded rationality (Aumann, 1981; Ben-Porath and Peleg, 1987; Neyman, 1985; Papadimitriou and Yannakakis, 1994) focused on modelling computa-

tion with finite automata, where limited information processing ability corresponded to having fewer states in a finite automaton. One key result showed that repeated games of the Prisoner's Dilemma have cooperative Nash equilibria if the number of states in the automata are limited, even though there are no cooperative Nash equilibria in the unconstrained version of the game when the number of rounds is finite (Neyman, 1985). Other key results included quantifying the complexity of strategies by the number of states needed to implement them in a finite automaton (Kalai and Stanford, 1988), and quantifying the advantage of having more states (Ben-Porath and Peleg, 1987).

Later work can roughly be divided into two approaches. The first approach models computation explicitly in the form of general computer programs (e.g. Turing machines) (Fortnow and Whang, 1994; Tennenholtz, 2004; Fortnow and Santhanam, 2010; Halpern and Pass, 2015; Oesterheld, 2018). As a result, this area is often concerned with high level results such as under which conditions Nash equilibria exist in computational games (Halpern and Pass, 2015; Halpern et al., 2019), or the behaviour of computer programs in limiting cases (Fortnow and Santhanam, 2010). Because these models deal with computer programs in full generality, as argued previously, it is much more difficult to derive explicit predictions about bounded rationality in finite cases.

The second approach models computation implicitly. Instead of explicitly reasoning about computer programs, one builds models which are intended to correspond to some aspect of bounded computation one is interested in. Work in this area is diverse and heterogeneous. Examples include modelling bounded memory with imperfect information games (Dow, 1991; Rubinstein, 1997), games with explicit complexity costs on strategies (Ben-Sasson et al., 2006), anytime algorithms which optimize the resource allocation of performance profiles (Zilberstein and Russell, 1996), models of how search tree depth affects strategy performance (Beal, 1980; Nau, 1979; Pearl,

1980), meta reasoning about optimal behaviour (Russell and Wefald, 1991; Conitzer and Sandholm, 2003), treating compute allocation as a learning problem (Balcan et al., 2018, 2017), and studying runtime selection in the context of multi-armed bandits and online learning (Dick et al., 2020; Kleinberg et al., 2017; Weisz et al., 2019; Balcan et al., 2020). There is therefore a substantial variety of different ways researchers have approached modelling specific aspects of bounded rationality where computational constraints are implicit in the model. In contrast to the first modelling approach, results in this area tend to lead to fairly concrete algorithms computationally constrained programs can implement. However, they are usually restricted to specific contexts, and therefore lack the same degree of generality as the first approach.

In this thesis, we argue that Lifelong Learning (LL) (see Chen and Liu (2018)) offers a third approach to studying bounded rationality. We are not aware of any prior work which makes this connection, so we will make the case for it here. LL was proposed in Thrun and Mitchell (1995), and has recently been gaining interest due to the rise of deep learning. The LL problem concerns itself with the setting of a continuous, never ending learning process which occurs while performing tasks. The objective is to give efficient algorithms which can continually learn to perform better over an indefinite period of time. Because the problem is never ending, one cannot write down the "final solution" of how a lifelong learner should behave. Rather, one is forced to model the *process* of continually learning. By design, LL therefore helps alleviate the problem identified by Russell and Wefald, where models associate a solution with "the action finally taken", rather than with "the whole process" which is computationally constrained. To see why, it can be helpful to reconsider problem 2 given at the beginning of this introduction, and compare the solutions for it offered by Reinforcement Learning (RL) (see Sutton and Barto (2018)) and lifelong learning. How would RL model the solution of a robot behaving optimally in the

real world to maximize a reward? A common approach is to assume there is some statistical model $\mathcal{M}$ of the environment (e.g. a Markov decision process). If we are to take this idea seriously, then we should imagine $\mathcal{M}$ as being a highly complex, intricate model of the entire world which describes every possible interaction the robot could have. If π is a policy which describes how the robot should behave, then we can write $\mathcal{M}(\pi)$ to denote the expected reward of π acting inside model $\mathcal{M}$. RL then says that the "solution" to problem 2 is the policy $\pi^* = \arg\max_\pi \mathcal{M}(\pi)$ which maximizes the expected reward. The solution given by RL is therefore simply "the action finally taken". Nothing is said about how this solution is actually found, or how it is implemented. There are three reasons why this proposed solution gives an unsatisfactory answer to problem 2:

1. **Bounded knowledge.** Robot agents are not omniscient. When the environment is as complicated as the entire world, the model $\mathcal{M}$ will never be known in sufficient enough detail to find π^*. Rather, the robot will continually learn new knowledge over time, and will need to continually adapt as a result. On the other hand, lifelong learning explicitly models how agents accumulate and maintain knowledge over time so that they are able to respond to new problems.

2. **Bounded memory.** Robot agents have a finite capacity to remember past observations. Even if we could obtain a huge number of samples from $\mathcal{M}$ in order to optimize the policy π, a robot would not be able to use these samples all at once. Lifelong learning explicitly studies how one can manage a finite memory budget without "forgetting" how to perform well on past experiences (McCloskey and Cohen, 1989; Kirkpatrick et al., 2016; Rebuffi et al., 2017; Shin et al., 2017).

3. **Bounded computation.** Suppose the model $\mathcal{M}$ were known. The RL solution is the maximizer of an incredibly complex optimization function, and is therefore

likely to be computationally intractable to find. In contrast, lifelong learning explicitly models how a system can incrementally solve subproblems which are computationally tractable given its current knowledge. As new subproblems are solved and new knowledge is learned, the horizon of computationally tractable problems expands, and the learning process continues. Lifelong learning therefore attempts to explain the process by which the policy π^* can be approached in a computationally tractable way.

With the increasing deployment of deep learning systems, understanding how computationally bounded agents should continually learn is an issue of increasing importance. For example, ChatGPT (Brown et al., 2020) is a chatbot which is periodically updated on new data. A single training session of ChatGPT is estimated to cost around $1.5 million.[2] While we might theoretically model ChatGPT as the solution to a fixed optimization objective as is done in RL, in practice it is used as an evolving system whose knowledge is periodically updated as new data arrives.

1.1 Organisation And Contributions

This thesis contributes three new models for understanding how computationally constrained computer programs should behave in different learning settings. The first two models explore precomputation in the context of extensive form games, and tradeoffs between regret and computation budgets in the context of online learning. They therefore fall under the second approach taken to model bounded rationality: making specific models for studying specific aspects of computational constraints. The third model occurs in the context of lifelong learning, and is therefore an example of the third approach. Chapter 3 is based on my own work published in IJCAI'21 (Orton, 2021). Chapter 4 is based on co-authored work published in NeurIPS'22

[2] https://www.techgoing.com/how-much-does-chatgpt-cost-2-12-million-per-training-for-large-models/.

(Orton and Falck, 2022). This is an equal-contribution paper with Damon Falck. All theoretical results presented in Chapter 4 are a product of collaborative work or my own contribution. The benchmarking experiments in Section 4.4 were implemented solely by Falck. Chapter 5 is based on a collection of my own currently unpublished work.

In Chapter 2 we provide a more detailed context for the fields of bounded rationality in computer science and lifelong learning. We give an overview of prior work and the areas of research in both fields.

In Chapter 3 we provide a model for the precomputation aspect of two player extensive form games. We imagine that before a game, both players can prepare in advance by memorizing strong moves given by an oracle. However, they have limited memory capacity to remember these moves. This model can be seen as a way of quantifying some of the observations made by Simon and Schaeffer about how professional human players tend to rely on memorized board positions while playing chess (Simon and Schaeffer, 1992). It can be contrasted with work such as Ben-Sasson et al. (2006) which models game playing where players pay a penalty for playing more computationally demanding strategies. Our model is specific to extensive form games and motivated by the idea of precomputation, while Ben-Sasson et al. (2006) considers normal form games with abstract complexity costs. A key result of this chapter includes being able to analytically lowerbound how randomly rational players play in Nash equilibria. Players need to trade off playing predictable "optimal" moves, while randomizing their strategy so that it is difficult for their opponent to precompute against them. Furthermore, we show the existence of polynomial-time algorithms for computing the best response and ε-Nash equilibrium. We run the best response algorithm on Stockfish, a chess playing algorithm, and empirically measure how susceptible it is to precomputation. As expected, we find a turning point where the benefit of playing more randomly is outweighed by the cost of playing

"suboptimal" moves more frequently.

Chapter 4 considers how to trade off processing time budgets for stronger regret guarantees. The work is analogous to prior work in anytime algorithms (Zilberstein and Russell, 1996) and online learning with computational budgets (Dick et al., 2020), where one wants to study the relationship between processing time budgets and performance. Chapter 4 occurs in the setting of online learning (see Cesa-Bianchi and Lugosi (2006) and Lattimore and Szepesvári (2020)). In each round $t \in [T]$, an algorithm has a computational budget to run B algorithms $S_t \subset [N]$ to solve a problem, and incurs a cost $c_t(S_t)$ equal to the cost of the best solution found amongst the algorithms chosen. We study the tradeoff between the computational budget B, and the expected regret R_T relative to the cost of the single best fixed algorithm in hindsight. When $B = 1$, it is well known that one can achieve $R_T \leq 2\sqrt{T \ln(N)}$ (Littlestone and Warmuth, 1994), while when $B = N$ one can trivially achieve $R_T = 0$. Our contribution is to study the intermediate regime where $1 < B < N$. By adapting the Follow The Perturbed Leader technique (Kalai and Vempala, 2005), we show how to give efficient algorithms which achieve regret $\mathcal{O}(T^{\frac{1}{B+1}} \ln(N)^{\frac{B}{B+1}})$ in the full feedback setting and $\mathcal{O}(T^{\frac{1}{B+1}} (K \ln(N))^{\frac{B}{B+1}})$ in the semi-bandit feedback setting given unbiased cost estimates $\hat{c}_t \in [0, K]$. This allows one to trade off a compute budget B for improved performance guarantees. We then show how solutions to this problem can have independent relevance to the wide family of algorithms which use regret minimizers as a subroutine (Arora et al., 2012). First, we show how to adapt the algorithm **OG** (Streeter and Golovin, 2008) for Online Submodular Function Maximization to give a regret bound which trades off a budget for stronger regret guarantees. We then benchmark this adapted algorithm on the task of online black box hyperparameter optimization, and observe improved performance. Secondly, we show how to automatically get new algorithms for linear programming which trade off oracle power for faster convergence rates.

Chapter 5 occurs in the setting of lifelong learning. It makes a number of independent contributions.

In Section 5.1, we consider the problem of how to incrementally accumulate knowledge over time. Knowledge is given to the algorithm online in the form of experts, where we do not know ahead of time whether a particular expert is "good" (e.g. will generalize) or "bad" due to non-stationarity (i.e. bounded knowledge). We introduce an efficient algorithm **FTGE** which guarantees that its average performance is at least as good as if we could look into the future and know which pieces of knowledge are "good" in advance. Its guarantee relies on ensuring that the rate at which knowledge is accumulated does not grow too quickly.

In Section 5.2, we introduce a universal framework for modelling knowledge in the context of incremental problem solving. We also introduce the conceptual algorithm **IPS** for studying incremental problem solving by a computationally bounded agent. It is based on extending and formalizing ideas in program synthesis in Ellis et al. (2020). The model is universal in two senses. Firstly, it can be applied to many different domains of learning automatically. This is significant because, in the field of LL, there is currently no consensus on what knowledge is or how to formally model it (Chen and Liu, 2018). Instead, researchers use different specific representations of knowledge depending on the specific problem they consider. Secondly, the model is universal in the sense that we prove there is no "short counterexample" of an incremental learning algorithm which performs significantly better than **IPS**. The framework gives a conceptual way of thinking about incremental problem solving in the lifelong learning setting.

In Section 5.3, we combine the algorithms from sections 5.1 and 5.2 to build an algorithmic framework called **ALRL** for a model based lifelong reinforcement learning algorithm. Existing approaches to lifelong reinforcement learning are generally either black box optimization approaches with neural networks (Ring, 1998; Tessler

et al., 2017; Mendez et al., 2022), or based on very principled idealized statistical models (Wilson et al., 2007; Brunskill and Li, 2014; Lecarpentier et al., 2021; Wang et al., 2022). While the black box methods can work nicely in practice for specific problems, they lack interpretable guarantees which allow them to be used as general lifelong reinforcement learners. On the other hand, the more rigorous methods tend to be unable to model meta knowledge, or do not benefit from the good empirical performance of black box techniques. **ALRL** is intended to make progress on both of these challenges. **ALRL** inherits high level theoretical guarantees from the algorithms derived in sections 5.1 and 5.2, and is able to model meta knowledge. On the other hand, **ALRL** is built on top of an oracle where practitioners can implement their own black box problem solvers (e.g. neural networks) to search for programs which are effective at solving RL tasks. The framework therefore attempts to make the best of theoretical guarantees from formal methods as well as heuristic black box techniques which can perform well in practice.

Chapter 5 ends with Section 5.4, where we briefly discuss an idea for how one can model incremental knowledge for world models in the form of objects and relations. It is a fusion of two existing techniques (Kipf et al., 2020; Hafner et al., 2023) in model based reinforcement learning. This section therefore contributes to existing work in knowledge representation techniques for LL (Thrun, 1996; Ruvolo and Eaton, 2013b; Pentina and Urner, 2016).

Notation: Because this thesis spans multiple subject areas, the notational tools can vary by chapter. We keep the following general notation conventions throughout. For an integer N, $[N] = \{1, \ldots, N\}$ denotes the set of integers from 1 to N. For a set S, $\Delta(S)$ denotes the probability simplex over the set S. Names of algorithms such as **FPML** appear in boldface.

Chapter 2

Background And Related Work

In this chapter, we give a more detailed account of the context in which this thesis occurs. We survey existing work in bounded rationality and lifelong learning, and outline some of the main approaches taken in each field.

2.1 Bounded Rationality

The origins of the study of bounded rationality are usually attributed to Simon following the publication of "Models of Man" in 1957 (Simon, 1957). Simon argued that current economic theories which modeled people as purely rational agents were too idealized. Rather, humans are limited by their cognitive ability. Moreover, Simon argued that cognitive ability could be viewed as a form of information processing, and modelled by computation. In the 1950s, Simon worked with Allen Newell to create a computational system which could execute complex thought processes similar to how a human with bounded information processing ability might reason. The result was the "Logic Theorist", a formal symbolic program which could prove theorems in propositional logic (Gugerty, 2006). Simon's initial work was grounded in both economics and computation. However, the field of bounded rationality has since branched into many diverse and heterogeneous subfields including psychology and behavioural economics (see e.g. Kahneman (2003)). In this thesis, we are interested in the problem of how *computer programs* with bounded resources can behave opti-

mally. We therefore focus solely on prior work at the intersection of computer science and bounded rationality for the remainder of this section.

2.1.1 Early Work: Finite Automata

Early work on (computational) bounded rationality in the 1980s focused on finite automata and repeated games of the Prisoner's Dilemma. The Prisoner's Dilemma is a game where players need to simultaneously choose whether to cooperate or defect. The result of the game is given by the following payoff matrix:

		Player Y	
		C	D
Player X	C	$(3,3)$	$(0,4)$
	D	$(4,0)$	$(1,1)$

In the repeated Prisoner's Dilemma, this game is played repeatedly for a known finite number of rounds. It can be shown by backward induction that the only Nash equilibrium in this game is for both players to always choose to defect. This result appears unsatisfactory, because human players often choose to cooperate (e.g. play a tit for tat strategy) to obtain a collectively higher payoff. In 1981 Aumann proposed modelling computationally bounded players as finite automata (FA) for this repeated game (Aumann, 1981). A finite automata consists of a finite number of states, a rule for which action to play in each state, and a rule for how to transition to a new state after observing an opponent's move. Limiting the number of states of the automaton can be thought as controlling the memory or complexity of the strategy. Following Aumann's proposal, a number of lines of research emerged. An influential result by Neyman showed that if the number of states of the automaton is restricted, then there are cooperative equilibria for this repeated game (Neyman, 1985). However, a published proof of this fact only appeared sometime later in Papadimitriou and Yannakakis (1994). Formally, for the n round Prisoner's Dilemma game with $\varepsilon > 0$, if at least one player's strategy has fewer than $2^{\frac{n\varepsilon}{6(1+\varepsilon)}}$ states, it was shown that there

exists a mixed Nash equilibrium with average payoff for each player $\geq 3-\varepsilon$. Moreover, Papadimitriou and Yannakakis (1994) showed how to prove this kind of cooperation result for any repeated game played by FA. The high level idea is for either player to punish the other if they do not behave in a way which requires a large amount of memory to compute. Because all their memory is used to avoid punishment, there is not enough memory left to count the number of rounds of the game elapsed. This means that it isn't possible to play the strategy which e.g. behaves cooperatively and chooses to defect at the very last round. On the other hand, the result of cooperation can be sensitive to the model choice. For example, Rubinstein (1986) studied a different variant of an infinitely repeated Prisoner's Dilemma game played by FA where cooperation does not occur. The main difference of this variant is that limited processing power is modelled exogenously. A player can play an automaton strategy of any size, and aims to maximize their average utility. However, their average utility is the sum of the utility from the Prisoner's Dilemma game minus a small fee which is linearly related to the size of the automaton strategy. This contrasts with the endogenous model of bounded computation in Neyman (1985), where there are no exogenous strategy computation costs, but the set of strategies a player is allowed to choose from is limited to automata with a bounded number of states.

A separate line of work studied how to relate the power of a strategy to its complexity as a finite automaton. Ben-Porath and Peleg (1987) found that in repeated two player zero sum games, there is asymptotically no gain in strategy value unless the automaton is exponentially larger. Kalai and Stanford (1988) gave an automata model which applied to general infinitely repeated games, and showed a correspondence between every game strategy and its minimal automaton. This allowed one to give a complexity measure of a strategy in terms of its corresponding FA. Even today, FA are still used as a proxy for understanding restricted game strategies. For example, Liu and Halpern (2023) studied the ranger poacher game played with finite

automata, and claimed that one observes human like behaviour when the number of states of the automata is restricted.

2.1.2 General Models

Later work began to consider more advanced models of computation. The most general form of this is to model computation being implemented by a Turing machine or general computer code. As argued in Chapter 1, proving unconditional results about specific problems in this setting can be as hard as solving long standing problems in computational complexity theory. Because of this, research in this area is often concerned with high level results (e.g. whether equilibria exist) and limiting cases.

Megiddo and Wigderson (1986) studied the repeated Prisoner's Dilemma game where strategies are deterministic Turing machines with unlimited space and processing ability, and each machine receives the number of rounds n as input. They showed that even just restricting strategies to be Turing machines imposes some restriction on the class of strategies (they have to be computable). Additionally, one can prove that there is no cooperative Nash equilibria by a simulation argument (a player's strategy can simulate the strategy of another player to determine whether to defect). When the number of states of the rule set of the Turing machine is bounded (i.e. it's program size is limited), Megiddo and Wigderson (1986) give an argument similar to Papadimitriou and Yannakakis (1994) that there should be a cooperative Nash equilibria (but do not prove it). This is done by constructing strategies where most of the code of the Turing machine is used in computing how to avoid punishment by the other player. Similarly, Fortnow and Whang (1994) studied when Turing machines or finite automata can be approximately optimal or dominant strategies in a form of infinite Prisoner's Dilemma game.

A different line of work introduced in Tennenholtz (2004) considers a game where strategies are computer programs, and programs receive the code of other players'

strategies before playing. The main motivation for this is to allow co-operation in equilibrium, since players can now change their behaviour depending on whether e.g. the code of the other player is cooperative. For example, this allows a cooperative Nash equilibrium for the Prisoner's Dilemma. Related work looks at incorporating model checking into this framework (van der Hoek et al., 2013), and choosing strategies to make cooperative equilibria more robust (Oesterheld, 2018). In the original model, Tennenholtz (2004) restricts programs to be straight line programs (i.e. have no loops and always terminate). Fortnow (2009) extends this model by allowing arbitrary programs (Turing machines), where player i's utility is discounted by a factor of δ^{t_i} where t_i is the time it takes player i's strategy to compute an action. Fortnow (2009) then proves a folk theorem about the existence of program equilibria in this model. The idea of discounting utility exponentially by computation time is also explored by Fortnow and Santhanam (2010). They consider a factoring game where one player chooses an integer and the other needs to factor it. The main result is to show that as the discounting factor δ approaches 1, there is a stable Nash equilibria (i.e. it doesn't change as δ gets sufficiently close to 1) whose payoff depends on whether there are polynomial-time factoring algorithms.

Halpern and Pass (2015) introduce Bayesian machine games where strategies are programs, and the game consists of a single round where strategies receive a type and must output a single action. The utility of the game depends not only on the actions output by the strategies but also the complexity of the programs. Because one is interested in behaviour which can be implemented by an efficient program, one only considers pure Nash equilibria. For example, the strategy consisting of a general mixed distribution over programs (strategies) may not be efficiently implementable by a single program. On the other hand, if a mixed distribution of strategies can be sampled from by an efficient program, then one can simulate this mixed distribution with a single efficient program. These observations motivate the reasoning behind why

it makes sense to only consider pure Nash equilibria. In this model, it is possible for even simple games to not have pure Nash equilibria. For example, Halpern and Pass (2015) consider the game of rock-paper-scissors where deterministic programs have a complexity cost of 1, but randomized programs have a complexity cost of 2. Because randomization is required to achieve Nash equilibrium in this game, but is too costly, there is no Nash equilibrium. They then extend the model to allow communication between players through a mediator, and show how standard results like the revelation principle in traditional game theory models do not hold in this model. Another model of computational games include extensive form computational games (Halpern et al., 2016). Here, one imagines a growing sequence of games which correspond to an underlying game, where two histories which are in the same information set (indistinguishable) in the underlying game are computationally indistinguishable in the game sequence. This can be used for modelling cryptographic protocols. Similarly, Halpern et al. (2019) consider a growing sequence of Bayesian games where the action space and utility function can be computed in polynomial-time, but there is no Nash equilibrium consisting of polynomial-time strategies (assuming the existence of one-way functions). The idea behind this is that there is a "computational arms race", where it is always advantageous to have slightly more computation.

2.1.3 Implicit Models Of Computation

Instead of modelling computation explicitly as a computer program, another approach is to build specialized models which implicitly capture some aspect of bounded computation one is interested in studying. Researchers have considered a variety of heterogeneous models.

In the context of bounded memory, one can use imperfect information games to model limited recall (Rubinstein, 1997). For example, Dow (1991) considers a game where one sequentially observes two prices before making a decision, but knows they

will not be able to remember the first price exactly. Knowledge of the first price is then modelled as a partition of possible prices the first price could be.

Another simple approach is to subtract a cost from a player's utility related to the complexity of the player's strategy. Ben-Sasson et al. (2006) study normal form games where each strategy has an associated complexity cost. A first key result is that if the game without costs is zero sum, then the game with costs has Nash equilibria which can be easily characterized in terms of the original game. This means they can be computed by finding the equilibria of the original game (which can be done efficiently because the original game is zero sum). A second key result concerns potential form games. A potential form game has a potential function Φ such that if σ,σ' are identical strategy profiles for all players except that player i's strategy is different in σ', then the difference in utility of player i can be written as $u_i(\sigma) - u_i(\sigma') = \Phi(\sigma) - \Phi(\sigma')$. Potential form games include routing games (e.g. avoiding congestion in traffic flow), and have nice properties for finding their Nash equilibria. A key result in Ben-Sasson et al. (2006) is that potential form games with complexity costs are still potential form games.

A different line of work tries to explain why the strategy quality of depth bounded mini-max search tends to improve as the depth (and search time) increases (Beal, 1980; Nau, 1979; Pearl, 1980). For example, chess engines are observed to increase in strength as their search depth increases (Ferreira, 2013). Pearl introduced a model where mini-max search is done on a game tree where once the search depth limit is reached, a heuristic function returns a noisy estimate of the expected value of the game at the current game state (Pearl, 1980). A pathological result of this model is that as the search depth increases, the quality of the estimate at the root of the game tree degrades. Subsequent work has proposed alternative models to avoid this pathology (see e.g. Luštrek et al. (2006)).

Meta Reasoning

Meta reasoning (see Zilberstein (2011)) encompasses a broad family of modelling approaches to bounded computational resources. Broadly speaking, this approach includes (1) learning meta rules for deciding what to do under computational constraints and (2) reasoning about which meta actions to take in order to balance performance and computational costs. Here, one imagines starting with a base set of procedures which themselves take computational resources, and a meta action is a decision about how to use these base procedures.

Two classical examples from "Do the Right Thing" by Russell and Wefald (Russell and Wefald, 1991) include **MGSS*** for game playing and **LDTA*** for search. **MGSS*** works in mini-max games. Naively one solves a mini-max game by performing a bounded depth search on the game tree and using a heuristic function to evaluate the game state at the depth limit. Instead of considering every state to a bounded depth, **MGSS*** tries to reason about the value of searching further past a particular state, while taking the cost of computation into account. One can then focus on expanding the states which are most likely to improve the estimate of the value of the game. **LDTA*** is conceptually similar but for search problems. It also updates its parameters as new states are searched in an attempt to improve its meta reasoning ability. A more modern example of these kinds of algorithms is "Learning to Branch" (Balcan et al., 2018). The setting occurs in tree search algorithms for mixed integer linear programming. One is given a collection of scoring rules $\{\text{score}_i\}_{i\in[n]}$ for deciding which states of the search tree to expand; in a formal learning setting, Balcan et al. (2018) study how to learn an optimal convex combination $\sum_{i=1}^{n} \lambda_i \text{score}_i$ of the base scoring rules to give a new scoring rule which minimizes the runtime of the search algorithm. Similarly, Balcan et al. (2017) study how to learn algorithms which minimize their runtime costs for combinatorial partitioning problems (e.g. clustering and max-cut), when these problems come from an application specific distribution. These papers

occur in the Probably Approximately Correct (PAC) learning setting (Valiant, 1984) (see Kearns and Vazirani (1994)). The approach of "learning" algorithms which perform well on tasks in the PAC learning setting was first introduced by Gupta and Roughgarden (2016).

Anytime algorithms (Russell and Subramanian, 1995; Zilberstein and Russell, 1996) model meta reasoning by imagining one can run an algorithm for varying amounts of time, where the quality of the solution the algorithm returns monotonically increases as more time is allocated. Here one assumes they have access to a performance profile $D_A : \mathbb{R}_{>0} \to \Delta(\mathbb{R})$ which maps the runtime of an algorithm A to a distribution over the solution quality of its output. One can then study how to optimally allocate a runtime budget B across algorithms to maximize solution quality. One can also consider composing anytime algorithms by specifying how the quality of the inputs to a program affects the quality of its outputs (as a function of processing time). Using dynamic programming, it is possible to compose anytime algorithms together with a tree structure and calculate the optimal resource allocation for each program in the tree.

While proving an algorithm is optimal subject to computational constraints is very challenging, meta reasoning can be appealing because it defines a new problem (e.g. optimizing performance profiles) which may seem more tractable to give "optimal" solutions for. However, as others have noted (Conitzer and Sandholm, 2003; Conitzer, 2011), sometimes the meta problem itself can be computationally intractable (e.g. NP-hard) to solve. One needs to be careful to not try to solve the intractability of the original problem by defining an intractable meta problem.

Online Learning

Online learning is a framework for decision making where information is revealed to the algorithm over time. The environment may be stochastic or adversarial. One is typically concerned with behaving in a way that minimizes hindsight regret, i.e.

the difference between the performance of the algorithm, and the performance of the algorithm which could have been achieved if you had known what would happen in the future.

While there is a long history of problem variants in online learning (see Cesa-Bianchi and Lugosi (2006) and Lattimore and Szepesvári (2020)), the setting where bandit arms correspond to running an algorithm for a certain time has only been recently studied. The problem, first introduced in Kleinberg et al. (2017), considers the following idea: suppose one has N algorithms, and one receives problem instances $j \sim \mathcal{D}$ from some unknown distribution. Let $R(i,j) \in [0,\infty]$ be the runtime of $i \in [N]$ on input j, and $R_\tau(i,j)$ the runtime capped at τ. The problem is to quickly find an $i \in [N], \tau \in \mathbb{R}$ such that (a) $\mathbb{P}_{j\sim\mathcal{D}}[R(i,j) > \tau] \leq \delta$ and (b) $\mathbb{E}_{j\sim\mathcal{D}}[R_\tau(i,j)]$ is "optimal" among all other $i' \in [N]$. The original paper defines "optimal" to mean $\forall i' \in [N]$, $\mathbb{E}_{j\sim\mathcal{D}}[R_\tau(i,j)] \leq (1+\varepsilon)\,\mathbb{E}_{j\sim\mathcal{D}}[R_{\tau_{\max}}(i',j)]$ where $\tau_{\max}$ is some very large runtime one is unwilling to exceed on any particular instance. The authors give an algorithm which is near optimal for this particular problem variant. The key difference between runtime bandits, and more generic online learning problems is (1) the runtime (i.e. loss) can be unbounded, and (2) one can adaptively choose to run algorithms (pull arms) for finite amounts of time, instead of incurring the full loss for a choice after a single pull.

Future papers build on and improve the problem introduced by Kleinberg et al. (2017). Weisz et al. (2018) remove the assumption of knowing a $\tau_{\max}$ in advance, and slightly improve the runtime of the meta algorithm. Moreover, if the observed variance of runtimes is low, this is exploited. Kleinberg et al. (2019) make the algorithm appearing in Kleinberg et al. (2017) more adaptive in the sense that it performs better when many $i \in [N]$'s perform poorly. The new algorithm also provides an anytime property: it can be asked to choose an $i \in [N]$ at any time during the algorithm's execution, and the guarantees for the $i \in [N]$ it chooses improve with the

duration the algorithm is run for. Weisz et al. (2019) extend the original definition of an optimal arm to be more robust. Balcan et al. (2020) study the case where there is some structure to $R(i, j)$, and the parameter i comes from a continuous parameter space. Specifically, they assume $R_\tau(i, j)$ is piecewise constant for fixed j, τ. This is motivated by the structure of certain optimization problems including integer program optimization. The goal is to find a small set of finite parameters which contains at least one good parameter. One can then feed this set into the algorithms which work for finite parameter spaces.

The above problems occur in the stochastic bandit setting, where one makes distributional assumptions about the runtime. In contrast, Dick et al. (2020) study the adversarial setting. In each round t one needs to choose an algorithm $i_t \in \mathbb{R}^d$, and observes runtime cost $c_t(i_t)$. The goal is to minimize the total runtime incurred over all rounds relative to choosing the best in hindsight fixed algorithm $i \in [N]$ across all the rounds. There is an implicit max runtime cap $\tau_{\max}$, and one only observes the runtime for the arms which are pulled. The authors get a regret bound by assuming the runtime for round t is given by a loss function $l_t : \mathbb{R}^d \to [0, 1]$ which is piecewise Lipchitz (similar to the assumption in Balcan et al. (2020))

2.2 Lifelong Learning

Lifelong Learning (LL) concerns itself with the setting of a continuous, never ending learning process which occurs while performing tasks. In order to improve its ability to perform tasks over time, a LL system continually accumulates and maintains a knowledge base derived from past experience. As argued in Chapter 1, LL can be seen as a natural response to bounded rationality. Due to limited knowledge and computation, it is intractable for an algorithm to give a solution to a large problem all at once. Instead, it needs to continually accumulate knowledge and solve tractable subproblems over time, using its accumulated knowledge to make previously intractable

subproblems solvable. The concept of lifelong learning appeared in "Lifelong Robot Learning" in 1995 by Thrun and Mitchell (Thrun and Mitchell, 1995). Since then, a number of different classes of LL have been considered, including:

1. **Lifelong supervised learning.** An algorithm receives new classification tasks online, and needs to incrementally learn each new classification task while re-using prior knowledge.

2. **Continual learning in deep neural networks.** An algorithm has a limited memory budget (cannot remember all prior experience), and needs to maintain a neural network which learns to perform new tasks without forgetting prior tasks.

3. **Lifelong unsupervised and semi-supervised learning.** Lifelong learning tasks where the objective may only be partially specified. For example, the task may be to accumulate a knowledge representation of data over time, or to decide whether new data belongs in the same category as previously learned knowledge.

4. **Lifelong reinforcement learning.** A reinforcement learning agent continually accumulates and uses knowledge to improve reward performance over time.

An excellent survey resource of the field is "Lifelong Machine Learning" by Chen and Liu (2018). As can be seen, the scope of LL is broad and diverse. Perhaps because of this breadth, the authors of Chen and Liu (2018) consider LL to be an "emerging field" where "our understanding of it is still limited". Indeed, there is no universally accepted definition of LL, nor a unified view on what knowledge is or how to define it. Instead, prior work has used different definitions for different specialized settings. It is therefore impractical to give a complete account of every approach in this literature review. Instead, we will focus on giving a holistic account of each topic area, complemented with illustrative examples of specific techniques.

2.2.1 Lifelong Supervised Learning

Lifelong supervised learning consists of learning a sequence of classification tasks $\mathcal{T}_1, \ldots, \mathcal{T}_N$ in an online fashion (where N may be unknown). Each task $\mathcal{T}_i$ consists of a labeled classification dataset $\mathcal{D}_i$ with $|\mathcal{D}_i| = n_i$. The ith task is to produce a classifier for dataset $\mathcal{D}_i$. A key goal is to somehow accumulate and use knowledge over time in order to improve classification performance in future tasks.

A useful way of differentiating existing approaches to lifelong supervised learning is by considering the different ways they model knowledge. An early proposal by Thrun (1996) models shared knowledge as a representation function $g : \mathcal{X} \to \mathbb{R}^d$ which embeds observations into a shared representation space across tasks. Here $\mathcal{X}$ is the input space, and $\mathbb{R}^d$ is the embedding space in which one learns a representation of the data. More abstract and heuristic methods such as in Silver et al. (2015) model shared knowledge as shared weights in a neural network, while concrete heuristic methods such as in Shu et al. (2017a) model knowledge as components in a conditional random fields model. On the more formal side, Ruvolo and Eaton (2013b) introduce the lifelong learning algorithm ELLA, which assumes the ith task can be encoded in a parameter vector $\theta_i \in \mathbb{R}$. Shared knowledge is modelled by assuming θ_i can be represented as a sparse linear combination of a library L of vectors shared between tasks. ELLA is extended in Ruvolo and Eaton (2013a) to the context of active task selection, where the algorithm chooses to observe new data based on the expected information gain for the shared knowledge library L. In the context of document topic classification, Chen et al. (2015) propose learning a naive Bayes classifier, where knowledge consists of a continually growing library of the frequencies of each word encountered per topic. Lifelong supervised learning has also been studied in the context of formal learning theory. In Pentina and Lampert (2014), knowledge is modeled by a prior distribution over classifiers; a new task is learned using Bayesian learning (i.e. applying Bayes' theorem to infer the new classifier). After the task is

learned, the posterior over classifiers is updated, and the next task classifier is inferred using this posterior. Another example comes from Pentina and Urner (2016) in the Probably Approximately Correct (PAC) learning setting, where it is assumed that future classifiers can be modelled as a weighted majority of previously learned classifiers. Another line of work (Ellis et al., 2018, 2020) models knowledge as subroutines of programs. Here, one learns pieces of program code to solve supervised problems, where the ability of the system to synthesize code improves as useful subroutines are continually added to a concept library. This work is discussed in detail in Chapter 5.

Different approaches to lifelong supervised learning each have their respective strengths and drawbacks. More heuristics methods such as Silver et al. (2015) and Shu et al. (2017a) can be tailored to practical applications but may not give insight into how to build general lifelong learning systems. More theoretical examples such as Pentina and Lampert (2014) and Pentina and Urner (2016) can offer clean theoretical models of knowledge which may aid in transparent understanding of the problem, but may be intractable in practice or too idealized to be useful. We now survey the details of some of the prior works mentioned to give a more concrete idea of how these ideas are implemented.

Nearest Neighbors And Representation Functions

Thrun (1996) is an early suggestion for supervised lifelong learning, motivated by the problem of concept learning. Each task consists of receiving inputs $x \in \mathcal{X}$ (e.g. an image of a cat or dog), and deciding whether x is an example of a target concept for that task (i.e. classify 1 if x is an image of a dog, 0 otherwise). The key idea is to learn a representation embedding $g : \mathcal{X} \to \mathbb{R}^d$ over the data for all tasks, which may be useful for new tasks. One can then use a nearest neighbors method on the representation g. For a binary classification task, Thrun (1996) proposes to learn g by parameterizing it by a neural network, and to then minimize the energy function

$$E(g) := \sum_{i=1}^{N} \sum_{\substack{(x,y)\in\mathcal{D}_i \\ y=1}} \left[\sum_{\substack{(x',y')\in\mathcal{D}_i \\ y'=1}} \|g(x) - g(x')\|_2 - \sum_{\substack{(x',y')\in\mathcal{D}_i \\ y'=0}} \|g(x) - g(x')\|_2 \right]$$

The intuition behind E is that inputs $x \in \mathcal{X}$ *within the same task* with $y = 1$ should have representations which are close together, while inputs with labels which are different should have representations which are further apart. Let $\mathcal{D}$ be the union of all training datasets observed so far. For classification, given a point $x \in \mathcal{X}$, one weights all the training samples by their distance to x under the embedding g to get a nearest neighbours prediction

$$\text{pred}(x) = \left(\sum_{(x',y')\in\mathcal{D}} \frac{y'}{\|x - x'\|_2 + \varepsilon} \right) \left(\sum_{(x',y')\in\mathcal{D}} \frac{1}{\|x - x'\|_2 + \varepsilon} \right)^{-1}$$

where $\varepsilon > 0$ is a small constant to ensure the expression is well-defined. A drawback of this approach is the need to re-optimize over all the training data once a new task $\mathcal{T}_i$ is received. This imposes a large computational and memory requirement in the lifelong learning setting.

ELLA

Efficient Lifelong Learning Algorithm (ELLA) is a more recent lifelong supervised learning method (Ruvolo and Eaton, 2013b). ELLA assumes that for each task $\mathcal{T}_i$, there is a hidden function f_i which maps an input $x \in \mathbb{R}^d$ to a set of labels or $\mathbb{R}$. The authors assume one can parameterize f_i as $f(\cdot, \theta_i)$ for some parameter $\theta_i \in \mathbb{R}^d$, and "sharing information between tasks" is modelled by supposing that each parameter θ_i can be written as a sparse linear combination of some shared basis set $L \in \text{Mat}_{d\times K}(\mathbb{R})$[1], where the K columns of L form the basis elements. Formally, if we let $\Theta \in \text{Mat}_{d\times N}(\mathbb{R})$ be the matrix with columns $\theta_1, \ldots, \theta_N$, and $S_{K\times N} \in \text{Mat}_{K\times N}(\mathbb{R})$

[1]We denote by $\text{Mat}_{d\times K}(\mathbb{R})$ the set of matrices with d rows and K columns and entries in $\mathbb{R}$

be the matrix consisting of the sparse vector columns $s_1, \ldots, s_N$, then we can write $\Theta = L \times S$. We wish to find the best latent basis L for our problem by minimizing the objective function:

$$\text{Objective}(L) = \frac{1}{N}\sum_{i=1}^{N} \min_{s^i \in \mathbb{R}^K} \left[\frac{1}{n_i} \sum_{j=1}^{n_i} \mathcal{L}(f(x_j^i, Ls^i), y_j^i) + \mu \|s^i\|_1 \right] + \lambda \|L\|_F^2$$

Where $(x_j^i, y_j^i) \in \mathcal{D}_i$ is the jth datapoint in $\mathcal{D}_i$, $Ls^i = \theta_i$ is the parameter for task i, $\mathcal{L}$ is a loss function, $\lambda, \mu > 0$ are regularization parameters, and $\|L\|_F^2$ denotes the Frobenius norm of L. A popular technique for sparse coding problems like these is to alternatively minimize by switching between holding either S or L fixed; note that when $f(x, Ls^i) = (Ls^i)^T x = \theta_i^T x$ is a linear predictor, then this objective is marginally convex in either L or S. The original paper notes two key inefficiencies with directly minimizing the above objective. The first is that it requires storing all the past data and re-optimizing every time a new task is encountered. To deal with this issue, the paper proposes considering a second order Taylor expansion approximation for

$$\frac{1}{n_i} \sum_{j=1}^{n_i} \mathcal{L}(f(x_j^i, Ls^i), y_j^i)$$

around $\theta^{i*} = \arg\min_{\theta \in \mathbb{R}^d} \frac{1}{n_i} \sum_{j=1}^{n_i} \mathcal{L}(f(x_j^i, \theta), y_j^i)$, resulting in the approximation

$$\text{ApproxObjective}(L) = \frac{1}{N}\sum_{i=1}^{N} \min_{s^i \in \mathbb{R}^K} \left[\frac{1}{n_i} \sum_{j=1}^{n_i} \|\theta^{i*} - Ls^i\|_{G^i}^2 + \mu \|s^i\|_1 \right] + \lambda \|L\|_F^2$$

where

$$G^i = \frac{1}{2} \nabla_\theta^2 \frac{1}{n_i} \sum_{j=1}^{n_i} \mathcal{L}(f(x_j^i, Ls^i), y_j^i)|_{\theta = \theta^{i*}}$$

is the hessian of the loss function $\mathcal{L}$ at $\theta = \theta^{i*}$. Using this approximation means that in order to optimize L, one only needs to remember θ^{i*} for each task i, rather

than the observation data. It can be shown that this approximation is exact when $\mathcal{L}$ is the squared loss and $f(x, \theta) = \theta^T x$ is a linear classifier. A second inefficiency with the optimization technique is the need to recompute the sparse codes whenever L is updated after each task. Ruvolo and Eaton (2013b) propose simply fixing the sparse vector s_i for all subsequent tasks $i' > i$, and state that empirical performance was not significantly harmed by this choice.

Weighted Majority Lifelong Expert Learning

Pentina and Urner (2016) explore supervised lifelong learning in a more formal framework of computational learning theory. "An Introduction to Computational Learning Theory" by Kearns and Vazirani is a definitive reference for computational learning theory (Kearns and Vazirani, 1994). Here we give an account of the high level ideas in Pentina and Urner (2016). The authors consider binary classification where examples (x, y) come from $\mathbb{R}^d \times \{0, 1\}$, there is a fixed hypothesis class $\mathcal{H}$, and there is a sequence of underlying distributions $W_1, \ldots, W_N$ associated with each task. We assume there is some function $f_i^* : \mathbb{R}^d \to \{0, 1\}$ such that labeled examples (x, y) from distribution W_i are generated via $(x, f_i^*(x))$ for $x \sim W_i$. The goal is to learn, for each task, a function f_i such that the generalization error

$$\mathop{\mathbb{E}}_{x \sim W_i} [\mathcal{L}(f_i(x_i), y_i)]$$

is less than some target $\varepsilon > 0$. Furthermore, in the ith task, one is only allowed to ask for examples from distribution W_i, or make oracle queries to previously learned functions $f_1, \ldots, f_{i-1}$. To do this, the paper models "using prior knowledge" as being able to represent new functions f_i in terms of weighted majorities of previously learned functions. Towards this goal, the following majority vote hypothesis class is defined:

$$\mathrm{MV}(\mathcal{H}, k) = \Bigg\{ g : \mathbb{R}^d \to \{0,1\} | h_1, \ldots, h_k \in \mathcal{H}, w_1, \ldots, w_k \in \mathbb{R},$$
$$g(x) = \mathrm{sign}\left(\sum_{i=1}^{k} w_i h_i(x) \right) \Bigg\}$$

It is shown that we can relate the VC dimension (a measure of the complexity) of the hypothesis class MV to that of the original class $\mathcal{H}$ via

$$\mathrm{VC}(\mathrm{MV}(\mathcal{H}, k)) \leq k(\mathrm{VC}(\mathcal{H}) + 1)3(\log(\mathrm{VC}(\mathcal{H}) + 1) + 2)$$

The next step is to quantify how related new tasks are in terms of previous tasks. To do this, the paper defines distance functions between hypothesises and hypothesis classes:

$$d_{W_i}(h, h') = \mathop{\mathbb{E}}_{x \sim W_i} [h(x) \neq h'(x)]$$
$$d_{W_i}(h, \mathcal{H}') = \min_{h' \in \mathcal{H}} d_{W_i}(h, h')$$
$$d_{W_i}(\mathcal{H}, \mathcal{H}') = \max_{h \in \mathcal{H}} \min_{h' \in \mathcal{H}'} d_{W_i}(h, h')$$

Note that $d_{W_i}(h, \mathcal{H}')$ measures how well h can be approximated by a hypothesis $h' \in \mathcal{H}'$. A sequence of tasks $(W_1, f_1^*), \ldots, (W_N, f_N^*)$ is called γ-separated if for each $i \in [N]$

$$d_{W_i}\left(f_i^*, \mathrm{MV}(\{f_1^*, \ldots, f_{i-1}^*\}, i-1)\right) \geq \gamma$$

and a sequence $(W_1, f_1^*), \ldots, (W_N, f_N^*)$ has γ effective dimension k if the largest γ-separated subsequence of these tasks has length k. Informally, this says that any f_i^* can be approximated by a weighted majority of some k functions $f_{i'}^*$ for $i' < i$. Likewise, the paper defines the discrepancy

$$\text{disc}_{\mathcal{H}}(W_i, W_j) = \max_{h,h' \in \mathcal{H}} |d_{W_i}(h, h') - d_{W_j}(h, h')|$$

which is a measure of task relatedness. Informally, a low discrepancy means that if h and h' approximate each other on distribution W_i, they also approximate each other on a different distribution W_j.

Given a target error of ε per task, the following algorithm is then proposed: for each task i, ask for a dataset $\mathcal{D}_i$ of samples of size n_i from W_i, where n_i is carefully calculated based on epsilon and the relatedness between tasks. If we can fit a function $f_i \in \text{MV}(\{f_1, \ldots, f_{i-1}\}, i-1)$ with small error on dataset $\mathcal{D}_i$, then set f_i as the classifier for task i. Otherwise, draw new samples from W_i and fit f_i to be the empirical minimizer from $\mathcal{H}$ on this dataset. The main theoretical result is the following. Suppose the following 4 conditions hold: (1) the γ-effective dimension is at most k (functions f_i^* can be expressed as linear combinations of k other functions from previous rounds), (2) $\text{disc}_{\mathcal{H}}(W_i, W_j) \leq r$ (task distributions are sufficiently related), (3) $\gamma \leq \varepsilon/4$, (4) $kr \leq \varepsilon/8$ (the degree of task relatedness is large enough relative to the error target). Then with probability at least $1 - \delta$, the true error of each fitted function f_i on task i is bounded by ε, and the total number of labeled examples requested by the algorithm is $\tilde{\mathcal{O}}\left(\frac{Nk + VC(\mathcal{H})k^2}{\varepsilon}\right)$. This allows some formal measure for comparing how prior information might help with learning new tasks.

2.2.2 Continual Learning In Neural Networks

Continual learning in neural networks can often be viewed as a special case of lifelong supervised learning where knowledge is represented by the evolving weights of a neural network. Suppose one is given a new dataset $\mathcal{D}_i$ for classification task i. The key problem is that naively updating the network weights with stochastic gradient descent on $\mathcal{D}_i$ tends to cause the network to perform worse on previously seen classification tasks. This phenomenon is known as Catastrophic Forgetting (CF), first coined in

McCloskey and Cohen (1989).

A body of work has grown around techniques for addressing the problem of CF. Usually, one assumes that the algorithm has a bounded capacity to remember training examples from prior tasks. The problem is to give a procedure for updating the weights of a neural network to perform well on new tasks, without degrading its performance on prior learned tasks. The literature in this space is varied. Since neural networks are not well understood theoretically, many techniques often rely on different styles of heuristics. We now give a representative sample of the core ideas behind these techniques.

Progressive Neural Networks

The idea of Progressive Neural Networks (Rusu et al., 2016) is to explicitly learn a new neural network for each task. When a new neural network is learned, it is able to use the outputs of layers of previous networks as inputs. Performance on prior tasks trivially remains the same, while the new task has the potential to take advantage of the learned representations of tasks already learned. Formally, on task $\mathcal{T}_i$ we learn weights $\{W_j\}_{j\in[L]}$ for an L layer neural network, and lateral connections $\{U_j^{i'}\}_{j\in[L],i'\in[i-1]}$, where the jth hidden layer output of network i is equal to

$$h_j^{(i)} = \max\left(0, W_j^{(i)} h_{j-1}^{(i)} + \sum_{i'<i} U_j^{(i')} h_{j-1}^{(i')}\right)$$

The obvious disadvantage of this approach is that it assumes one is told which task is being performed, and the number of parameters of the model can grow prohibitively large very quickly.

Elastic Weight Consolidation

Elastic Weight Consolidation (EWC) (Kirkpatrick et al., 2016) is an example of a technique which tries to control how quickly the weights of a neural network change

on new tasks, in the hope that this will prevent performance degradation on prior tasks. EWC proposes using a heuristic Bayesian approach to mitigating CF. Suppose two datasets $\mathcal{D}_A, \mathcal{D}_B$ drawn independently for different tasks, and suppose we model the neural network parameters $\theta \in \mathbb{R}^d$ as a random variable. Then we can write

$$\begin{aligned}
\mathbb{P}(\theta|\mathcal{D}_A \wedge \mathcal{D}_B) &= \frac{\mathbb{P}(\mathcal{D}_A \wedge \mathcal{D}_B|\theta)\mathbb{P}(\theta)}{\mathbb{P}(\mathcal{D}_A \wedge \mathcal{D}_B)} \\
&= \frac{\mathbb{P}(\mathcal{D}_A|\theta)\mathbb{P}(\mathcal{D}_B|\theta)\mathbb{P}(\theta)}{\mathbb{P}(\mathcal{D}_A)\mathbb{P}(\mathcal{D}_B)} \\
&= \frac{\frac{\mathbb{P}(\theta|\mathcal{D}_A)\mathbb{P}(\mathcal{D}_A)}{\mathbb{P}(\theta)}\mathbb{P}(\mathcal{D}_B|\theta)\mathbb{P}(\theta)}{\mathbb{P}(\mathcal{D}_A)\mathbb{P}(\mathcal{D}_B)} \\
&= \frac{\mathbb{P}(\mathcal{D}_B|\theta)\mathbb{P}(\theta|\mathcal{D}_A)}{\mathbb{P}(\mathcal{D}_B)}
\end{aligned}$$

Therefore, writing $\mathcal{D} = \mathcal{D}_A \wedge \mathcal{D}_B$, we have

$$\log \mathbb{P}(\theta|\mathcal{D}) = \log \mathbb{P}(\mathcal{D}_B|\theta) + \log \mathbb{P}(\theta|\mathcal{D}_A) - \log \mathbb{P}(\mathcal{D}_B)$$

$\log \mathbb{P}(\mathcal{D}_B|\theta)$ is the loss function for θ on dataset $\mathcal{D}_B$ and so can be optimized. The term $\log \mathbb{P}(\theta|\mathcal{D}_A)$ is not tractable to compute, so EWC assumes $\log \mathbb{P}(\theta|\mathcal{D}_A)$ can be approximated by a Gaussian distribution with mean θ^*_A (the parameters which minimize the loss on the dataset $\mathcal{D}_A$) and a diagonal precision given by the diagonal of the fisher information matrix F. If we assume this, we can maximize the likelihood $\mathbb{P}(\theta|\mathcal{D})$ on the joint data by minimizing the loss function

$$\mathcal{L}(\theta) = \mathcal{L}_B(\theta) + \sum_{j \in [d]} \frac{\lambda}{2} F_j(\theta_j - \theta^*_{A,j})^2$$

where

$$F_j = \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}_A}\left[\left(\frac{\partial}{\partial \theta_j} \log p((x,y)|\theta)\right)^2 |\theta = \theta^*_A\right]$$

$\mathcal{L}_B(\theta)$ encourages a low loss on $\mathcal{D}_B$, while $\sum_{j\in[d]} \frac{\lambda}{2} F_j(\theta_j - \theta^*_{A,j})^2$ encourages the parameters of θ to not change too quickly. We then learn the parameter $\theta^*_{A\cup B}$ for both tasks. When we receive a new task C, we set $\theta^*_A = \theta^*_{A\cup B}$, $A = A \cup B$, and $B = C$ and repeat. Note that to maintain the matrix F for all previous tasks, it suffices to estimate F individually for each dataset we encounter, and then add F to a rolling tally.

Learning Without Forgetting

Learning Without Forgetting (LwF) (Li and Hoiem, 2016) proposes learning a collection of shared parameters θ_s and task specific parameters θ_i for each task $i \in [N]$. For example, θ_s can correspond to the layers of a Convolutional Neural Network (CNN), and θ_i can correspond to a single final classification layer which can be appended to the end of the shared CNN. When given a new task dataset $\mathcal{D}_i$, LwF updates the shared parameters $\theta_{s,\text{old}}$ to $\theta_{s,\text{new}}$ and learns task specific parameters θ_i by training on a loss function which is the sum of two components. The first component is the task specific loss for task i. The second component is a distillation loss, which measures how much the outputs for the previous task networks using $\theta_{s,\text{old}}$ change on dataset $\mathcal{D}_i$ when using $\theta_{s,\text{new}}$ instead. While there is some intuition of why this approach might work in idealized cases, it is perhaps unclear why this would prevent neural networks from forgetting old tasks if e.g. the new task distribution is significantly different from old task distributions.

Incremental Classifier And Representation Learning (iCaRL)

In iCaRL (Rebuffi et al., 2017), one imagines learning a feature extractor neural network ϕ_θ and task specific weight vectors $\{w_i\}_i$ which will be used for classification over multiple classes. Each task corresponds to learning a new image class to classify. Similarly to LwF (Li and Hoiem, 2016), iCaRL uses a form of distillation loss to discourage forgetting between tasks. However, we keep a set of example training

points P_i for each task encountered so far. When given new training data $\mathcal{D}_i$ for task i (e.g. a collection of image examples from class i), we first compute target values $q_{i',j} = \frac{1}{1-\exp(w_{i'}^T \phi(x_{i',j}))}$ for the jth exemplar point $x_{i',j}$ of task i'. ϕ and the w's are then trained to minimize the loss of the sum of (a) the classification error on the new task, and (b) a distance between the $q_{i',j}$'s evaluated using either the old or new parameters of the neural network. For actual classification, on constructs centers $\mu_i = \sum_{x_{i,j} \in P_i} \phi(x_{i,j})$ for each class, and classifies x according to the nearest center of $\phi(x)$. Likewise, the exemplars for task i are greedily chosen such that $\sum_{x_{i,j} \in P_i} \phi(x_{i,j})$ most accurately represents the mean $\sum_{x_{i,j} \in D_i} \phi(x_{i,j})$ computed using the entire task data. One can interpret iCaRL as learning a latent feature space given by ϕ used for nearest neighbour classification, where we use a distillation loss on the latent feature space instead of the neural network outputs as is done in LwF.

Expert Gate

Expert Gate (Aljundi et al., 2017) proposes incrementally learning an autoencoder A_i for each incremental dataset $\mathcal{D}_i$. Given a new dataset $\mathcal{D}_{i+1}$ and a learned autoencoder A_{i+1}, a measure of task similarity between T_{i+1} and the prior tasks $T_1, \ldots, T_i$ is proposed based on the average reconstruction errors of $A_1, \ldots, A_i$ on the dataset D_{i+1}. Based on this similarity score, either fine-tuning[2] or LwF is used to construct a classifier C_{i+1} from the most related previous task. At test time, given a datapoint x, the classifier C_i whose autoencoder A_i has lowest reconstruction loss is most likely to be selected for prediction.

Generative Replay

Generative Replay (Shin et al., 2017) proposes learning a generative model of past training data, alleviating the need to save the entire training set across multiple tasks.

[2]Fine-tuning is when a neural network is initialized with weights optimized from a previous task, and then trained for a limited duration on new data.

A scholar model $H_i = (G_i, S_i)$ corresponding to tasks $1, \ldots, i$ is maintained throughout the algorithm. The algorithm needs to maintain a classifier which performs well across all the tasks (without being told which task it is performing). Here G_i is a Generative Adversarial Network (GAN) which generates samples $x \in \mathcal{X}$ from tasks $1, \ldots, i$. S_i is a solver which takes an x generated from G_i and is optimized to output the correct value. When a new dataset $\mathcal{D}_{i+1}$ for the $i+1$th task is given, samples x and their predicted labels y are drawn from G_i and S_i respectively. This creates a generated dataset corresponding to data from past tasks. G_{i+1} is then optimized on this generated data as well as the data from $\mathcal{D}_{i+1}$ to generate samples for tasks $1, \ldots, i+1$. Likewise, S_{i+1} is optimized on both the generated data and the new data $\mathcal{D}_{i+1}$. One drawback of Generative Replay is that training GANs can be unstable, so there may be a particular step i where the GAN fails to properly capture the full distribution of past data.

2.2.3 Meta Learning

Meta learning (Thrun and Pratt, 1998) is a branch of computer science which studies *how to learn* how to learn. From the perspective of lifelong learning, this means not simply accumulating new knowledge to be able to perform more tasks, but also accumulating knowledge which allows the system to learn future tasks more effectively. While there is some disagreement over whether there is a fundamental distinction between meta learning and learning in general (Vilalta and Drissi, 2002), meta learning generally creates a distinction between "long term" and "short term" learning, and asks how to improve the process of short term learning via a "long term" learning technique. There are many different approaches to meta learning. Earlier work included Stacked Generalization (Wolpert, 1992) (an ensemble learning approach), and modelling meta learning as adaptively changing the hypothesis space ("bias") a learning algorithm uses over time (Vilalta and Drissi, 2002). More recent examples include

learning an algorithm which can perform classification when only given a few number of labeled examples (few shot learning (Snell et al., 2017; Vinyals et al., 2016)), and "fast learners" to learn new tasks quickly coupled with "slow learners" to adapt to long term trends (Munkhdalai and Yu, 2017). Ultimately, it can be argued that meta learning is about constructing structured ways to adapt a learning procedure over time. We will now focus on some of the more modern meta learning techniques which have shown promise in practical applications.

Few-Shot Learning For Image Recognition

Few-Shot learning is the problem of learning how to construct a classifier after being given only a very limited amount of training data. In the case of image recognition, "Prototypical Networks for Few-shot Learning" (Prototypical Networks) (Snell et al., 2017) and "Matching Networks for One Shot Learning" (Matching Networks) (Vinyals et al., 2016) are two influential papers in this area. Formally, suppose one is given a small set $\mathcal{D}$ of labeled data from a subset of classes $C \subset [K]$ (e.g. classes of images). The task is to "meta learn" an algorithm A which takes as input $\mathcal{D}$, and then performs well at classifying images from classes in C. Suppose we sample a subset of image classes C, a small set of training data $\mathcal{D}$ for these classes, and a large set of test data $\mathcal{D}_{\text{test}}$. Let $A(\mathcal{D})(x)$ be the classification prediction of x when trained on data $\mathcal{D}$. Then the objective is to pick the algorithm A to minimize the expectation of

$$\sum_{(x,y)\in\mathcal{D}_{\text{test}}} \mathcal{L}(A(\mathcal{D})(x), y)$$

Both papers take a similar approach by parameterizing A as a form of smooth nearest neighbors classifier with embedding f_θ, where the task is to learn the embedding parameters θ. This is similar to the idea given in Thrun (1996). While not directly related to few shot learning, there have been similar ideas for learning embeddings in order to improve the learning process. For example, Raina et al. (2007)

proposes learning a sparse embedding of unlabeled image data in order to better represent labeled image data. These sparse representations of labeled data are then fed into a learning algorithm.

Other Neural Network Approaches To Meta Learning

There are a number of conceptual approaches which have been taken to designing neural networks for meta-learning problems. Neural Turing Machines (Santoro et al., 2016) are neural networks augmented with memory, trained to learn few-shot learning tasks such as Omniglot image classification. "Meta Networks" (Munkhdalai and Yu, 2017) propose a neural network architecture which consists of fast learners which are updated quickly, and slow learners which are updated less frequently. Similarly, "Fast Reinforcement Learning via Slow Reinforcement Learning" (Duan et al., 2016) trains a "slow" Recurrent Neural Network (RNN) on a changing sequence of Markov Decision Processes (MDP), and shows that by training a RNN on this sequence, the network can learn to quickly adapt ("learn fast") when the MDP changes. Both papers emphasize how fast adaptive behavior can automatically result on shorter time scales by choosing training objectives which span longer periods of time. "Learning to Remember Rare Events" (Kaiser et al., 2017) argues that an important part of lifelong few-shot learning is being able to remember rare events, and proposes a nearest neighbor memory module architecture for this purpose. This suggestion also hints at the idea that one may need to take a more nuanced approach than simply optimizing the mean squared error of a loss function when designing lifelong learning systems.

Gradient Specific Approaches To Training Neural Networks In A Meta Learning Context

One challenge of meta learning in general is the question of how to search for algorithms which perform well on long timescale meta learning tasks. This is particularly

true for optimization methods like stochastic gradient descent: a meta-learning algorithm might make a small change to a base learning algorithm, but in order to compute the gradient of this change, one may need to perform a gradient computation over a very long period of time-steps. It is unclear how meaningful gradients are in these settings (where small changes to a network parameter might presumably have large effects), or how to compute gradients in this setting efficiently. The most direct response to this concern is "Gradient-based Hyperparameter Optimization through Reversible Learning" (Maclaurin et al., 2015), which propose a technique for learning hyperparameters for a neural network via stochastic gradient descent with a reversible gradient. The key idea is that, in this setting, one only needs to keep a single time step (gradients of the neural network) in memory in order to be able to compute the gradient with respect to the hyperparameters."Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks" (MAML) (Finn et al., 2017) takes the approach of learning parameters θ for a neural network, such that when new data $\mathcal{D}$ for a task is received, the parameter $\theta' = \text{Update}(\theta, \mathcal{D})$ performs well on the new task, where Update consists of a few (1-4) steps of stochastic gradient descent. This approach requires computing the gradient of the gradient update step, which requires computing the Hessian. MAML also proposes a linear approximation to avoid computing second order derivatives. "Learning to learn by gradient descent by gradient descent" (Andrychowicz et al., 2016) proposes searching for an RNN which takes as input the gradient history of a neural network parameter, and then outputs a parameter update. The RNN is optimized by stochastic gradient descent (with the objective of producing low final error of the classifier), and is made tractable by making an assumption about partial derivatives of the classifier and RNN.

Evolution Strategies For Meta Learning

Given the complications of training a meta neural network with stochastic gradient descent, it is useful to look at simpler alternatives which have been tried. "AutoML-

Zero: Evolving Machine Learning Algorithms From Scratch" (Real et al., 2020) directly evolves python-like programs to perform image classification, where the programs consist of a sequence of basic instructions. The optimization procedure is a search for simple programs which, after given a small training set, empirically perform well on a test set. The final evolved programs are similar to convolutional neural networks using stochastic gradient descent, suggesting that vanilla evolution on programs might be a viable approach to some problems.

"Evolution Strategies as a Scalable Alternative to Reinforcement Learning" (ES) (Salimans et al., 2017) considers an interpretation of an evolution optimization procedure where the problem is to find a parameter θ to maximize the expected reward $\mathbb{E}_{\varepsilon \sim N(0,1)} F(\theta + \sigma\varepsilon)$, where $N(0,1)$ is a multivariate standard normal distribution, for some function F. The interpretation is that $\theta + \sigma\varepsilon$ is a parameter from the population distribution centered around θ. They propose optimizing this objective via

$$\nabla_\theta \mathop{\mathbb{E}}_{\varepsilon \sim N(0,1)} F(\theta + \sigma\varepsilon) = \frac{1}{\sigma} \mathop{\mathbb{E}}_{\varepsilon \sim N(0,1)} F(\theta + \sigma\varepsilon)\varepsilon$$

This leads to the following optimization algorithm (natural gradient descent):

1. Compute samples $F_i = F(\theta_t + \varepsilon_i)$ for $i = 1, \ldots, n$.
2. Update $\theta_{t+1} = \theta_t + \alpha \frac{1}{n\sigma} \sum_{i=1}^{n} F_i \varepsilon_i$

One advantage of using random samples is that the memory complexity required only scales with the number of random seeds which must be remembered. When sampling in parallel, this means that only the F_i's need to be sent across the network (and the ε_i's can be deduced by a shared random seed). The paper found that, using the same deep architectures for playing Atari, computation time was quicker (no back-propagation needed to be computed) and performance mixed (roughly half the results were better, half were worse, than baselines). Similarly, "Simple random

search provides a competitive approach to reinforcement learning" (Mania et al., 2018) shows that linear and radial basis function policies, using policy search with natural gradient, can perform as well as deep neural networks (using Trust Region Policy Optimization (Schulman et al., 2015)) for certain locomotion tasks. Other differences from ES included "unbiased" direction sampling (e.g. sampling $F(\theta+\varepsilon)$ and $F(\theta-\varepsilon)$ in pairs), occasionally keeping some linear coefficients fixed from being updated, and normalizing means and covariances of inputs. State of the art performance (at time of the technique's publication) for benchmark MuJoCo locomotion tasks is achieved, and the method is significantly faster than competing methods.

In contrast to using natural policy gradient, "Deep Neuroevolution" (Such et al., 2017) considers pure evolution on neural networks, using only mutation (adding noise to parameters) and no crossover. It is shown that when this is applied to learning reinforcement learning policies for some atari games, performance can be better or worse than other standard RL gradient methods for this domain, and on average the performance is comparable with other RL methods. Natural selection can perform better where traditional gradient based RL methods have failed, usually when the reward requires a long time horizon with little greedy signaling. Similar to ES, the update method is exploited. In this case, instead of remembering the entire parameter of the neural network, the random seed choices which led to the network are remembered instead, so memory grows linearly in the number of evolution steps. It was noted that natural selection generally out performed random search, suggesting that making random mutations in parameter space for neural networks leads to evolution paths where this is better than pure random search.

2.2.4 Unsupervised And Semi-Supervised Lifelong Learning

The majority of work in unsupervised learning occurs in language modelling. For example, in lifelong topic modelling (Blei et al., 2003; Wang et al., 2016; Chen and

Liu, 2014), one is given an online sequence of text documents with the task of learning the topics which constitute them. As an example, Chen and Liu (2014) introduce Lifelong Topic Model: a method for continually growing a collection of topics, and using prior learned topics to help infer new topics. Another language based lifelong learning system is NELL (Never-Ending Language Learner) (Carlson et al., 2010; Mitchell et al., 2015). NELL is a semi-supervised lifelong learner which has been continuously reading the web since 2010, in an effort to better "understand" text and accumulate knowledge. It is designed with a systems-based approach to LL, in which a collection of components (e.g. knowledge base, image classifier, text context patterns) are combined together. The system is optimized to perform a variety of tasks, some of which are semi-supervised, and maintains a knowledge database of known relations between objects. Another text based knowledge learning system is given in Mazumder et al. (2018). Here, given text generated by a user interacting with a chatbot, the algorithm learns to infer triples (s, r, t) which indicate whether s and t can be linked by relation r. Many of the techniques used in unsupervised language modelling are quite tailored to e.g. modelling documents as a bag of words, or have very specific designs which are not necessarily applicable in other contexts. We therefore choose not to review them in great detail.

A separate class of semi-supervised lifelong learning methods can be found in open world learning (Bendale and Boult, 2015, 2016; Fei et al., 2016). Open world learning considers the problem where one needs to continually learn to classify new classes, while also being able to determine when one is given an input from an unseen class. There are two main themes of ideas in this area. The first is to learn an N vs 1 classifier for each of the N classes. Here, the ith classifier is optimized to output 1 if an input x comes from class i, and 0 otherwise. Given a new input x, one can then reject x as coming from an unseen class if none of the classifiers output a value above a certain threshold. For example, this approach is used by DOC (Deep

Open Classification) (Shu et al., 2017b) to classify text documents, and later in open world image classification (Shu et al., 2018). The second key theme is *open space risk* (Scheirer et al., 2013). Given binary classifier f, one can consider the region of space $R = f^{-1}(1) \subset \mathbb{R}^d$ in which f classifies an example $x \in \mathbb{R}^d$ as positive. Informally speaking, one would like R to not be "unnecessarily large": just large enough to capture positive examples, but small enough such that unseen negative examples do not lie in the region R. Of course, since these negative examples have not yet been seen, this is often an inherently unsupervised learning task. For example, Fei et al. (2016) give a modified support vector machine which can control the size of the region R. To estimate how large the region R should be, they use an unsupervised clustering technique called center-based similarity space learning. Likewise, DOC calibrates its open risk by changing the output threshold at which an N vs 1 classifier is determined to predict a detection of its class. This is done using a heuristic statistical estimation technique on the positive samples of that class.

2.2.5 Lifelong Reinforcement Learning

Reinforcement learning (see Sutton and Barto (2018)) (RL) is the study of finding policies which maximize a reward function while interacting with an environment. A common way to model the problem is with a Markov Decision Process (MDP). For example, a fully observable MDP consists of a tuple $(\mathcal{S}, \mathcal{A}, \mathbb{P}, R, \lambda)$ where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions available to the policy, $\mathbb{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is a state transition probability function, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, and $\lambda \in [0, 1)$ is a reward discount factor. The environment consists of time steps $t = 1, 2, \ldots$. The initial state $s_1 \in \mathcal{S}$ is sampled from some prior distribution. At time step t, an agent observes state s_t, and must pick an action $a_t \in \mathcal{A}$. The agent then receives reward $r_t = R(s_t, a_t)$, and the environment transitions to state s_{t+1} with probability $\mathbb{P}(s_t, a_t, s_{t+1})$. The goal is to find a policy $\pi : \mathcal{S} \to \mathcal{A}$ for the agent to follow such that

the expected value of the discounted rewards $\sum_{t=1}^{\infty} \gamma^{t-1} r_t$ is maximized. Alternative models for reinforcement learning have also been considered. For example, one might assume the MDP is not fully observable, so the agent does not observe the full state s_t at time t but instead an observation $o_t = f(s_t)$ which is a function of s_t.

Lifelong reinforcement learning is the problem of continually updating the agent policy π as the environment is learned, so that the reward prospects continually improve over time. It was first proposed in Thrun and Mitchell (1995), which discusses the importance of lifelong learning in RL and various ways this could be achieved. Early work often focused on exploring how intuitions about continual learning could be incorporated into neural networks. For example, Tanaka and Yamamura (1997) proposed learning new environments by initializing a neural network with the average of weights of neural networks used to solve previous RL tasks, weighted by the degree to which the components of the weights change across tasks. Ring (1998) proposed CHILD, which used a hierarchical neural network architecture to attempt to use knowledge from easy tasks to learn harder tasks. Tessler et al. (2017) is a more modern example of this approach, where a hierarchical neural network architecture combined with knowledge distillation (see Subsection 2.2.2) is used to continually acquire skills in Minecraft. Mendez et al. (2022) explore how compositional neural network architectures can be used to improve learning future tasks.

On the more theoretical side, lifelong RL is often studied in the context of MDPs or as an optimization problem where principled simplifying assumptions are made. An example of the latter is PG-ELLA (Bou-Ammar et al., 2014), an adaptation of ELLA (see Subsection 2.2.1) for RL. Like ELLA, PG-ELLA assumes knowledge for a task can be represented as a sparse linear combination of vectors from a shared library L. However, instead of minimizing a classification loss, it minimizes a policy gradient loss (Sutton et al., 1999), using the same Taylor series approximation as ELLA to make this optimization tractable in the lifelong learning setting. In the

context of MDPs, Brunskill and Li (2014) study how learning options (temporally extended action sequences) can be used to speed up learning in new tasks. Wilson et al. (2007) introduce the MTRL algorithm, which learns a Bayesian hierarchical mixture of MDPs for different tasks, and groups MDPs into the same class if they are similar. Given a new task, one can search for similar MDPs and use this as a basis for learning the new task more quickly. Wang et al. (2022) introduce a similar style of approach, where a mixture of environment models is learned using the EM algorithm, and environments are clustered based on similarity. In these kinds of approaches, one needs to decide what it means for two MDPs to be similar. A simple choice is to say two MDPs are similar if the models have similar state transition distributions, however there are other reasonable choices. For example, Lecarpentier et al. (2021) study a metric for MDPs where closeness implies the have the same optimal value functions, allowing one to give transfer learning algorithms with formal performance guarantees. Reusing models of previously learned environments can be contrasted with reusing policies of previously learned environments. For example, Fernández and Veloso (2013) propose a system for maintaining a library of previously learned policies, and switching between a previously learned policy and a new policy based on the current performance of each policy.

Another branch of work studies cross domain transfer learning (Taylor et al., 2007; Bou-Ammar et al., 2015; Isele et al., 2016). Here, one tries to learn a correspondence from a new environment to a previous learned environment, so that one can re-use knowledge of how to behave in previous learned environments. For example, Bou-Ammar et al. (2015) propose learning a mapping between trajectories between two different environments using Unsupervised Manifold Alignment (Wang and Mahadevan, 2009), an unsupervised learning technique for learning a correspondence between two datasets based on geometric priors.

As in the supervised LL case, there are a variety of different approaches to rein-

forcement LL. Approaches vary by both their level of formality, as well as the specific way knowledge is modelled.

2.2.6 Other Related Areas

Lifelong learning is an especially broad field. We conclude this section by briefly mentioning a few related fields connected to both lifelong learning and bounded rationality.

Neurosymbolic AI

Neurosymbolic AI (see Garcez and Lamb (2023)) attempts to merge the strengths of logical reasoning systems and neural networks into a single framework. Logical reasoning systems have the advantage of being able to use deduction to infer knowledge. On the other hand, inductive learning systems like neural networks offer a degree of flexibility due to their ability to fit on observed data, but may not generalize to new data in the same way as logical reasoning systems. A concrete example of a neurosymbolic system is Logic Tensor Networks (LTNs) (Badreddine et al., 2022). LTNs define a form of logic called "Real Logic", where logical relations have degrees of truth which can be parametrically represented by a neural network. Given a logical specification of a problem, one can then attempt to learn these truth functions from the data. Within the context of lifelong learning, neurosymbolic AI is one approach to representing knowledge for an incremental learner. Within the context of bounded rationality, the logical reasoning component of neurosymbolic AI is a fairly explicit reasoning system whose limitations can be understood analytically.

Learning In Games

Learning in games (Fudenberg and Levine, 1998) studies how players might incrementally learn to play better while playing a game. In these settings, one does not assume the game has already reached Nash equilibrium. Instead, one studies how

players might adapt their behaviour as they learn more about their opponents while playing the game. An early example of such a model is fictitious play (Robinson, 1951). In a two player game, each player assumes that their opponent chooses strategies from an unknown stationary distribution. When a player observes a choice from their opponent, they update their beliefs about the stationary distribution of their opponent's choices using Bayes' rule. They then play optimally given this posterior belief. Other work (Bush and Mosteller, 1955) considers a reinforcement learning model. Here one studies how players behave if they simply react to positive or negative stimulus without having any model of the game or their opponents. In the context of lifelong learning, learning in games can be seen as a way of studying continual learning in the more classical setting of strategic games.

Machine Teaching

Machine teaching (see Zhu et al. (2018)) studies how the difficulty of learning changes when an algorithm has access to a teacher which can assist the algorithm. Different models of assistance have been considered. For the sake of concreteness, suppose the problem is to learn a binary classifier $f : [0,1] \to \{0,1\}$ where $f(x) = 1$ iff $x \geq \theta^*$. If a learning algorithm were given uniformly random training samples in $[0,1]$, then it would require roughly $\Omega(\varepsilon^{-1})$ samples to estimate θ^* within an additive error of ε. On the other hand, a teacher which knows θ^* could make this learning task significantly easier. If the learning algorithm is allowed to query points $(x, f(x))$ to the teacher in the context of active learning (see Settles (2009)), then it can use binary search to find θ^* within an additive error of ε in only $\mathcal{O}(\log(\varepsilon^{-1}))$ queries. A different approach would be for the teacher to give the learning algorithm two training samples $(\theta^* - \varepsilon, 0), (\theta^* + \varepsilon, 1)$, from which the learning algorithm could immediately deduce a good estimate for θ^*. Towards formalizing this idea, the teaching dimension (Goldman and Kearns, 1991) is a way of quantifying the minimum number of samples required to solve a particular learning problem if the teacher is allowed to choose which samples

to show the algorithm. In the context of lifelong learning, machine teaching studies a particular setting in which continual learning might occur. Because it is difficult to make strong assumptions about the future in the context of lifelong learning, models which allow access to a well-behaved teacher may play an especially important role in making certain learning problems more tractable in the lifelong learning setting.

Chapter 3

Precomputation, Time And Memory

3.1 Introduction

While there is a firm theoretical understanding of how perfectly rational agents should play games in equilibrium, understanding the setting where agents have bounded computational resources appears to be more challenging. For example, in theory, an agent playing a mini-max game such as chess always has an optimal move it can make. In practice, it may take a very long time to compute an optimal move, and it can be unclear how a rational agent should (or would) behave when it only has a finite amount of time to decide on a move.

Consider an extensive form game played between two players, where each player has a finite amount of total time during the game to make their moves. One simple yet under-explored property of these games in the time-constrained setting is the following. Before the game, both players can typically spend a large amount of time practicing and preparing, and this time is typically much larger than the amount of time spent playing the game. During the game, if players encounter situations they have prepared for in advance, they are able to play very strong moves quickly from their preparation memory. Otherwise, they have to use their time-limited budget to compute a new move. This simple structure leads to interesting tradeoffs. For

example, if player 1 plays very deterministically, it will be easy for player 2 to prepare (precompute) strong moves against player 1 using the much larger compute resources available before the match. On the other hand, if player 1 plays more randomly to try and make the future of the game harder to predict, player 1 necessarily needs to play "optimal" moves less frequently. Modeling just this precomputation tradeoff leads to surprisingly rich behavior in time constrained games. For example, in chess (which will be used as a running example throughout this chapter), human players often:

1. Prepare against other players by studying certain opening lines they expect to occur in a game based on the playing styles of the other player.
2. Play the first few moves of the game quickly from an "opening book" of memorized moves.
3. Intentionally randomize their strategies to make it harder for their opponent to prepare against them. In practice, this can done by playing e.g. the 5th or 6th best move recommendation of a chess engine during an opening sequence of moves.

In contrast, many modern chess engines do not explicitly exhibit some of the behavior described above; for example, they may deterministically play the "best move" found within an allotted time, mirroring the idea that in the computationally unbounded setting, there is always an optimal move to play for a mini-max game like chess. In contrast, this paper will show that in the computationally bounded setting where players can precompute, it is perhaps more useful to think in terms of the "best distribution of moves" to play.

Prior work in bounded rationality is reviewed in Chapter 2. In the particular context of precomputation, Simon and Schaeffer (1992) give an account of how professional human chess players rely on a memory of previously encountered board positions while playing. This work can therefore be seen as a concrete mathematical

model of some of the observations made in Simon and Schaeffer (1992). This work can also be contrasted with other models of bounded rationality which penalize players for playing more computationally demanding strategies, such as in Ben-Sasson et al. (2006). In this work, players are penalized for how many moves they memorize in an extensive form game, which leads to a specific game structure which we show has analytical guarantees and polynomial-time algorithms. In contrast, the work of Ben-Sasson et al. (2006) considers normal form games with generic cost functions which do not specifically model the structure of precomputation. More broadly, this work fits within the context of constructing models of bounded rationality for understanding specific aspects of computationally bounded agents (e.g. Pearl (1980); Dow (1991); Balcan et al. (2018); Dick et al. (2020)).

3.1.1 Contributions

We first give a novel and flexible formalism for modeling the precomputation aspect of two player, zero sum, perfect information extensive form games played under time constraints. Next, we give theoretical results establishing the importance of randomness of play in the computationally constrained setting. We show that one can efficiently compute how exploitable a fixed strategy is to precomputation, and the equilibrium precomputation strategy between two players. We then empirically demonstrate how useful precomputation can be in practical contexts, by exploring how susceptible the chess playing algorithm Stockfish is to precomputation as a function of randomization.

3.2 Definitions

3.2.1 Background

We consider two-player, zero sum, perfect information extensive form games, where the players alternate in turns. In order to provide maximum flexibility of the model,

we use standard terminology for game playing which makes no explicit reference to computational constraints. However, we implicitly think of player policies σ_i as being implemented by some computationally constrained algorithm (e.g. alpha-beta search), and the game being played under some time limit. We will give some concrete examples shortly. A textbook reference on game theory can be found in Maschler et al. (2020). We begin by reviewing some standard terminology.

A two-player game consists of a set of **histories** H, a subset of **terminal histories** $Z \subset H$, a **utility function** $u : Z \to [0, 1]$ defined on terminal histories $h \in Z$, finite sets of possible actions $A(h)$ for each history $h \in H \setminus Z$, and a **transition function** π which takes as input a history $h \in H \setminus Z$, an action $a \in A(h)$, and returns a unique history $h' = \pi(h, a)$.

A **behavioral strategy** (policy) for player $i \in \{1, 2\}$ consists of a function σ_i defined on $H \setminus Z$, where $\sigma_i(h) \in \Delta(A(h))$[1], i.e. σ_i maps histories to distributions over actions, where $\sigma_i(h)(a)$ is the probability that player i plays action $a \in A(h)$ when at history h. A **player function** $P : H \setminus Z \to \{1, 2, c\}$ indicates which player's turn it is to choose an action at history h. If $P(h) = c$, then h corresponds to a placeholder chance node (i.e. chance player) and an action $a \in A(h)$ is chosen according to a fixed probability distribution $\sigma_c(h)$.

For notational reasons, we associate each history $h \in H$ with the unique ordered sequence $(a_1, \ldots, a_k)$ of actions which must be played to reach h beginning from the starting history $\emptyset \in H$, and we say h has length $|h| = k$. For $i \in [k]$ we denote $h_i = a_i$, and denote by $h_{:i}$ the history reached by playing $a_1, \ldots, a_i$, so in particular we have $h_{:i} = \pi(h_{:i-1}, a_i)$. We say $h \in H$ is a prefix of h' if $\exists i$ s.t. $h = h'_{:i}$, and we write this as $h \leq h'$. The **successor function** $\text{Succ}(h) : H \setminus Z \to 2^H$ (here 2^H is the power set of H) maps a history h to the set of histories $H' \subset H$ where player $P(h)$ next gets to have a turn, or the following terminal histories (whichever occurs first).

[1]Here Δ is the probability simplex.

Formally, let $X(h', h)$ be true if $P(h') = P(h)$ or $h' \in Z$, and false otherwise. Then $\text{Succ}(h) = \{h' \in H | X(h', h) \wedge h' > h \wedge (\nexists h'' \in H \text{ s.t. } P(h'') = P(h) \wedge h < h'' < h')\}$. Unless explicitly stated, we will consider games where **player 1 and 2 alternate in turns** with no chance nodes, i.e. $P(h) = 1$ if $|h|$ is even, $P(h) = 2$ otherwise.

Given a **policy profile** $\sigma = (\sigma_1, \sigma_2)$ for both players, the probability of history h' occurring when starting from history $h \leq h'$ is therefore

$$\pi^\sigma(h, h') = \prod_{i=|h|}^{|h'|-1} \sigma_{P(h'_{:i})}(h'_{:i})(h'_{i+1})$$

and 0 when $h \not\leq h'$. We define the probability of history h as $\pi^\sigma(h) = \pi^\sigma(\emptyset, h)$. We say that the expected value of the game for player 1 when starting from history $h \in H$ is $u_1^\sigma(h) := \sum_{h' \in Z} \pi^\sigma(h, h') u(h')$, and $u_2^\sigma(h) := 1 - u_1^\sigma(h)$ for player 2. The expected value of the game for player 1 is $u_1^\sigma := u_1^\sigma(\emptyset)$.

3.2.2 Algorithmic Strategies And Precomputation

Precomputation strategies are intended to capture the idea that, before a game, we can study some subset $S \subset H \setminus Z$ of game histories and plan in advance what to play for these histories. For example, in the case of chess, we could look at some board positions $S \subset H \setminus Z$, and see which moves a powerful chess engine σ_{pre} would recommend in these positions after running for an hour. In the actual game, if we ever end up in a position $h \in S$, we can immediately play $\sigma_{\text{pre}}(h)$ from a lookup table (without spending any time computing moves). Otherwise, if $h \notin S$, we can just play our usual policy $\sigma_i(h)$ (where we would need to spend compute time during the game).

Definition 3.1. *Given policies σ_{pre}, σ_i, a $(\sigma_{pre}, \sigma_i, B)$ **precomputation strategy** with memory budget B is a policy $\tilde{\sigma}_i$ such that:*

1. *There exists a memorization set $S \subset H \setminus Z$, where $|S| \leq B$ and S is prefix closed (i.e. if $h \in S$ and $h' < h, P(h') = P(h) \implies h' \in S$).*

2. $h \in S \implies \tilde{\sigma}_i(h) = \sigma_{pre}(h)$.

3. $h \notin S \implies \tilde{\sigma}_i(h) = \sigma_i(h)$.

For convenience, we will always think of modeling policies σ_{pre} as taking 0 time per move. One justification for considering prefix closed memorization sets is that if a player chooses to memorize history $h \in H$, it is natural for them to have also considered the histories $h' < h$ leading up to h. We now consider the following meta-game: before a match between player 1 and player 2, which takes place in a time-constrained setting, each player i can spend time preparing for the match by specially choosing a prefix-closed subset of histories $S \subset H \setminus Z$, and constructing a precomputation strategy $\tilde{\sigma}_i$ which plays strong moves according to some policy σ_{pre} when in this set. We assume that there is some limiting factor on how many moves each player can memorize for a particular game. In practice, this may be enforced by e.g. a maximum memory budget B, or some limited capacity to memorize moves. We model this by penalizing each player by a linear factor depending on the size of the memorization set of their precomputation strategy. The theory in Section 3 is relatively robust to other choices of penalty functions, but we found the linear penalty led to the most efficient algorithms in Section 4.[2] For more concrete intuition of the formalism one can consider chess: a player can specially prepare a list of opening moves before a tournament, but their capacity to memorize chess lines for a particular match is limited, and their choice invariably depends on the opening move choices prepared for by their opponent as well. This results in an evolving "meta game" of the best opening lines to play.

Definition 3.2. *(The meta-precomputation game) Let* $Pre(\sigma_{pre}, \sigma_i)$ *denote the set of all* $(\sigma_{pre}, \sigma_i, K)$ *precomputation strategies for all* $K \in \mathbb{N}$*, and denote* $\Delta(Pre(\sigma_{pre}, \sigma_i))$ *by the set of all mixed strategies on* $Pre(\sigma_{pre}, \sigma_i)$*. For* $\tilde{\sigma} \in Pre(\sigma_{pre}, \sigma_i)$*, let* $|\tilde{\sigma}|$ *denote*

[2]However, one can still find polynomial-time algorithms for other natural penalty functions such as a hard limit on the memorization set size.

the size of the memorization set used by precomputation strategy $\tilde{\sigma}$. Consider a meta-game where each player i instantaneously chooses a mixed precomputation strategy $\Delta\tilde{\sigma}_i \in \Delta(Pre(\sigma_{pre}, \sigma_i))$, and the outcome utility of the game is

$$\tilde{u}^{(\Delta\tilde{\sigma}_1, \Delta\tilde{\sigma}_2)} := \tilde{u}_1^{(\Delta\tilde{\sigma}_1, \Delta\tilde{\sigma}_2)} = \mathop{\mathbb{E}}_{\tilde{\sigma}_1 \sim \Delta\tilde{\sigma}_1, \tilde{\sigma}_2 \sim \Delta\tilde{\sigma}_2} \left[u_1^{(\tilde{\sigma}_1, \tilde{\sigma}_2)} - \lambda_1 |\tilde{\sigma}_1| + \lambda_2 |\tilde{\sigma}_2| \right]$$

And likewise $\tilde{u}_2 = 1 - \tilde{u}_1$. For $\lambda_1, \lambda_2 \in \mathbb{R}^+$, $\Delta\tilde{\sigma}_i \in \Delta(Pre(\sigma_i, \sigma_{pre}))$ for $i \in \{1, 2\}$ is an ε-Nash equilibrium if

$$\begin{aligned} \tilde{u}^{(\Delta\tilde{\sigma}_1, \Delta\tilde{\sigma}_2)} + \varepsilon &\geq \sup_{\Delta\tilde{\sigma}_1' \in \Delta(Pre(\sigma_{pre}, \sigma_1))} \tilde{u}^{(\Delta\tilde{\sigma}_1', \Delta\tilde{\sigma}_2)} \\ \tilde{u}^{(\Delta\tilde{\sigma}_1, \Delta\tilde{\sigma}_2)} - \varepsilon &\leq \inf_{\Delta\tilde{\sigma}_2' \in \Delta(Pre(\sigma_{pre}, \sigma_2))} \tilde{u}^{(\Delta\tilde{\sigma}_1, \Delta\tilde{\sigma}_2')} \end{aligned}$$

When this is an $\varepsilon = 0$ Nash equilibrium, we omit the ε and say that $v_{Nash} = \tilde{u}^{(\Delta\tilde{\sigma}_1, \Delta\tilde{\sigma}_2)}$ is a Nash equilibrium value of the game.

Kuhn's theorem is a well known result which states that in any extensive form game where player i has perfect recall, any mixed strategy $\Delta\tilde{\sigma}_i$ for player i has an equivalent behaviour strategy $(\Delta\tilde{\sigma}_i)^b$. Here, equivalent means that the probability of reaching any history $h \in H$ is the same regardless of whether player i uses the mixed strategy $\Delta\tilde{\sigma}_i$ or behaviour strategy $(\Delta\tilde{\sigma}_i)^b$, for any fixed opponent strategies. For this reason, we will implicitly associate the mixed strategy $\Delta\tilde{\sigma}_i$ with its equivalent behaviour strategy when the context is clear. A derivation of Kuhn's theorem is given in Chapter 6 of Maschler et al. (2020).

3.2.3 Examples

We now give two explicit examples of how we might model games with computational constraints in this framework:

Example 3.1

Chess with a time limit per move. Suppose we are playing chess, so $h \in H$ corresponds to the prior moves made by each player, $A(h)$ contains all legal moves for player $P(h)$ in state h, and $u_1(z) = 1$ if black is checkmated for $z \in Z$, 0 if white is checkmated by black, and $\frac{1}{2}$ for a draw. We think of σ_1, σ_2 as chess playing algorithms which each have 1 second per move (concretely, $\sigma_1, \sigma_2 =$ Stockfish(1), Stockfish[3] with a time limit of 1 second per move). If $\tilde{\sigma}_1$ is a $(\sigma_{\text{pre}}, \sigma_1, B)$ precomputation strategy for white with memorization set S, and σ_{pre} is a chess playing algorithm which spends 1 hour precomputing per move (concretely, Stockfish(3600)), then when $h \in S$, $\tilde{\sigma}_1(h)$ plays Stockfish(3600)(h) instantaneously; otherwise $\tilde{\sigma}_1(h)$ runs Stockfish(1) on board state h.

Example 3.2

Chess with a total time limit per game. Consider the previous example, except that instead of having a time limit per move, we now have a time limit of 1 hour for the entire game. Move histories $h \in H$ are now augmented to include the time each player took to make their corresponding moves, and the terminal set Z now includes states where players have run out of time (in which case the opposing player wins).

Note that allowing the time an algorithm σ_1 takes to depend on the entire history of moves h (instead of just the chess board state) allows us to capture the behavior of algorithms whose computation depends on what they computed in the past; for example, chess playing algorithms often keep a lookup table of moves and positions they have considered during prior moves, in order to speed up future computations.

[3]Stockfish is a popular open source chess playing algorithm.

3.3 When Do Good Precomputation Strategies Exist?

In this section, we will study how susceptible a fixed policy σ_i is to precomputation as a function of the randomness of the policy. For simplicity, we will make the definitions from the perspective of player 1 (but the analogous statements hold for player 2). To relate precomputation strategies to randomness of play, we show how to explicitly construct a precomputation strategy whose size depends on how randomly σ_2 plays with respect to σ_{pre}. This construction then serves as a lower bound on the quality of the best precomputation strategy.

The first key observation we make is the following: suppose the game reaches a state $h \in H$ such that player 1 now has significant advantage against player 2. Then it should be the case that player 1 can play their normal strategy σ_1 against σ_2, without any precomputation, and convert this game to a win. Therefore, one way of constructing a precomputation strategy is the following: have player 1 play precomputed moves from σ_{pre} against σ_2 until a significant advantage (of value v) is gained, and then continue to play normal moves from σ_1 afterwards. If σ_{pre} is a much stronger policy than σ_2, then we expect to not need to play too many moves until an advantage is reached.

We can make this idea more intuitive by considering the domain of chess. Imagine that two humans are playing chess, but that player 1 is allowed to use their smartphone to get move recommendations from a strong chess engine σ_{pre}. A very natural strategy for player 1 to follow is to start the game by copying the moves recommended by the chess engine. Because the chess engine is much stronger than player 2, we expect that after some number of moves the chess board will be in a state where player 1 has a significant advantage. For example, player 1 might have significantly more pieces on the board than player 2. From such an advantageous position, player 1 no longer needs the help of σ_{pre} to guarantee they will win the match. They can just

play their normal strategy σ_1 for the remainder of the game. For example, if player 1 has significantly more pieces on the board, a common winning strategy is simply to aggressively trade off pieces until player 1 is left with e.g. a single minor piece and pawns and player 2 has no pieces remaining, in which case playing for a win is fairly straightforward.

We now formalize the idea of copying σ_{pre} until a winning position is reached with the following definition:

Definition 3.3. *Given policy profile $\sigma = (\sigma_1, \sigma_2)$, a policy σ_{pre} and $v \in [0,1]$, the set histories where player 1 first gets an advantage $\geq v$ is*

$$\begin{aligned} S^\sigma(v) := &\{h \in H | (P(h) = 1 \vee h \in Z) \\ &\wedge u_1^\sigma(h) \geq v \\ &\wedge \nexists h' < h \text{ s.t. } u_1^\sigma(h') \geq v \wedge P(h') = 1\} \end{aligned}$$

We define a distribution $\mathcal{S}^{\sigma,\sigma_{pre}}(v)$ on $S^\sigma(v)$ via

$$\Pr_{\mathcal{S}^{\sigma,\sigma_{pre}}(v)}[h] := \frac{\pi^{(\sigma_{pre},\sigma_2)}(h)}{P_{norm}(v)}$$

where the normalizing constant

$$P_{norm}(v) = \sum_{h \in S^\sigma(v)} \pi^{(\sigma_{pre},\sigma_2)}(h)$$

is the probability of reaching any $h \in S^\sigma(v)$ when σ_{pre} plays against σ_2.

Example 3.3

For intuition, if σ_{pre} is a much stronger policy than σ_2, we expect $P_{\text{norm}}(v)$ to be close to 1. In the running example of chess, if $\sigma_{\text{pre}} = \text{Stockfish}(1000)$ and $\sigma_1, \sigma_2 = \text{Stockfish}(1)$, we have

$$P_{\text{norm}}(v) \geq P[\text{Stockfish(1000) as white checkmates Stockfish(1) as black}]$$

for all $v \leq 1$, because $u_1^{\sigma_1,\sigma_2}(h) = 1$ in every terminal history h where white has checkmated black.

Suppose we follow the strategy outlined above. How many histories do we need to memorize in order to be able to implement this strategy? One way to think about this question is to consider the entire game tree, and imagine σ_{pre} playing against σ_2. Now focus on $S^\sigma(v)$, the inner "boundary" of the tree where player 1 first gets an advantage of at least v against player 2. In the worst case, we need to memorize σ_{pre} on the entire subtree above this boundary. However we can do better by considering the probability of reaching any particular history $h \in S^\sigma(v)$ when σ_{pre} plays against σ_2, where $\mathcal{S}^{\sigma,\sigma_{\text{pre}}}(v)$ is the probability distribution over $S^\sigma(v)$ as defined in Definition 3.3. If the entropy of the distribution $\mathcal{S}^{\sigma,\sigma_{\text{pre}}}(v)$ on the boundary is small (i.e. σ_2 does not play very randomly against σ_{pre}), then it should be possible to find a small subset of $S^\sigma(v)$ which captures most of the probability of $\mathcal{S}^{\sigma,\sigma_{\text{pre}}}(v)$. In this case, we could instead choose to only memorize moves on histories which are ancestors of this small subset. This would result in only a small number of histories to memorize. On the other hand, because this subset captures most of the probability of $\mathcal{S}^{\sigma,\sigma_{\text{pre}}}(v)$, this precomputation strategy will be almost as good as memorizing σ_{pre} on the entire subtree above $S^\sigma(v)$.

To summarize, if the entropy $H(\mathcal{S}^{\sigma,\sigma_{\text{pre}}}(v))$ of the distribution $\mathcal{S}^{\sigma,\sigma_{\text{pre}}}(v)$ is small, then we do not need to memorize too many states to construct the precomputation strategy which memorizes moves until an advantage v is reached. Since we get a value of at least v whenever we end up in $h \in S^\sigma(v)$, we expect a utility of approximately $vP_{\text{norm}}(v)$ if we follow this precomputation strategy. We can quantify this observation with the following Theorem:

Theorem 3.1. *Given $\sigma = (\sigma_1, \sigma_2)$ and a policy σ_{pre}, suppose that the maximum number of moves per game is $L := \max_{h \in H} |h|$. Then $\forall v \in [0,1], \varepsilon \in (0,1)$, there exists a $(\sigma_{pre}, \sigma_1, L(1-\varepsilon)e^{H(\mathcal{S}^{\sigma,\sigma_{pre}}(v))/\varepsilon})$ precomputation strategy $\tilde{\sigma}_1$ for player 1 such that*

$$u_1^{(\tilde{\sigma}_1, \sigma_2)} \geq (1-\varepsilon) v P_{norm}(v)$$

where H is the (base e) entropy of the distribution $\mathcal{S}^{\sigma,\sigma_{pre}}(v)$.

For any fixed $v \in [0,1], \varepsilon \in (0,1)$, Theorem 3.1 therefore gives us a lower bound on how exploitable a policy σ_2 is to precomputation. As expected, there is a direct tradeoff between the size of the precomputation set required to exploit σ_2, and how randomly σ_2 plays against some fixed policy σ_{pre}. To prove Theorem 3.1, we first need the following lemma:

Lemma 3.1. *Let D be a discrete distribution with probabilities $p_1 \geq p_2 \geq \cdots \geq p_{|D|}$. Then $\forall \gamma \in (0,1), \forall z \in \mathbb{N}$, if $z \geq \gamma e^{H(D)/(1-\gamma)}$, then $\sum_{i=1}^{z} p_i \geq \gamma$.*

Proof. Suppose $\sum_{i=1}^{z} p_i < \gamma$. Then we have

$$\begin{aligned} H(D) &\geq \sum_{i=z+1}^{|D|} p_i \ln\left(\frac{1}{p_i}\right) && (3.1)\\ &\geq \sum_{i=z+1}^{|D|} p_i \ln\left(\frac{1}{p_z}\right) && (3.2)\\ &> (1-\gamma)\ln\left(\frac{1}{p_z}\right) && (3.3)\\ &\geq (1-\gamma)\ln\left(\frac{1}{\gamma/z}\right) && (3.4) \end{aligned}$$

Where the last inequality follows because $p_z \leq \gamma/z$. Solving gives

$$z < \gamma e^{H(D)/(1-\gamma)} \tag{3.5}$$

and the result follows.

□

We can now prove Theorem 3.1.

Proof. Order the histories $h_1, \cdots \in S^{\sigma}(v)$ descending by probability with respect to $\pi^{(\sigma_{\text{pre}},\sigma_2)}$, with probabilities $p_1 \geq p_2 \geq \ldots$ respectively. Then setting $\gamma = 1 - \varepsilon$ in Lemma 3.1 gives us

$$\sum_{i=1}^{z} \pi^{(\sigma_{\text{pre}},\sigma_2)}(h_i) \geq (1-\varepsilon) P_{\text{norm}}(v) \tag{3.6}$$

where $z = \lceil (1-\varepsilon) e^{H(\mathcal{S}^{\sigma,\sigma_{\text{pre}}}(v))/\varepsilon} \rceil$. Now consider the precomputation strategy $\tilde{\sigma}_1$ which memorizes all histories in $S := \{h \in H : \exists i \in [z] \text{ s.t. } h < h_i\}$, where $|S| \leq Lz$. Then $\forall i \in [z]$, we have $\pi^{(\sigma_{\text{pre}},\sigma_2)}(h_i) = \pi^{(\tilde{\sigma}_1,\sigma_2)}(h_i)$. Since $h_i \not\leq h_j$ for $i \neq j$, and $\forall i \in [z], u^{(\sigma_1,\sigma_2)}(h_i) \geq v$, we have

$$u^{(\tilde{\sigma}_1,\sigma_2)} \geq \sum_{i=1}^{z} \pi^{(\sigma_{\text{pre}},\sigma_2)}(h_i) u^{(\sigma_1,\sigma_2)}(h_i) \geq (1-\varepsilon) v P_{\text{norm}}(v) \tag{3.7}$$

□

We now make the following observation: suppose that $\Delta\tilde{\sigma}_2^*$ is a Nash equilibrium strategy for player 2 in the meta-precomputation game. If this equilibrium has a favourable equilibrium value for player 2, then it follows from Theorem 3.1 that $\mathcal{S}^{\sigma,\sigma_{\text{pre}}}(v')$ must have high entropy. For the sake of contradiction, suppose that this were not the case: it would then be possible to construct a small precomputation strategy for player 1 which performs well against $\Delta\tilde{\sigma}_2^*$ but has low precomputation penalty, contradicting the assumption that $\Delta\tilde{\sigma}_2^*$ has a favourable equilibrium value for player 2. By following this proof by contradiction logic precisely, we obtain the following Theorem. We simplify the statement by considering the case where σ_{pre} always wins against a strategy sampled from $\Delta\tilde{\sigma}_2^*$.

Theorem 3.2. *Suppose $\Delta\tilde{\sigma}_2^*$ is a Nash equilibrium strategy for player 2 with value v in the meta-precomputation game. Suppose that $u_1^{(\sigma_{pre},\Delta\tilde{\sigma}_2^*)} = 1$,[4] the maximum number of moves in the game is L, and that the precomputation penalty term for player 1 is $\lambda_1 > 0$. Then $\forall v' \in (v + 0.01, 1)$, we have that*

$$H\left(\mathcal{S}^{(\sigma_1,\Delta\tilde{\sigma}_2^*),\sigma_{pre}}(v')\right) \geq (v' - v - 0.01)\left(\ln\left(\frac{1}{\lambda_1 L}\right) - 5\right)$$

For any fixed $v' \in (v + 0.01, 1)$, Theorem 3.2 therefore quantifies a lower bound on how randomly equilibrium policies play in the precomputation model. The bound tells us that the better the strategy $\Delta\tilde{\sigma}_2^*$ is for player 2 (the smaller the equilibrium value v is), the more randomly player 2 must play. In contrast to mini-max games in the computationally unbounded setting where optimal strategies are deterministic, optimal strategies in this model are inherently randomized. This qualitative finding is consistent with what one might have expected from intuition. The result is significant because it allows one to analytically quantify a lower bound on how randomly strategies behave in equilibrium.

Proof. By Kuhn's theorem, any mixed strategy in an extensive form game with perfect recall has an equivalent behaviour strategy. We therefore interchangeably associate the mixed strategy $\Delta\tilde{\sigma}_2^*$ with its equivalent behaviour strategy for the remainder of the proof. Since $u_1^{(\sigma_{\text{pre}},\Delta\tilde{\sigma}_2^*)} = 1$ by assumption, for any $v' \leq 1$ we have $P_{\text{norm}}(v') = 1$ for the distribution $\mathcal{S}^{\sigma,\sigma_{\text{pre}}}(v')$ where $\sigma = (\sigma_1, \Delta\tilde{\sigma}_2^*)$. Write $H(v') := H\left(\mathcal{S}^{\sigma,\sigma_{\text{pre}}}(v')\right)$ as a shorthand for the entropy of distribution $\mathcal{S}^{\sigma,\sigma_{\text{pre}}}(v')$. Because the Nash equilibrium value of the game is v, any precomputation strategy $\tilde{\sigma}_1$ for player 1 must have utility $\tilde{u}_1^{(\tilde{\sigma}_1,\Delta\tilde{\sigma}_2^*)} \leq v$; otherwise the mixed strategy which only plays $\tilde{\sigma}_1$ will have utility greater than v. Theorem 3.1 provides the existence of a precomputation strategy for player 1, and implies that $\forall \varepsilon \in (0, 1), v' \in [0, 1]$,

[4]Here we implicitly associate the mixed strategy $\Delta\tilde{\sigma}_2^*$ with its equivalent behaviour strategy as given by Kuhn's theorem.

$$v \geq \tilde{u}_1^{(\tilde{\sigma}_1, \Delta\tilde{\sigma}_2^*)} \geq (1-\varepsilon)v' - \lambda_1 L(1-\varepsilon)e^{H(v')/\varepsilon} \tag{3.8}$$

Assume $(1-\varepsilon)v' - v > 0$. Then we can write

$$e^{H(v')/\varepsilon} \geq \frac{(1-\varepsilon)v' - v}{\lambda_1 L(1-\varepsilon)} \tag{3.9}$$

$$H(v') \geq \varepsilon \left(\ln\left(\frac{1}{\lambda_1 L}\right) + \ln\left(\frac{(1-\varepsilon)v' - v}{1-\varepsilon}\right)\right) \tag{3.10}$$

Assume $v' > v + e^{-k}$, and set $\varepsilon = \frac{v'-v-e^{-k}}{v'-e^{-k}} \in (0,1)$. It can be verified that this choice of ε satisfies our previous assumption $(1-\varepsilon)v' - v > 0$. Simplifying for this choice of ε gives:

$$H(v') \geq \frac{v'-v-e^{-k}}{v'-e^{-k}} \left(\ln\left(\frac{1}{\lambda_1 L}\right) + \ln(e^{-k})\right) \tag{3.11}$$

$$\geq (v'-v-e^{-k}) \left(\ln\left(\frac{1}{\lambda_1 L}\right) - k\right) \tag{3.12}$$

$$\geq (v'-v-0.01) \left(\ln\left(\frac{1}{\lambda_1 L}\right) - 5\right) \tag{3.13}$$

for $k = 5$. This gives the required inequality.

□

3.4 Efficiently Finding Precomputation Strategies And Their Equilibria

Suppose we are given oracle access to algorithmic strategies $\sigma_1, \sigma_2, \sigma_{\text{pre}}$. Concretely, we could have $\sigma_1, \sigma_2, \sigma_{\text{pre}}$ = Stockfish with different time per move settings. We might believe there is a good precomputation strategy for σ_1 against σ_2, but it is perhaps unclear how to find such a strategy. Suppose $\tilde{\sigma}_1$ is an optimal precomputation strategy for player 1 against σ_2, with memorization set S. The first observation is that we can assume $h \in S \implies \pi^{(\sigma_{\text{pre}}, \sigma_2)}(h) \geq \lambda_1$; otherwise, player 1 could strictly increase

$\tilde{u}_1^{(\tilde{\sigma}_1,\sigma_2)}$ by removing h and any descendants of h from S. This is because the cost of including h and any descendants of h in S is at least λ_1, but the gain in utility from any histories which pass through h when precomputing up to h is at most $\pi^{(\sigma_{\text{pre}},\sigma_2)}(h)$. Using this observation, it is possible to show $|S| \leq \frac{L+1}{\lambda_1}$ (Lemma 3.2). It follows that finding the optimal precomputation strategy for player 1 against σ_2 is equivalent to finding the optimal subtree (memorization set) contained within a bounding tree of size $\frac{L+1}{\lambda_1}$ to memorize. By using standard Hoeffding bounds to sample the value of the game at the leaves of this bounding tree, we can compute the optimal subtree with standard dynamic programming techniques. We now make these ideas precise.

Lemma 3.2. $|\{h \in H : \pi^{\sigma_1,\sigma_2}(h) \geq p\}| \leq \frac{L+1}{p}$, *where* σ_1, σ_2 *are any policies and the maximum history length of the game is* $L := \max_{h \in H} |h|$.

Proof. Note sum of probabilities of all histories with length equal to $l \in \mathbb{N}$ is at most 1. We therefore have

$$
\begin{aligned}
L + 1 &\geq \sum_{l=0}^{L} \sum_{\substack{h \in H \\ |h|=l}} \pi^{\sigma_1,\sigma_2}(h) && (3.14)\\
&= \sum_{h \in H} \pi^{\sigma_1,\sigma_2}(h) && (3.15)\\
&\geq \sum_{\substack{h \in H \\ \pi^{\sigma_1,\sigma_2}(h) \geq p}} \pi^{\sigma_1,\sigma_2}(h) && (3.16)\\
&\geq p|\{h \in H | \pi^{\sigma_1,\sigma_2}(h) \geq p\}| && (3.17)
\end{aligned}
$$

so

$$
|\{h \in H | \pi^{\sigma_1,\sigma_2}(h) \geq p\}| \leq \frac{L+1}{p} \tag{3.18}
$$

□

Algorithm 1: ComputeBestPrecomputationResponse (**CBPR**)

Receive as input $(\sigma_1, \sigma_2), \sigma_{\text{pre}}, \lambda_1, \varepsilon, \delta$;

Define $\text{est}(h, K)$ to be an estimate of $u_1^{\sigma_1,\sigma_2}(h)$ using the mean value of K games sampled from history h played according to $\sigma = (\sigma_1, \sigma_2)$. This can be computed in time $\mathcal{O}(LK)$;

Initialize BESTCHOICES, $S = \emptyset$;

For $h \in Z$ define $\text{bestvalue}(h) = u(z)$. Recursively compute $\text{bestvalue}(h) = \text{est}(h, K) \times \pi^{(\sigma_{\text{pre}},\sigma_2)}(h)$ if $\pi^{(\sigma_{\text{pre}},\sigma_2)}(h) < \lambda_1$. Otherwise, $\text{bestvalue}(h) = \max[\sum_{h' \in \text{Succ}(h)} \text{bestvalue}(h') - \lambda_1, \text{est}(h, K)\pi^{(\sigma_{\text{pre}},\sigma_2)}(h)]$. If the left argument of the max is larger, we add h to BESTCHOICES. Here we set $K = \mathcal{O}\left(\ln\left(\frac{AL}{\delta\lambda_1}\right)/\varepsilon^2\right)$;

If the starting history $\emptyset \in$ BESTCHOICES, then depth first search from $\emptyset$, where there is an edge $h \to h'$ iff $h' \in \text{Succ}(h)$ and $h' \in$ BESTCHOICES. Add each history visited to S;

Return $\tilde{\sigma}_1'$, the precomputation strategy which memorizes σ_{pre} on set S. $\tilde{\sigma}_1'$ is a precomputation strategy for player 1 which approximately maximizes $\tilde{u}^{(\tilde{\sigma}_1',\sigma_2)}$;

Theorem 3.3. *For any $\varepsilon, \delta > 0$, and precomputation penalty factor $\lambda_1 > 0$ for player 1, suppose we are given a policy profile $\sigma = (\sigma_1, \sigma_2)$ and a policy σ_{pre} as constant-time oracles. Suppose further that $\exists \tilde{\sigma}_1 \in Pre(\sigma_{pre}, \sigma_1)$ such that $\tilde{u}_1^{(\tilde{\sigma}_1,\sigma_2)} \geq v$. Then with probability at least $1 - \delta$, Algorithm 1 will return $\tilde{\sigma}_1' \in Pre(\sigma_{pre}, \sigma_1)$ such that $\tilde{u}_1^{(\tilde{\sigma}_1',\sigma_2)} \geq v - \varepsilon$. Moreover, Algorithm 1 runs in time $\mathcal{O}\left(A^2 \frac{L^2}{\lambda_1 \varepsilon^2} \ln(\frac{AL}{\delta\lambda_1})\right)$, where $L = \max_{h \in H} |h|$ and $A = \max_{h \in H} |A(h)|$.*

Proof. For any $S \subset H \setminus Z$ which is a prefix closed subset of $H_1 = \{h \in H | P(h) = 1\}$, and for a function $f : H \to [0, 1]$ define

$$\text{value}(S, f) = -\lambda_1 |S| + \sum_{h \in \text{boundary}(S)} f(h)\pi^{(\sigma_{\text{pre}},\sigma_2)}(h) \tag{3.19}$$

where $\text{boundary}(S) = (\cup_{h \in S} \text{Succ}(h)) \setminus S$ if $S \neq \emptyset$, and $\text{boundary}(\emptyset) = \{\emptyset\}$. $\text{value}(S, f)$ computes the penalization term $-\lambda_1|S|$ for the size of S, plus the expectation of $f(h)$ weighted by the probability of reaching h when player 1 plays strategy σ_{pre} and player 2 plays strategy σ_2. The set $\text{boundary}(S)$ is equal to the histories

when player 1 first gets a turn (or the game has ended) after leaving the precomputing set S. It follows that $\text{value}(S, u_1^{(\sigma_1,\sigma_2)}(\cdot)) = \tilde{u}_1^{(\tilde{\sigma}_1,\sigma_2)}$ is precisely the value for player 1 when using precomputation strategy $\tilde{\sigma}_1$ with memorization set S against player 2 with strategy σ_2. On the other hand, fix the estimates $\text{est}(h, K)$ for $h \in H$. In line 5, Algorithm 1 uses a standard dynamic programming technique to compute the optimal prefix-closed set $S' \subset \{h \in H_1 | \pi^{(\sigma_{\text{pre}},\sigma_2)}(h) \geq \lambda_1\}$ which maximizes $\text{value}(S', \text{est}(\cdot, K))$. We will show that this set S', computed using estimates and on a smaller portion of the entire game tree, corresponds to a precomputation strategy with value at least $v - \varepsilon$ with probability at least $1 - \delta$.

Let S be the memorization set of the precomputation strategy $\tilde{\sigma}_1$ assumed in Theorem 3.3. Then without loss of generality we can assume $h \in S \implies \pi^{(\sigma_{\text{pre}},\sigma_2)}(h) \geq \lambda_1$; otherwise, we could strictly increase $\tilde{u}_1^{(\tilde{\sigma}_1,\sigma_2)}$ by removing h and any descendants of h from S. Let $W_1 = \{h \in H | \pi^{(\sigma_{\text{pre}},\sigma_2)}(h) \geq \lambda_1\}$, $W_2 = \text{Succ}(W_1) = \cup_{h \in W_1} \text{Succ}(h)$ and $W = W_1 \cup W_2$. Suppose that we have $\forall h \in W, |\text{est}(h, K) - u_1^{(\sigma_1,\sigma_2)}(h)| < \frac{\varepsilon}{2}$. Since $\text{value}(S, u_1^{(\sigma_1,\sigma_2)}(\cdot)) = \tilde{u}_1^{(\tilde{\sigma}_1,\sigma_2)} \geq v$, it follows that

$$\begin{aligned}
\text{value}(S, \text{est}(\cdot, K)) &\geq -\lambda_1 |S| + \sum_{h \in \text{boundary}(S)} \left(u_1^{(\sigma_1,\sigma_2)}(h) - \frac{\varepsilon}{2} \right) \pi^{(\sigma_{\text{pre}},\sigma_2)}(h) && (3.20) \\
&= -\lambda_1 |S| + \sum_{h \in \text{boundary}(S)} u_1^{(\sigma_1,\sigma_2)}(h) \pi^{(\sigma_{\text{pre}},\sigma_2)}(h) \\
&\quad - \sum_{h \in \text{boundary}(S)} \frac{\varepsilon}{2} \pi^{(\sigma_{\text{pre}},\sigma_2)}(h) && (3.21) \\
&= v - \varepsilon/2 && (3.22)
\end{aligned}$$

where we have used that the sum of the probabilities of the histories in $\text{boundary}(S)$ is equal to 1. Therefore, the optimal precomputation set S' computed by Algorithm 1 satisfies $\text{value}(S', \text{est}(\cdot, K)) \geq v - \frac{\varepsilon}{2}$. By the same reasoning, it must also be that $\text{value}(S', u_1^{(\sigma_1,\sigma_2)}(\cdot)) \geq v - \varepsilon$. It follows that the precomputation strategy $\tilde{\sigma}_1'$ returned by Algorithm 1 satisfies $\tilde{u}_1^{(\tilde{\sigma}_1',\sigma_2)} \geq v - \varepsilon$.

It remains to show that $\forall h \in W, |\text{est}(h,K) - u_1^{(\sigma_1,\sigma_2)}(h)| < \frac{\varepsilon}{2}$ holds with probability at least $1-\delta$. We have $|W| \leq |A|^2 \left(\frac{(L+1)}{\lambda_1}\right) + \frac{(L+1)}{\lambda_1} \leq 2|A|^2 \frac{(L+1)}{\lambda_1}$ by applying Lemma 3.2 on the set W_1, and using the fact that the number of successors of W_1 can be at most a factor A^2 larger. By Hoeffding's inequality (Theorem A.1 in appendix) and union bound, the probability of the event $\exists h \in W$ s.t. $|\text{est}(h,K) - u_1^{(\sigma_1,\sigma_2)}(h)| \geq \frac{\varepsilon}{2}$ is

$$\leq |W| 2 \exp\left(-\frac{K}{2}(\varepsilon/2)^2\right) \leq \delta \tag{3.23}$$

for $K = 16 \ln\left(\frac{|W|}{\delta}\right)/\varepsilon^2 \leq 16 \ln\left(\frac{2A^2(L+1)}{\delta\lambda_1}\right)/\varepsilon^2$. Thus with probability at least $1-\delta$, the estimates are all within $\frac{\varepsilon}{2}$ of the true values and we find a precomputation policy $\tilde{\sigma}_1'$ with $\tilde{u}_1^{(\tilde{\sigma}_1',\sigma_2)} \geq v - \varepsilon$. The algorithm considers at most $|W|$ histories, where each history takes time LK to sample, giving a total running time of $|W|LK = \mathcal{O}\left(|A|^2 \frac{L^2}{\lambda_1 \varepsilon^2} \ln(\frac{AL}{\delta\lambda_1})\right)$. □

3.4.1 Computing Equilibria

In the previous setting, we imagined the opposing player as fixed, while the precomputing player optimizes the best set of game states to memorize in order to exploit the opposing player. In reality, the opposing player reacts to this precomputation by preparing their own lines of play in anticipation to counter this precomputation. In this subsection we will show how an $\varepsilon-$Nash equilibrium for the meta-precomputation game can be efficiently found, by reducing the problem to a form which can be solved by the well-studied method of Counter Factual Regret Minimization (CFR) (Zinkevich et al., 2007). CFR is an algorithm which finds an ε-Nash equilibrium to an imperfect information, perfect recall, zero sum extensive form game. It works by iteratively playing strategies against each other in a way which minimizes regret. While in theory CFR reaches an ε-Nash equilibrium after a number of steps which is quadratic in the size of the game, in practice this can occur much faster. Moreover,

the algorithm can be parallelized. It has been used to find Nash equilibria to very large Texas Hold'em games.

To compute an equilibrium of the meta precomputation game, we first observe (similar to the previous subsection) that we can ignore low probability histories $h \in H$ where the possible gain in utility of precomputing at h, regardless of the opposing player's strategy, is always upper bounded by the precomputation penalty λ_i. For example, player 2 only has an incentive to precompute on the set $W_2 = \left\{h \in H | \sup_{\Delta\tilde{\sigma}_1 \in \Delta\text{Pre}(\sigma_{\text{pre}},\sigma_1)} \pi^{\Delta\tilde{\sigma}_1,\sigma_{\text{pre}}}(h) \geq \lambda_2\right\}$. If we denote the set of high probability histories by W, then we can bound the size of W by $|W| \leq \mathcal{O}\left(A^2L^2(\frac{1}{\lambda_1} + \frac{1}{\lambda_2})\right)$ (Lemma 3.3).

The second observation is that the meta-precomputation game G can be viewed as an equivalent extensive form game G' with imperfect information and perfect recall. At each game state $h \in H$, we can imagine that instead of players choosing a distribution of actions in $A(h)$ to play, they choose whether to play an action from distribution $\sigma_{\text{pre}}(h)$ (i.e. continue precomputing) or distribution $\sigma_i(h)$ (stop precomputing); however, if a player chooses to play from distribution $\sigma_{\text{pre}}(h)$, they must pay a cost of λ_i corresponding to the precomputation penalty. We must reweight this penalty so that it is not discounted by the probability of reaching history h. Players cannot observe whether their opponent chose to play from distribution $\sigma_{\text{pre}}(h)$ or $\sigma_i(h)$, but they do see the action which was sampled from the distribution their opponent chose (equivalent to observing the history which results from that action). The memorization set for a precomputation strategy then naturally corresponds to the set of histories where a player chooses to play from distribution $\sigma_{\text{pre}}(h)$, and so there is a bijective correspondence between *pure strategies* in game G, and *pure deterministic strategies* in game G'. Using the equivalence between behavior strategies and mixed strategies for extensive form games with perfect recall (Kuhn's theorem), we can then get an equivalence between behavior strategies in game G', and mixed strategies in

game G. We have therefore reduced the problem of finding an ε-Nash equilibrium of mixed strategies in game G to finding an ε-Nash equilibrium of behavior strategies in game G'. This latter problem is precisely what is solved by CFR. Moreover, we can ensure CFR runs efficiently if we apply previous insights. In particular, we remove low probability histories $H \setminus W$ from the game, and estimate the value of histories with random roll-outs. We now make these ideas precise.

Lemma 3.3. *Let*

$$W_1 = \left\{ h \in H | \sup_{\Delta\tilde{\sigma}_2 \in \Delta Pre(\sigma_{pre}, \sigma_2)} \pi^{\sigma_{pre}, \Delta\tilde{\sigma}_2}(h) \geq \lambda_1 \right\}$$
$$W_2 = \left\{ h \in H | \sup_{\Delta\tilde{\sigma}_1 \in \Delta Pre(\sigma_{pre}, \sigma_1)} \pi^{\Delta\tilde{\sigma}_1, \sigma_{pre}}(h) \geq \lambda_2 \right\}$$
$$W = W_1 \cup Succ(W_1) \cup W_2 \cup Succ(W_2)$$

where $Succ(X) = \cup_{h \in X} Succ(h)$. *Then*

$$|W| \leq \mathcal{O}\left(A^2 L^2 \left(\frac{1}{\lambda_1} + \frac{1}{\lambda_2} \right) \right)$$

where $L = \max_{h \in H} |h|$ *and* $A = \max_{h \in H} |A(h)|$.

Proof. Denote the player opposing i by $-i$. For $l' = \{0, \ldots, L+1\}$, let $\tilde{\sigma}_{-i,l'}$ be the precomputation strategy for player $-i$ which plays σ_{pre} on every $h \in H$ such that $|h| < l'$, and σ_{-i} otherwise. Let $(\sigma_{\text{pre}}, \tilde{\sigma}_{-i,l'})$ be the policy profile where player i plays σ_{pre} and player $-i$ plays $\tilde{\sigma}_{-i,l'}$. Define $W_{i,l'} = \{h \in H | \pi^{(\sigma_{\text{pre}}, \tilde{\sigma}_{-i,l'})}(h) \geq \lambda_i\}$ The key idea of the proof is as follows. Suppose $\pi^{\sigma_{\text{pre}}, \Delta\tilde{\sigma}_{-i}}(h) \geq p$ for some mixed precomputation strategy $\Delta\tilde{\sigma}_{-i}$. Then $\pi^{\sigma_{\text{pre}}, \tilde{\sigma}_{-i}}(h) \geq p$ for some pure precomputation strategy $\tilde{\sigma}_{-i}$. If l' is the maximum integer such that $\tilde{\sigma}_{-i}$ precomputes on histories $h_{:0}, \ldots, h_{:l'}$, then $\pi^{\sigma_{\text{pre}}, \tilde{\sigma}_{-i,l'}}(h) = \pi^{\sigma_{\text{pre}}, \tilde{\sigma}_{-i}}(h) \geq p$. This is because the policies $\tilde{\sigma}_{-i,l'}$ and $\tilde{\sigma}_{-i}$ assign exactly the same action probabilities on any history h' where $P(h') =$

$-i$ and h' is a prefix of h. Therefore, in order to count $|W_i|$, we can consider each strategy $\tilde{\sigma}_{-i,l'}$ for $l' \in \{0, \ldots, L+1\}$ and sum over the $h \in H$ where $\tilde{\sigma}_{-i,l'}(h) \geq p$. We therefore have

$$|W_i| = \sum_{l'=0}^{L+1} \sum_{h \in W_{i,l'}} 1 \tag{3.24}$$

$$\leq \sum_{l'=0}^{L+1} |\{h \in H : \pi^{\sigma_{\text{pre}}, \tilde{\sigma}_{-i,l'}}(h) \geq \lambda_i\}| \tag{3.25}$$

$$\leq (L+2)\frac{(L+1)}{\lambda_i} \tag{3.26}$$

$$\leq \frac{(L+2)^2}{\lambda_i} \tag{3.27}$$

Line 3.26 follows from Lemma 3.2. Using that $|W_i \cup \text{Succ}(W_i)| \leq \mathcal{O}(A^2|W_i|)$, and summing for $i = 1, 2$ gives the required bound. □

Recall that in an imperfect information game, the histories of the game are partitioned into information sets. Players are constrained to behaving identically on histories which are contained in the same information set.

Definition 3.4. *Given a meta-precomputation game G with policies $\sigma_1, \sigma_2, \sigma_{pre}$ and histories H, define the extensive form imperfect information, perfect recall game G' with the following tree structure:*

1. *The game G' consists of player 1, player 2 and a chance player denoted by c. Denote the set of histories for G' by H' and terminal histories by Z'. Each history $h' \in H'$ for game G' where $P(h') \neq c$ is associated with a history $h = Q(h') \in H$ in the original game G. Histories $h' \in H'$ are defined recursively as follows. The starting history h' is associated with $\emptyset = Q(h')$, and $P(h') = 1$. At any history $h' \in H$, if $P(h') \neq c$, then player $i = P(h')$ has the following action choices. Either player i can choose to continue precomputing provided they have not previously chosen to stop precomputing (action 1), or player i can choose to*

stop precomputing (action 0). h' then transitions to a history h'' corresponding to a chance node, i.e. $P(h'') = c$. The chance player picks an action $a \in A(h)$ according to the distribution $\sigma_{pre}(h)$ if player i chose to continue precomputing in the previous step, or according to distribution $\sigma_i(h)$ otherwise. The game then transitions to a history h''' where $P(h''') = P(\pi(h, a))$ if $\pi(h, a) \notin Z$ is non-terminal. History $h''' \in H'$ is associated with history $\pi(h, a) = Q(h''') \in H$.

2. *Players can see their own actions and the actions of the chance player, but not the actions of the opposing player. Formally, two histories $h', h'' \in H'$ for player $P(h') = P(h'') = i \in \{1, 2\}$ are in the same information set iff they have the same sequence of actions for player i and the chance player leading up to them.*

3. *Terminal histories Z' in game G' are histories $h' \in H'$ where the associated history $h = Q(h') \in Z$ is a terminal history in game G. The utility of a terminal history $h' \in Z'$ for player 1 is $u'(h') := u(Q(h')) - \lambda_1 z_1(h') + \lambda_2 z_2(h')$ for some weighting functions z_i to be specified. For $h' \in H'$ and policy profile $(\tilde{\sigma}'_1, \tilde{\sigma}'_2)$, we denote the expected value of the game G' for player 1 starting from history h' by $(u')_1^{(\tilde{\sigma}'_1, \tilde{\sigma}'_2)}(h')$.*

Lemma 3.4. *Given a meta-precomputation game G with policies $\sigma_1, \sigma_2, \sigma_{pre}$, let G' be the game defined in Definition 3.4. Then there exists weighting functions $z_1, z_2 : H' \to [0, 1]$ such that for every pair of behavior strategies $(\tilde{\sigma}'_1, \tilde{\sigma}'_2)$ in G' with game value v, there is a pair of mixed strategies $\tilde{\sigma}_1, \tilde{\sigma}_2$ in the meta precomputation game with the same value v, and vice versa. Moreover, let $\tilde{\sigma}'$ be a policy profile where each player plays deterministically in this meta game. Then for these choices of z_1, z_2, we have that*

$$\sum_{h' \in Z'} \pi^{\tilde{\sigma}'}(h')|u'(h')| \leq 1 + \lambda_1|S_1| + \lambda_2|S_2| \tag{3.28}$$

where $|S_i|$ is the size of the memorization set for player i in game G corresponding to policy profile $\tilde{\sigma}'$.

Proof. See Appendix A. □

We need a slight adaption the guarantees of CFR in Zinkevich et al. (2007), which we give below.

Lemma 3.5. *Given an imperfect information, perfect recall, zero sum two player extensive form game G', let X be the space of policy profiles, and let*

$$C = 2 \max_{i \in \{1,2\}} \sup_{\sigma \in X} \sum_{h' \in Z'} \pi^{\sigma}(h') |u_i(h')|$$

where $u_1^{\sigma}(h')$ is the value of the game for player 1 at terminal history h'. Let $|\mathcal{I}|$ be the number of information sets in the game. Then when CFR is run on G' for $T = \mathcal{O}\left(\frac{C^2 |\mathcal{I}|^2 A}{\varepsilon^2}\right)$ iterations, an ε-Nash equilibrium is reached. Each iteration of CFR takes time $\mathcal{O}(\mathcal{H}')$, where $\mathcal{H}'$ is the number of histories in game G'.

Proof. See Appendix A. □

Algorithm 2: ComputeMetaNash (**CMN**).

1 Receive as input $(\sigma_1, \sigma_2), \sigma_{\text{pre}}, \lambda_1, \lambda_2, \varepsilon, \delta$;
2 Define $\text{est}(h', K)$ to be an estimate of $(u')_1^{\sigma'}(h')$ using the mean value of K games sampled from history h' played according to $\sigma' = (\text{null}, \text{null})$, where null is the policy which never chooses to precompute. This can be computed in time $\mathcal{O}(LK)$;
3 Let G be the meta precomputation game. Compute sets W_1, W_2, W which bound the set of high probability histories for G (using the proof for Lemma 3.3). Let G' be the equivalent extensive form game for G (Definition 3.4);
4 Modify G' as follows. (1) Remove the option for player i to precompute in history $h' \in H'$ if $Q(h') \notin W_{P(h')}$. (2) Histories $h' \in H'$ for game G' are now made to be terminal histories if $h = Q(h') \notin (W_1 \cup W_2)$ or both players have stopped precomputing. For such h', estimate the utility by $\text{est}(h', K)$ for $K = \mathcal{O}\left(\ln\left(\frac{L|W|}{\delta}\right)/\varepsilon^2\right)$;
5 Run CFR from Zinkevich et al. (2007) on the modified form of G' for $T = \mathcal{O}\left(A(L|W|)^4 \frac{(\lambda_1+\lambda_2)^2}{\varepsilon^2}\right)$ iterations;
6 Return the result of CFR;

Theorem 3.4. *For any $\varepsilon > 0, \delta > 0$ and precomputation penalty factors $\lambda_1, \lambda_2 > 0$, suppose we are given a policy profile $\sigma = (\sigma_1, \sigma_2)$ and a policy σ_{pre} as constant-time oracles. Then Algorithm 2 returns an ε-Nash equilibrium to the meta-precomputation game with probability at least $1 - \delta$. Moreover, the total number of CFR iterations is bounded by $T = \mathcal{O}\left(A(L|W|)^4 \frac{(\lambda_1+\lambda_2)^2}{\varepsilon^2}\right)$, and the total runtime is bounded by*

$$\mathcal{O}\left(A\left(A^2L^3\left(\frac{1}{\lambda_1}+\frac{1}{\lambda_2}\right)\right)^5 \frac{(\lambda_1+\lambda_2)^2}{\varepsilon^2}\right)$$
$$+\mathcal{O}\left(\frac{1}{\varepsilon^2}A^2L^4\left(\frac{1}{\lambda_1}+\frac{1}{\lambda_2}\right)\ln\left(AL\left(\frac{1}{\lambda_1}+\frac{1}{\lambda_2}\right)/\delta\right)\right)$$

How significant is this bound? From a theoretical perspective, it shows that one can approximate the Nash equilibrium in polynomial time. On the other hand, the bound is too large to imply practical consequences. This is to be expected because the underlying technique used to prove the bound relied on CFR, and CFR gives an impractical regret guarantee which is quadratic in the number of states of the game. However, impractical worst case bounds do not rule out that this algorithm could not be used efficiently in practice. In practice it is possible that a Nash equilibrium can be reached after fewer than $T = \mathcal{O}\left(A(L|W|)^4 \frac{(\lambda_1+\lambda_2)^2}{\varepsilon^2}\right)$ iterations of CFR. In particular, at iteration $T' < T$, one can use Theorem 3.3 to verify whether the ε-Nash equilibrium condition holds by computing the optimal precomputation strategy value against the current opponent strategy. One can use different variants of CFR which are designed to converge faster in practice depending on the specific structure of the game in question. As discussed earlier, CFR is used in practice to efficiently find Nash equilibria for Texas hold'em poker, despite the fact that the theoretical worst case runtime of CFR for poker is impractical.

Proof. Without loss of generality, we can assume any strategy for player i in game G' only chooses to precompute at histories $h' \in H'$ where $Q(h') \in W_i$. Therefore an ε-Nash equilibrium for the modified form of G' is an ε-Nash equilibrium for G'.

Note that in the modified form of G', we will always reach a terminal history before player i reaches a history $h' \in H'$ where $Q(h') \notin W$. Therefore the number of histories in the modified game G' is $\mathcal{O}(L|W|)$. This is because for each history h' in the modified game, either at least one player is still precomputing, or both players have stopped precomputing and we are at a terminal history. Without loss of generality, let's suppose player i had chosen to stop precomputing at a prior step and the other player is still precomputing. Then history h' where $P(h') \in \{1, 2\}$ can be uniquely identified by its corresponding history $h \in Q(h')$ (which encodes the actions of the chance player), and the step number $x_i \in \{0, \ldots, L\}$ during which player i chose to stop precomputing along history h (which encodes the actions of player i along h'). Suppose we pick $K = \mathcal{O}\left(\ln\left(\frac{L|W|}{\delta}\right)/\varepsilon^2\right)$ samples for each call to $\text{est}(h', K)$ for $Q(h') \in W$, so that with probability at least $1 - \delta$ we have $\forall h' \in H'$ s.t. $Q(h') \in W, |\text{est}(h', K) - (u')_1^{(\sigma_1,\sigma_2)}(h')| < \varepsilon/4$ (by applying Hoeffding and Union bound, similarly to the proof of Theorem 3.3). Let $\tilde{u}_{1,est}$ be the value function of the modified game G' where we use the estimated values to compute the terminal values of the game. Suppose we run counter factual regret minimization to get a $\frac{\varepsilon}{2}$-Nash equilibrium $(\Delta\tilde{\sigma}_1, \Delta\tilde{\sigma}_2)$ to this game using value function $\tilde{u}_{1,est}$. Then we have

$$\tilde{u}_1^{(\Delta\tilde{\sigma}_1,\Delta\tilde{\sigma}_2)} + \frac{\varepsilon}{4} \geq \tilde{u}_{1,est}^{(\Delta\tilde{\sigma}_1,\Delta\tilde{\sigma}_2)} \tag{3.29}$$

$$\geq \sup_{\Delta\tilde{\sigma}_1 \in \Delta(\text{Pre}(\sigma_{\text{pre}},\sigma_1))} \tilde{u}_{1,est}^{(\Delta\tilde{\sigma}_1,\Delta\tilde{\sigma}_2)} - \frac{\varepsilon}{2} \tag{3.30}$$

$$\geq \sup_{\Delta\tilde{\sigma}_1 \in \Delta(\text{Pre}(\sigma_{\text{pre}},\sigma_1))} \tilde{u}_1^{(\Delta\tilde{\sigma}_1,\Delta\tilde{\sigma}_2)} - \frac{\varepsilon}{2} - \frac{\varepsilon}{4} \tag{3.31}$$

Thus

$$\tilde{u}_1^{(\Delta\tilde{\sigma}_1,\Delta\tilde{\sigma}_2)} + \varepsilon \geq \sup_{\Delta\tilde{\sigma}_1 \in \Delta(\text{Pre}(\sigma_{\text{pre}},\sigma_1))} \tilde{u}_1^{(\Delta\tilde{\sigma}_1,\Delta\tilde{\sigma}_2)} \tag{3.32}$$

and the analogous inequality holds for player 2, giving an ε Nash equilibrium.

Each round of Counter Factual Regret Minimization requires time $\mathcal{O}(|H'|) = \mathcal{O}(L|W|)$, where $|\mathcal{H}'|$ is the number of histories in the modified version of G'. Note there are at most $\mathcal{O}(|H'|) = \mathcal{O}(L|W|)$ information sets, and

$$\max_{i\in\{1,2\}} \sup_{\sigma\in X} \sum_{h'\in Z'} \pi^{\sigma}(h')|u_i'(h')| \leq \mathcal{O}((\lambda_1+\lambda_2)|H'|) \tag{3.33}$$

Therefore by Lemma 3.5, after

$$\begin{aligned} T &= \mathcal{O}\left(\frac{C^2|\mathcal{I}|^2A}{\varepsilon^2}\right) && (3.34)\\ &= \mathcal{O}\left(\frac{((\lambda_1+\lambda_2)L|W|)^2(L|W|)^2A}{\varepsilon^2}\right) && (3.35)\\ &= \mathcal{O}\left(A(L|W|)^4\frac{(\lambda_1+\lambda_2)^2}{\varepsilon^2}\right) && (3.36) \end{aligned}$$

iterations we get an $\varepsilon/2$-Nash equilibrium.

The total runtime for the regret minimization component is

$$\mathcal{O}(T\times L|W|) = \mathcal{O}\left(A(L|W|)^5\frac{(\lambda_1+\lambda_2)^2}{\varepsilon^2}\right) \tag{3.37}$$

and the runtime for sampling utility estimates is

$$\mathcal{O}(|H'|\times KL) = \mathcal{O}(L|W|KL) \tag{3.38}$$

giving a total runtime bound of

$$\begin{aligned} &\mathcal{O}\left(A\left(A^2L^3\left(\frac{1}{\lambda_1}+\frac{1}{\lambda_2}\right)\right)^5\frac{(\lambda_1+\lambda_2)^2}{\varepsilon^2}\right)\\ &+\mathcal{O}\left(\frac{1}{\varepsilon^2}A^2L^4\left(\frac{1}{\lambda_1}+\frac{1}{\lambda_2}\right)\ln\left(AL\left(\frac{1}{\lambda_1}+\frac{1}{\lambda_2}\right)/\delta\right)\right) \end{aligned} \tag{3.39}$$

□

3.5 Experiments

While numerically evaluating CFR to compute equilibria in games is well-studied, finding the correct heuristic variant of CFR to make Algorithm 2 converge quickly for a particular game in practice is a research question on its own, and beyond the scope of this chapter. However, the susceptibility of real-world strategies to precomputation, and how useful precomputation can be as a function of the randomness of an opposing strategy, is both relatively unexplored and also feasible to compute in practice without further optimization. We therefore choose to focus on this aspect, and propose an experiment to numerically explore the popular chess engine Stockfish's susceptibility to precomputation using Algorithm 1. Specifically, we choose σ_{pre} to be Stockfish with 50ms per move, while σ_1 and σ_2 play as Stockfish with 10ms per move. For example, Algorithm 1 will transform white=Stockfish(10ms) into a new algorithm white=Stockfish'(10ms), which still takes at most 10ms per move, uses slightly more memory, and performs substantially better against black=Stockfish(10ms) if black does not sufficiently randomize. In particular, we explore how varying the randomness of the policy affects its susceptibility to precomputation. To do this, we introduce a randomness parameter r, and have a policy play more randomly as r increases. As $r \to 0$, σ_i will play only the best moves found by Stockfish in the required time period. As $r \to \infty$, σ_i will play uniformally between available moves. If player i is the fixed opponent, we set $\sigma_i(h) = \text{Softmax}(\text{Stockfish}(h)/r)$, where Softmax is the softmax function and Stockfish(h) returns a vector of the center pawn (cp) scores of available moves.[5] First we fix the default policies for white (σ_1,σ_{pre} with $r = 10^{-6}$), and vary r for black (σ_2), computing the optimal precomputation value against black for each level of randomness. Then we repeat the experiment for black precomputing against white. We set $\lambda_1 = \lambda_2 = 10^{-5}$ in these experiments, and plot the precomputed strategy utility (without precomputation penalty) and memorization set size for varying

[5] A center pawn score of 100 corresponds to having an advantage of one center pawn.

levels of randomness. In order to keep the computation requirements modest, only the top $K = 2$ moves of the Stockfish engine were considered at each board position. Instead of sampling the game value, a conservative cp bound was used to approximate when either player had a decisive advantage. Further experimentation details can be found in the appendix, and the full code can be found on Github.[6]

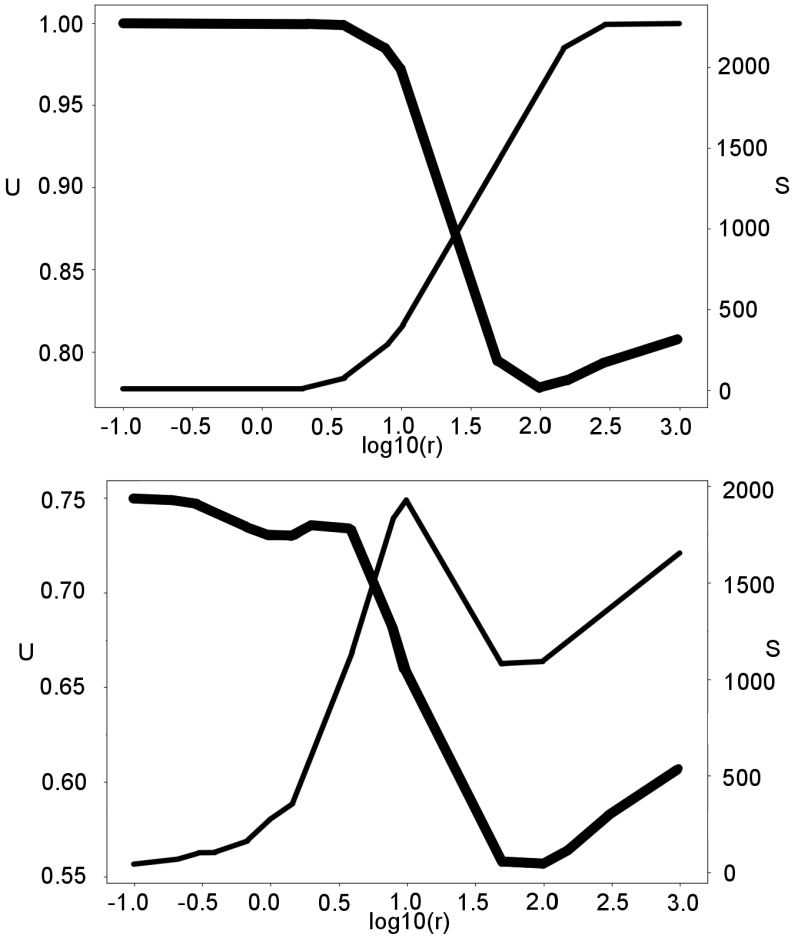

Figure 3.1: Precomputation effectiveness as a function of player randomness. Top: white as precomputing player. Bottom: black as precomputing player. The utility U of the precomputation strategy for the precomputing player, ignoring the precomputation memorization penalty (thicker line, left axis) and the size S of the optimal precomputation set (thinner line, right axis) are plotted against the opponent's randomness level $\log_{10}(r)$.

.

We see from the results that when either white or black is precomputing, precom-

[6] https://github.com/Thomas-Orton/chess-precomputation.

putation is particularly effective when the opposition plays predictably (r is small). As the randomness parameter increases, precomputation effectiveness decreases, and more states need to be memorized (the precomputation set size increases). In both plots, there is a transition point at $r = 100$ (corresponding to one center pawn) where the gain in playing more randomly against precomputation is offset by the loss in playing suboptimally. This suggests an interesting tradeoff between playing too predictably and playing the perceived best moves available.

Note that white is able to achieve generally higher precomputation values against black. This can be explained by two observations. The first is that white is generally considered to have an advantage against black, and so Stockfish with 50ms as black does not always win against Stockfish with 10ms as white[7]. The second is that black moves second, which means that black must remember more move variations than white to get to the same precomputation game depth in the game tree. This suggests that the first moving player has an inherent advantage when it comes to precomputation.

3.6 Chapter Conclusion

An unsolved problem in computer science is understanding how computationally bounded agents play games. We contribute to the understanding of this problem by presenting a novel formalism for modeling the precomputation aspect of computationally constrained agents, and show that in contrast to the computationally unconstrained setting, randomization plays an essential role for constructing effective strategies. This result is perhaps in opposition to the conventional heuristic of (deterministically) playing the strongest move found within the available time period. Moreover, we showed that optimal strategies in this model can be computed efficiently, and presented empirical results which suggest that precomputation strategies

[7]Note that even when r is close to 0, there can still be randomness in the move selection if the two best moves have equal center pawn scores.

are practically useful in real-world time constrained games such as chess. Our model is flexible, and is essentially equivalent to charging a player depending on which strategy (σ_i or σ_{pre}) they play from in each history. However, the model also has a fundamental structure in connection to the following question in computational complexity theory. Consider a game where there is a time limit of 1 second per move. Informally speaking, suppose we believe there is an algorithm σ_1 which achieves value at least v against any other algorithm σ_2 which also takes 1 second per move.[8] For example, maybe we believe that there is a chess engine σ_1 which takes 1 second per move, and always wins at least 10% of the time against any other chess engine σ_2 which also takes only 1 second per move. Whether such a σ_1 exists is a highly non-trivial problem. However, we can make one concrete observation. Because a precomputation strategy $\tilde{\sigma}_2 \in \text{Pre}(\sigma_{\text{pre}}, \sigma_2)$ takes no more time than one second per move, it follows that σ_1 must also have value at least v against $\tilde{\sigma}_2$, i.e. σ_1 must be robust against other algorithms hard-coding constants into their strategy and having non-uniform advice. Exploring this relationship in detail is an area for future work, but the observations about randomness being essential for competitive play are expected to carry over.

[8] Here we are informally ignoring the details of program size.

Chapter 4

Trading Off Resource Budgets For Improved Regret Bounds

4.1 Introduction

In the previous chapter, the compute budget was fixed, and we studied how varying memory (precomputation) interacted with the performance of algorithms. In contrast, in this chapter we study how varying a compute budget B interacts with performance. While we are primarily interested in thinking of B as a compute budget, we will set up our problem within the general framework of adversarial online learning. This will allow us to think about the budget B more abstractly, and allow the results to automatically transfer to a broader family of resource allocation problems. More broadly, this work fits within the family of work which studies computational constraints in the framework of online learning and bandit algorithms (Balcan et al., 2020; Kleinberg et al., 2017; Weisz et al., 2018; Kleinberg et al., 2019; Weisz et al., 2019). A more detailed overview of this work can be found in Section 2.1.

Adversarial online learning is a well-studied framework for sequential decision making with numerous applications. In each round $t = 1, \ldots, T$, an adversary chooses a hidden cost function $c_t : \mathcal{A} \to [0, 1]$ from a set of arms $\mathcal{A}$ to costs in $[0, 1]$. An algorithm must then choose an arm $a_t \in \mathcal{A}$, and incurs cost $c_t(a_t)$. In the *full feedback* setting (Online Learning with Experts (OLwE)), the algorithm then observes the

cost function c_t. In the *partial feedback* setting (Multi-Armed Bandits (MAB)) the algorithm only observes the incurred cost $c_t(a_t)$. The objective is to find algorithms which minimize *regret*, defined as the difference between the algorithm's cumulative cost and the cumulative cost of the single best arm in hindsight.

In this chapter we consider a search-like variant of these problems where in each round one can pick a *subset* of arms $S_t \subset \mathcal{A}$ with $|S_t| = B \geq 1$, and *keep the arm with the smallest cost*. This variant appears naturally in many settings, including:

1. Online algorithm portfolios (Gomes and Selman, 2001): In each round t, one receives a problem instance x_t, and can pick a subset $S_t \subset \mathcal{A}$ of algorithms to run in parallel to solve x_t. For example, x_t could be a boolean satisfiability (SAT) problem, and $\mathcal{A}$ could be a collection of different SAT solving algorithms. We let $c_t(a) = 0$ if a solves x_t and $c_t(a) = 1$ otherwise. Then if any $a \in S_t$ finds a solution to x_t we incur 0 cost in this round. Another example is online hyperparameter optimization (see Section 4.4). In this case, the budget B corresponds to the amount of compute available during each round.

2. Online bidding (Chen et al., 2016a): In each round t, an auctioneer sets up a first-price auction for bidders $S_t \subset \mathcal{A}$. Each bidder $a \in \mathcal{A}$ has a price $1 - c_t(a)$ they are willing to pay, and the auctioneer receives $\max_{a \in S_t} 1 - c_t(a) = 1 - \min_{a \in S_t} c_t(a)$, and so maximizing revenue is equivalent to minimizing costs.

3. Adaptive network routing (Awerbuch and Kleinberg, 2008): In each round t, a network router receives a data packet x_t and can pick a selection of network routes $S_t \subset \mathcal{A}$ to send it to its destination via in parallel. Let the cost $c_t(a)$ of a route a be the total time taken for x_t to reach its destination via a; the router receives cost $\min_{a \in S_t} c_t(a)$ equal to the smallest delay.

In many applications the budget B is a restricted resource (e.g. compute time or number of cores) we would like to keep small; in this chapter we study how one can

trade off budget resources for better guarantees on the standard regret objective.

Formally, for any randomized algorithm **ALG** which chooses subset $S_t \subset \mathcal{A}$ in round t, and thus incurs cost $c_t(S_t) := \min_{a \in S_t} c_t(a)$, define

$$R_T^*(\mathbf{ALG}) := \max_{c_1,\ldots,c_T} \mathbb{E}\left[\sum_{t=1}^{T} c_t(S_t) - \min_{a^* \in \mathcal{A}} \sum_{t=1}^{T} c_t(a^*)\right]$$

to be the worst-case expected regret of **ALG** relative to the single best arm in hindsight, where the expectation is with respect to the randomness of **ALG**.[1] What guarantees can we give on R_T^* as a function of our budget B? In the full feedback setting when $B = 1$, this is the standard OLwE problem where it is known that $R_T^* = \Omega(\sqrt{T})$ and there are algorithms which achieve $R_T^* \leq 2\sqrt{T \ln(N)}$ (Lattimore and Szepesvári, 2020), where $N = |\mathcal{A}|$. When $B = N$ the algorithm which chooses $S_t = \mathcal{A}$ in each round achieves $R_T^* = 0$. But what bounds on R_T^* can be achieved in the intermediate regime when $1 < B < N$? To the best of our knowledge this question has not been directly answered by any prior work.

4.1.1 Contributions

Theoretical results: We present a new algorithm for this learning problem called *Follow the Perturbed Multiple Leaders* (**FPML**), and show that in the full feedback setting $R_T^*(\mathbf{FPML}) \leq \mathcal{O}(T^{\frac{1}{B+1}} \ln(N)^{\frac{B}{B+1}})$. This allows for a direct tradeoff between the budget B and the regret bound (in particular, allowing resources $B \geq \Omega(\ln(T))$ leads to regret *constant* in T) and recovers the familiar $\mathcal{O}(\sqrt{T \ln(N)})$ bound when $B = 1$. We then show that in the *semi-bandit feedback* setting (where the algorithm finds out only the costs of the arms it chooses) this bound can be converted to $R_T^*(\mathbf{FPML}) \leq \mathcal{O}(T^{\frac{1}{B+1}}(K \ln(N))^{\frac{B}{B+1}})$ if one has unbiased cost estimators bounded in $[0, K]$.

We also consider the more general problem of Online Submodular Function Maximization (OSFM), for which prior work gives an online greedy algorithm **OG** (Streeter

[1] Here we consider an oblivious adversary model for simplicity, but we believe the results of this chapter carry through to adaptive adversaries as well.

and Golovin, 2008). When given a budget of B per round, **OG** achieves regret $\mathcal{O}(\sqrt{TB\ln(N)})$ with respect to $(1-e^{-1})\text{OPT}(B)$, where $\text{OPT}(B)$ is the performance of the best fixed length-B schedule (see Section 4.3 for a formal definition of OSFM). Note that in this guarantee the regret benchmark is a function of the algorithm budget. By replacing a subroutine in **OG** with **FPML**, we generalize **OG** to a new algorithm $\textbf{OG}_{\textbf{hybrid}}$. Unlike **OG**, $\textbf{OG}_{\textbf{hybrid}}$ is able to give regret bounds against benchmarks which are decoupled from the algorithm budget. This allows one to more easily quantify the tradeoff of increasing the budget against a fixed regret objective. As a special case, we are able to show that having a budget of $B = B'\lceil\ln(T)^2\rceil$ per round allows one to achieve regret $\mathcal{O}(B'\ln(T)\ln(N))$ with respect to $\text{OPT}(B')$. One interpretation of this result is that if you are willing to increase your budget (e.g. runtime) by a factor of $\ln(T)^2$, you are able to improve your performance guarantee benchmark from $(1-e^{-1})\text{OPT}(B')$ to $\text{OPT}(B')$. Likewise, your regret growth rate in terms of the number of rounds changes from $\mathcal{O}(\sqrt{T})$ to $\mathcal{O}(\ln(T))$.

Finally, in Section 4.5 we show how to use **FTML** to generalize a technique for solving linear programs assuming access to an oracle which solves a relaxed form of the linear program. To obtain a solution to the linear program which is within an ε tolerance of the linear constraints requires $\left(\frac{1}{\varepsilon}\right)^{\frac{B+1}{B}}(4\rho)^{\frac{B+1}{B}}(1+\ln(n))$ oracle calls, where the parameters (B,ρ) are related to the power of the oracle and n is the number of linear constraints. The case $B=1$ coincides with known results.

Experimental results: We benchmark both **FPML** and $\textbf{OG}_{\textbf{hybrid}}$ on an online black-box hyperparameter optimization problem based on the 2020 NeurIPS BBO challenge (Turner et al., 2021). We find that both these new algorithms outperform **OG** for various compute budgets. We are able to explain why this happens for this specific dataset, and discuss the scenarios under which each algorithm would perform better.

Techniques: Minimizing R_T^* is an important subroutine for a large variety of applications including Linear Programming, Boosting, and solving zero sum games (Arora et al., 2012). Traditionally an experts algorithm such as **Hedge** (Littlestone and Warmuth, 1994), which pulls a single arm per round, will be used as a subroutine to minimize R_T^*. We highlight how in the cases of OSFM and Linear Programming, one can simply replace a single arm R_T^*-minimizing subroutine with **FPML** and get performance bounds with little or no alteration to the original proofs. The resulting algorithms have improved bounds (due to improved bounds on R_T^* when $B > 1$) at the cost of qualitatively changing the algorithm (e.g. requiring a larger budget or more powerful oracle). This is significant because it highlights the potential of how bounds on R_T^* when $B > 1$ can lead to new results in other application areas. In Section 4.2.1 we also highlight how the proof techniques of Kalai and Vempala (2005) for bounding R_T^* in the traditional experts setting can naturally be generalized to the case when $B > 1$, which is of independent interest.

4.1.2 Relation To Prior Work In Online Learning

One can alternatively formulate the problem as receiving the maximum *reward* $r_t(a) = 1 - c_t(a)$ of each arm chosen instead of the minimum cost. In this maximum of rewards formulation, the problem fits within the OSFM framework where (a) all actions are unit-time and (b) the submodular *job* function is always a maximum of rewards. The rewards formulation of the problem has also been separately studied as the K-MAX problem (here $K = B$) Chen et al. (2016a). In the OSFM setting, Streeter and Golovin (2008) give an online greedy approximation algorithm which guarantees $\mathbb{E}[(1 - e^{-1})\text{OPT}(B) - \text{Reward}_T] \leq \mathcal{O}(\sqrt{TB\ln(N)})$ in the full feedback adversarial setting, where $\text{OPT}(B)$ is the cumulative reward of the best fixed subset of B arms in hindsight, and Reward_T is the cumulative reward of the algorithm. A similar bound of $\mathcal{O}(B\sqrt{TN\ln(N)})$ can be given in a semi-feedback setting. Conversely in

the full feedback setting, Streeter and Golovin (2007) show that any algorithm has worst-case regret $\mathbb{E}[\text{OPT}(B) - \text{Reward}_T] \geq \Omega(\sqrt{TB\ln(N/B)})$ when one receives the maximum of rewards in each round. Chen et al. (2016a) study the K-MAX problem and other non-linear reward functions in the stochastic combinatorial multi-armed bandit setting. Assuming the rewards satisfy certain distributional assumptions, they give an algorithm which achieves distribution-independent regret bounds of $\mathbb{E}[(1-\varepsilon)\text{OPT}(B) - \text{Reward}_T] \leq \mathcal{O}(\sqrt{TBN\ln(T)})$ for $\varepsilon > 0$ with semi-bandit feedback. Note however that we consider the adversarial setting in this paper.

More broadly, these problems fall within the combinatorial online learning setting where an algorithm may pull a subset of arms in each round. Much prior work has focused on combinatorial bandits where the reward is linear in the subset of arms chosen, which can model applications including online advertising and online shortest paths (Cesa-Bianchi and Lugosi, 2012; Audibert et al., 2014; Combes et al., 2015). The case of non-linear reward is comparatively less studied, but having non-linear rewards (such as max) allows one to model a wider variety of problems including online expected utility maximization (Li and Deshpande, 2011; Chen et al., 2016a). As some examples of prior work in the stochastic setting, (Gopalan et al., 2014) uses Thompson Sampling to deal with non-linear rewards of functions of subsets of arms (including the max function), but requires the rewards to come from a known parametric distribution. Chen et al. (2016b) consider a model where the subset of arms pulled is randomized based on pulling a 'super-arm', and the reward is a non-linear function of the values of the arms pulled. In the adversarial setting, Han et al. (2021) study the combinatorial MAB problem when rewards can be expressed as a d-degree polynomial.

In contrast to prior work which focuses on giving algorithms which compete against benchmarks which have *the same* budget as the algorithm, this work is concerned with the tradeoff between regret bounds and budget size. We focus on giving

regret bounds against OPT(1), and we use this result in Section 4.3 to get regret bounds against OPT(B') for $B' < B$ in OSFM. Decoupling the regret benchmark OPT(B') from the algorithm budget B can be useful when one would like to control the strength of a regret bound against a specific target OPT(B') for theoretical or applied reasons. For example Arora et al. (2012) survey a wide variety of applications which rely on bounding R_T^*, but bounds such as $\mathbb{E}[(1-e^{-1})\text{OPT}(B) - \text{Reward}_T] \leq \mathcal{O}(\sqrt{TB\ln(N)})$ do not immediately imply useful bounds on $\text{OPT}(1) - \text{Reward}_T$.

Update on related work: Following the publication of the work this chapter is based on, another work appeared which studies a closely related problem. In Bhaskara et al. (2023), the authors consider the setting where, in each round, before pulling an arm, one is able to probe an oracle on a subset of B arms and ask which arm has the lowest cost. The motivation behind this idea is that one may have e.g. a machine learning algorithm (oracle) which can provide advice about which arm is best to pick. The authors show, in the full feedback setting, that one can achieve regret $\mathcal{O}(\ln(N))$ for $B = 2$, which implies an improvement over the bound we give in Theorem 4.1. This is done by using a different analysis technique than the one used in this chapter, using ideas from differential privacy. However, the authors appear unable to use this technique to extend the result to the adversarial partial feedback case, as we do in Theorem 4.2. Note the partial feedback case is of particular interest from the perspective of compute constraints, because the only way to observe the cost of all arms is to pay the compute to evaluate all of them, which by assumption is infeasible. On the other hand, we believe the improved bounds in the full feedback case given in Bhaskara et al. (2023) make our contributions in sections 4.3 and 4.5 stronger, because the guarantees in those sections automatically improve when the regret minimizing subroutine has a tighter regret bound. Finally, we note from Section 4.4 that the relationship between analytical regret bounds and empirical performance can be non-

obvious; it can therefore be useful to have multiple approaches to compare with. This suggests an area for future work exploring (1) the relative strengths of different regret analysis techniques and their corresponding algorithms and (2) whether improved results similar to those in sections 4.3 and 4.5 can be given.

4.2 Follow The Perturbed Multiple Leaders

We begin by considering the full feedback setting. We first check that allowing the algorithm to choose $B > 1$ arms per round, while only competing against the best *single* fixed arm in hindsight, does not make the problem trivial. We do this by showing that any deterministic algorithm with budget $B < N$ still achieves linear regret in the number of rounds. This is achieved by setting $c_t(a) = 1$ if $a \in S_t$, $c_t(a) = 0$ otherwise.

Proposition 4.1. *In the full feedback setting, any deterministic algorithm with arm budget $B \leq N$ per round has worst-case regret $R_T^* \geq \left(1 - \frac{B}{N}\right) T$.*

Proof. Let $S_t \subset \mathcal{A}$ be the deterministic choice of arms the bandit algorithm chooses for round t given the previously chosen cost functions $c_1, \ldots, c_{t-1}$. Now choose $c_t(a) = 1$ if $a \in S_t$, and $c_t(a) = 0$ otherwise. Then the bandit algorithm achieves cost T. The total cost summed over all arms is $\sum_{t=1}^{T} |S_t| \leq BT$, so there must exist at least one $a' \in \mathcal{A}$ such that $\sum_{t=1}^{T} c_t(a') \leq \frac{BT}{N}$. Thus $R_T^* \geq T - \frac{BT}{N} = (1 - \frac{B}{N})T$. □

Likewise, it can be shown that the algorithm which chooses a uniformly random subset of B arms in each round has worst-case expected regret at least $T(1 - \frac{B}{N})$ (achieved by having one arm have cost 0 across all rounds and every other arm having cost 1). These two observations show that any solution for achieving sub-linear regret in T requires randomization which depends in some non-trivial way on the prior observed costs even when $B > 1$.

4.2.1 Generalizing Follow The Perturbed Leader

Choosing the current lowest perturbed-cost arm in each round, *Follow the Perturbed Leader* (**FPL**) (Kalai and Vempala, 2005), is a well-known regret minimization technique which achieves optimal worst-case regret against adaptive adversaries in the OLwE setting. In this section we generalize the **FPL** algorithm to *Follow the Perturbed Multiple Leaders* (**FPML**). In each round, **FPML** perturbs the cumulative costs of each arm by adding noise, and then picks the B arms with lowest cumulative perturbed cost. This is precisely **FPL** when $B = 1$. We show how one can extend the proof techniques of Kalai and Vempala (2005) in a natural way to prove that **FPML** achieves worst-case regret $R_T^*(\textbf{FPML}) \leq 2T^{\frac{1}{B+1}}(1+\ln(N))^{\frac{B}{B+1}}$. In what follows, we denote the distribution of a standard exponential random variable scaled by the factor $\frac{1}{\varepsilon}$ by $\frac{1}{\varepsilon}$Exp.

Algorithm 3: FPML(B,ε)

Accept as input $N \geq B \geq 1, \varepsilon > 0$;
Initialize the cumulative cost $C_0(a) \leftarrow 0$ for each arm $a \in \mathcal{A}$;
for *round* $t = 1, \ldots, T$ **do**
    For each arm $a \in \mathcal{A}$, draw a noise perturbation $p_t(a) \sim \frac{1}{\varepsilon}$ Exp;
    Calculate the perturbed cumulative costs for round $t-1$, $\tilde{C}_{t-1}(a) \leftarrow C_{t-1}(a) - p_t(a)$;
    Pull the B arms with the lowest perturbed cumulative costs according to $\tilde{C}_{t-1}$. Break ties arbitrarily;
    Update the cumulative costs for each arm, $C_t(a) \leftarrow C_{t-1}(a) + c_t(a)$;

Theorem 4.1. *In the full feedback setting, where $S_t \subset \mathcal{A}$ is the subset of arms chosen by **FPML** in round t, we have*

$$\max_{c_1,\ldots,c_T} \mathbb{E}\left[(1-\varepsilon^B)\sum_{t=1}^{T} c_t(S_t) - \min_{a^* \in \mathcal{A}} \sum_{t=1}^{T} c_t(a^*)\right] \leq \frac{(1+\ln(N))}{\varepsilon}.$$

In particular, for $\varepsilon = ((\ln(N)+1)/T)^{1/(B+1)}$, we have

$$R_T^*(\textbf{FPML}) \leq 2T^{\frac{1}{B+1}}(1+\ln(N))^{\frac{B}{B+1}}.$$

The proof follows the same three high level steps which appear in Kalai and Vempala (2005) for **FPL**, but we extend these ideas to the case where $B > 1$. We first observe that the algorithm which picks the B lowest cumulative cost arms in each round only incurs regret when the best arm in round t is not one of the best B arms in round $t-1$.

Lemma 4.1. *Consider a fixed sequence of cost functions $c_1, \ldots, c_T$. Let $a_t^{*,j}$ be the j^{th} lowest cumulative cost arm in hindsight after the first t rounds, breaking ties arbitrarily. Let $S_t^* := \{a_{t-1}^{*,j} \mid j \in [B]\}$ be the set of the B lowest cost arms at the end of round $t-1$. Then for each $i \in [T]$,*

$$R_i := \sum_{t=1}^{i} c_t(S_t^*) - \min_{a^* \in \mathcal{A}} \sum_{t=1}^{i} c_t(a^*) \leq \sum_{t=1}^{i} \mathbb{1}[a_t^{*,1} \notin S_t^*]$$

and $R_i - R_{i-1} \leq \mathbb{1}[a_i^{*,1} \notin S_i^*]$.

This is a generalization of the familiar result that when $B = 1$, following the leader has regret bounded by the number of times the leader is overtaken (Kalai and Vempala, 2005).

Proof. Let $R_i = \sum_{t=1}^{i} c_t(S_t^*) - \min_{a^* \in \mathcal{A}} \sum_{t=1}^{i} c_t(a^*)$ be the regret at the end of round i. Then the increase in regret in round i is

$$\begin{aligned}
r_i &:= R_i - R_{i-1} \\
&= \sum_{t=1}^{i} \left(c_t(S_t^*) - c_t(a_i^{*,1})\right) - \sum_{t=1}^{i-1} \left(c_t(S_t^*) - c_t(a_{i-1}^{*,1})\right) \\
&= c_i(S_i^*) - c_i(a_i^{*,1}) + \left(\sum_{t=1}^{i-1} c_t(a_{i-1}^{*,1}) - \sum_{t=1}^{i-1} c_t(a_i^{*,1})\right) \\
&\leq c_i(S_i^*) - c_i(a_i^{*,1}) \\
&\leq \mathbb{1}[a_i^{*,1} \notin S_i^*]
\end{aligned}$$

and the result follows by evaluating $\sum_{t=1}^{T} r_t$. □

The second step is to argue that if the cumulative costs are perturbed slightly, it becomes unlikely that the event $\{a_t^{*,1} \notin S_t^*\}$ will occur. One way to see this is as follows: fix a round t, and let $C_{t-1}(a)$ be the cumulative cost of a at the end of round $t-1$. Let $M \coloneqq C_{t-1}(a^{*,B+1})$. Then every $a \in S_t^*$ has $C_{t-1}(a) \leq M$. If it is also true that $C_{t-1}(a) < M - c_t(a)$ for any $a \in S_t^*$ then the event $\{a_t^{*,1} \notin S_t^*\}$ cannot occur. This is because $C_t(a) < (M - c_t(a)) + c_t(a) = M$ but any $a' \in \mathcal{A} - S_t$ has $C_t(a') \geq M$, so $a' \neq a_t^{*,1}$. If we had initially perturbed each $C_{t-1}(a)$ by subtracting independent exponential noise $p(a) \sim \frac{1}{\varepsilon}\text{Exp}$, then conditional on M (and randomizing over the noise) the event $\{C_{t-1}(a) < M - c_t(a)\}$ is jointly independent for each $a \in S_t^*$. Moreover the probability of this inequality not holding is equal to $\mathbb{P}[p(a) < v + c_t(a) | p(a) \geq v]$ for v equal to the unperturbed cost of a at round $t-1$ minus M, which is bounded by $\varepsilon c_t(a)$ (due to the memorylessness property of the exponential distribution).

Lemma 4.2. *Fix a sequence of cost functions $c_1, \ldots, c_T$. Let $C_i(a) = \sum_{t=1}^{i} c_t(a)$ and $\tilde{C}_i(a) = C_i(a) - p(a)$ be the perturbed cumulative cost of arm a at the end of round i, where $p(a) \sim \frac{1}{\varepsilon}$Exp. Let $\tilde{a}_t^{*,j}$ be the j^{th} lowest cumulative cost arm in hindsight after the first t rounds using these perturbed costs $\tilde{C}_i$, and let $\tilde{S}_t^* = \{\tilde{a}_{t-1}^{*,j} \mid j \in [B]\}$. Then*

$$\mathbb{E}\left[\mathbb{1}\left[\tilde{a}_t^{*,1} \notin \tilde{S}_t^*\right]\right] \leq \mathbb{E}\left[\varepsilon^B c_t(\tilde{S}_t^*)\right].$$

Again, when $B = 1$ this argument and bound coincides with the argument given by Kalai and Vempala (2005).

Proof. Fix a round t. Consider the jointly independent random variables $X_a = \tilde{C}_{t-1}(a)$ for $a \in \mathcal{A}$. Condition on the values and identities of the $N - B$ largest of these random variables, i.e. condition on $E = \{(X_{a_{t-1}^{*,j}}, a_{t-1}^{*,j})\}_{j=B+1}^{N}$, and let $M = X_{a_{t-1}^{*,B+1}}$ be the minimum perturbed cost among these non-leading arms. Impose an ordering on $\mathcal{A}$ and let $l_1, \ldots, l_B \in \mathcal{A} - \{a^{*,j}\}_{j=B+1}^{N}$ be the remaining arms (the top B leaders) ordered lexicographically (i.e. not necessarily in order of cumulative perturbed

cost). Then the distribution of the random variables $X_{l_1}, \ldots, X_{l_B}$ conditioned on E is jointly independent, and the marginal distribution of X_{l_j} given E is $X_{l_j}|(X_{l_j} \leq M)$ (see Lemma B.1). Now observe that if $X_{l_j} < M - c_t(l_j)$ for any $j \in [B]$, then the event $(\tilde{a}_t^{*,1} \notin \tilde{S}_t^*)$ is impossible. This is because $l_j \in \tilde{S}_t^*$, but for any $a \notin S_t^*$, $\tilde{C}_t(a) \geq M$ but $\tilde{C}_t(l_j) = X_{l_j} + c_t(l_j) < M$ (i.e. l_j cannot be overtaken by any non-top-B-leader in round t). Therefore we have

$$\mathbb{E}\left[\mathbb{1}[\tilde{a}_t^{*,1} \notin \tilde{S}_t^*]|E\right] \leq \mathbb{P}\left[\bigwedge_{j=1}^{B} \neg(X_{l_j} < M - c_t(l_j)) \;\middle|\; E\right] \tag{4.1}$$

$$= \prod_{j=1}^{B} \mathbb{P}\left[\neg(X_{l_j} < M - c_t(l_j))|X_{l_j} \leq M\right] \tag{4.2}$$

$$= \prod_{j=1}^{B} \left(1 - \mathbb{P}\left[p(l_j) > C_{t-1}(l_j) + c_t(l_j) - M | p(l_j) \geq C_{t-1}(l_j) - M\right]\right) \tag{4.3}$$

$$\leq \prod_{j=1}^{B} \left(1 - \mathbb{P}\left[p(l_j) > c_t(l_j)\right]\right) \tag{4.4}$$

$$= \prod_{j=1}^{B} \left(1 - \int_{c_t(l_j)}^{\infty} \varepsilon e^{-\varepsilon x} dx\right) = \prod_{j=1}^{B} (1 - e^{-\varepsilon c_t(l_j)}) \tag{4.5}$$

$$\leq \prod_{j=1}^{B} (\varepsilon c_t(l_j)) \tag{4.6}$$

$$\leq \varepsilon^B c_t(\tilde{S}_t^*) \tag{4.7}$$

(4.2) follows by independence, (4.4) is due to the memorylessness property of the exponential distribution (with equality unless $C_{t-1}(l_j) - M < 0$), and (4.7) follows because $1 - e^{-x} \leq x$ for $x \geq 0$ and $c_t(S_t^*) \geq \prod_{a \in S_t^*} c_t(a) = \prod_{j=1}^{B} c_t(l_j)$. The final claim follows by taking the expectation over the conditioned event E. □

The final step is to combine Lemmas 4.1 and 4.2 to argue that **FPML** achieves expected regret at most $\mathbb{E}\left[\varepsilon^B \sum_{t=1}^{T} c_t(\tilde{S}_t^*)\right]$ with respect to the perturbed cumulative cost $\tilde{C}_T$. Since $\max_{a \in \mathcal{A}} \mathbb{E}[p(a)] \leq \frac{1+\ln(N)}{\varepsilon}$ we can argue we also achieve low expected

regret with respect to the unperturbed cost C_T. In the setting of this paper, drawing new random perturbations $p_t(a)$ in each round is not strictly necessary (we can take $p_t(a) = p_1(a)$ for $t > 1$), but it is necessary to achieve regret bounds when cost functions can depend on prior arm choices of the algorithm (the *adaptive* adversarial setting). In the setting of this paper where the costs are fixed, the expected regret in either case is the same.

Proof of Theorem 4.1

Proof. Consider a modified version of **FPML** where $p_t(a) = p_1(a) = p(a)$ for all $t > 1$ (i.e. we keep the random perturbation fixed across rounds). Then this version of **FPML** picks the set $\tilde{S}_t^*$ in round t, and the regret can be bounded as

$$\sum_{t=1}^{T} c_t(\tilde{S}_t^*) - \min_{a^* \in \mathcal{A}} \sum_{t=1}^{T} c_t(a^*) \leq \sum_{t=1}^{T} c_t(\tilde{S}_t^*) - \min_{\tilde{a}^* \in \mathcal{A}} \left(\left(\sum_{t=1}^{T} c_t(\tilde{a}^*) \right) - p(\tilde{a}^*) \right) \tag{4.8}$$

The right hand side can be interpreted as the regret of a modified version of OLwE with a 0^{th} round with cost function $c_0 = -p$, where the algorithm is only allowed to pull arms from round $t = 1$ (and incurs cost 0 in the 0th round). Note that modifying the cost function in this way gives us perturbed cumulative costs precisely of the form in Lemma 4.2. The regret increase incurred in the 0^{th} round is at most $\max_{a \in \mathcal{A}} p(a)$. For the remaining rounds, we use Lemma 4.1 followed by Lemma 4.2 to get

$$\mathbb{E}\left[\sum_{t=1}^{T} c_t(S_t^*) - \min_{a^* \in \mathcal{A}} \sum_{t=1}^{T} c_t(a^*) \right] \leq \sum_{t=1}^{T} \mathbb{E}\left[\mathbb{1}[\tilde{a}_t^{*,1} \notin \tilde{S}_t^*] \right] + \mathbb{E}\left[\max_{a \in \mathcal{A}} p(\tilde{a}) \right] \tag{4.9}$$

$$\leq \varepsilon^T \, \mathbb{E}\left[\sum_{t=1}^{T} c_t(S_t^*) \right] + \frac{(1 + \ln(N))}{\varepsilon} \tag{4.10}$$

Where the inequality on $\mathbb{E}\left[\max_{a \in \mathcal{A}} p(\tilde{a})\right]$ comes from Kalai and Vempala (2005). The final step is to argue that the unmodified version of **FPML** which chooses independent noise $p_t(a)$ in each round also achieves this bound. This is immediate because

both versions of the algorithm have the same marginal distribution of action choices in each round. By linearity of expectation, they also have the same expected cost across all the rounds, and therefore the same expected regret. Having new random perturbations in each round is therefore not necessary against oblivious adversaries, but is necessary to achieve the regret bound against adaptive adversaries. □

Probabilistic guarantees: One advantage of this proof technique is that the regret is bounded using the positive random variable $\sum_{t=1}^{T} \mathbb{1}\left[\tilde{a}_t^{*,1} \notin \tilde{S}_t^*\right]$. This means that one can apply methods like Markov inequality to give a probabilistic guarantee of small regret, which is substantially stronger than saying the regret is small in expectation.

Comments on settings of parameters: When $B = 1$ we recover the standard $\mathcal{O}(\sqrt{T \ln(N)})$ regret bound for the OLwE problem. For $B > 1$, the regret growth rate as a function of the number of rounds is $T^{\frac{1}{B+1}}$. In particular, when $B = \Omega(\ln(T))$ grows slowly with the number of rounds, the expected regret becomes $\mathcal{O}(\ln(N))$ and does not grow with the number of rounds T. If we use a tighter inequality in the proof of Lemma 4.2, it is possible to get constant expected regret when $B = \ln(T)\ln(A)$ grows slowly with the number of arms and rounds.

Significance of bound: How significant is the bound of Theorem 4.1? From a theoretical perspective, we now know from future work that one only requires $B = 2$, not $B = \mathcal{O}(\log(T))$, to achieve regret bounds which are constant in T in the full feedback setting. On the other hand, from a practical perspective in the context of online algorithm portfolios, Theorem 4.1 says that one only needs to pay an extra $\mathcal{O}(\log(T))$ factor in runtime in order to reduce the regret scaling by a factor of $\sqrt{T}$. Since there are many contexts in which log factors of runtime are not considered significant, Theorem 4.1 gives a significant regret improvement for a very modest

increase in runtime.

Lower bounds: A standard technique for constructing lower bounds in the online experts setting with $B = 1$ is to consider costs which are i.i.d. Bernoulli(p) (Lattimore and Szepesvári, 2020). Unfortunately this technique fails when $B > 1$ because the expected cost of the minimum of B i.i.d. Bernoulli random variables is generally smaller than the expected cost of the best arm in hindsight unless p is very close to 1. On the other hand, work following the publication of this work by Bhaskara et al. (2023) imply that the lower bound for $B \geq 2$ is at most $\mathcal{O}(\ln(N))$. Below we give a simple proof that this is indeed the lower bound for constant B in the full feedback setting. The idea is to maintain a set of "alive" arms, initially equal to $[N]$. In each round, we randomly choose half of the alive arms to have cost 0, and the other half to have cost 1. All non-alive arms also incur cost 1. For any fixed choice of arms by the algorithm in the current round, there is then some constant probability (if B is constant and the number of alive arms is sufficiently large) that the algorithm only chooses arms with cost 1 in this round. After the round, we set the "alive" arms to be those arms which have only ever incurred 0 cost in all prior rounds. Thus for each round, the expected cost of the algorithm is a constant, and we can ensure this for $\mathcal{O}(\log_2(N))$ rounds (because we can only halve the number of "alive" arms this many times). We will ensure there is always one alive arm which always incurs cost 0, so that the expected regret of the algorithm is equal to its expected cost. We now make this argument formal.

Proposition 4.2. *In the full feedback setting, any randomized algorithm has* $R_T^* \geq \Omega\left(\left(\frac{1}{4}\right)^B (\log_2(N) - \log_2(B))\right)$ *for* $T \geq \Omega(\log_2(N) - \log_2(B))$.

Proof. First suppose $N = 2^k$ for some $k \in \mathbb{N}$. Let $\mathcal{A}_0 = \mathcal{A}$. In round $t = 1, \ldots, T$, we let $\mathcal{A}_t$ be a uniformly randomly chosen subset of $\mathcal{A}_{t-1}$ of size $\max(2^{k-t}, 1)$, and we let $c_t(a) = 0$ if $a \in \mathcal{A}_t$, 1 otherwise. Note that the probability of a fixed arm

$a \in A_{t-1}$ being contained in A_t given that i other arms in A_{t-1} are not in A_t is $\frac{|\mathcal{A}_t|}{|\mathcal{A}_{t-1}|-i}$. Suppose an algorithm chooses arms $S_t = \{a_t^1, \ldots, a_t^B\} \subset \mathcal{A}$ in round t. Let $t \leq k - \log_2(B) - \log_2\left(\frac{3}{2}\right)$. Then we have

$$\mathbb{P}[S_t \cap A_t = \emptyset] = \prod_{i=1}^{B} \mathbb{P}[a_t^i \notin \mathcal{A}_t | a_t^1, \ldots a_t^{i-1} \notin \mathcal{A}_t] \tag{4.11}$$

$$\geq \prod_{i=0}^{B-1} \left(1 - \frac{|\mathcal{A}_t|}{|\mathcal{A}_{t-1}| - i}\right) \tag{4.12}$$

$$\geq \left(1 - \frac{|\mathcal{A}_t|}{|\mathcal{A}_{t-1}| - B}\right)^B \tag{4.13}$$

$$= \left(1 - \frac{1}{2 - B/|\mathcal{A}_t|}\right)^B \tag{4.14}$$

$$\geq \left(1 - \frac{3}{4}\right)^B \tag{4.15}$$

$$= \left(\frac{1}{4}\right)^B \tag{4.16}$$

where line 4.14 holds provided $t \leq k$, and line 4.14 holds provided $|\mathcal{A}_t| \geq \frac{3}{2}B$. These two conditions are satisfied when $t \leq k - \log_2(B) - \log_2\left(\frac{3}{2}\right)$. If we set $T = \lfloor k - \log_2(B) - \log_2\left(\frac{3}{2}\right) \rfloor$, then the expected cost of any algorithm ALG is $\geq \left(\frac{1}{4}\right)^B T = \Omega\left(\left(\frac{1}{4}\right)^B (\log_2(N) - \log_2(B))\right)$. By construction, the cost of the best expert in hindsight is 0. Since the expected regret is $\Omega\left(\left(\frac{1}{4}\right)^B (\log_2(N) - \log_2(B))\right)$, there exists fixed cost functions $c_1, \ldots, c_T$ such that the expected regret of ALG on this on this sequence is $\Omega\left(\left(\frac{1}{4}\right)^B (\log_2(N) - \log_2(B))\right)$. If N is not a power of 2, we can just let $\mathcal{A}_0 \subset \mathcal{A}$ be any subset of size $2^{\lfloor \log_2(N) \rfloor}$ and the asymptotic bounds remain the same. □

Partial feedback: The *semi-bandit feedback* setting is a form of partial feedback where the algorithm only observes the individual costs of the arms it pulls. It can be shown that passing unbiased cost function estimates to **FPML** results in a similar regret bound in the semi-bandit feedback setting; the result is specific to **FPML**

and using unbiased cost functions does not generally work for any R_T^*-minimizing algorithm when $B > 1$ because of the non-linearity of the cost function. In the case of **FPML** this is not an issue because the same bounding technique using Lemma 4.1 holds in expectation when using unbiased cost estimators. A naïve way to generate unbiased cost estimates in this setting is to use an additional arm to uniformly sample costs; in Section 4.4 we explore geometric sampling (Neu and Bartók, 2013) for getting unbiased cost estimates which is effective in practice.

Theorem 4.2. *Let $\mathcal{F}_t$ be the σ-algebra generated by all the outcomes occurring during rounds up to and including round t.*[2] *Define the algorithm **FPML-partial** which simulates **FPML**, passing it unbiased cost estimates $\hat{c}_t \in [0, K]$ at round $t \in [T]$ where $\forall a \in \mathcal{A}, \mathbb{E}[\hat{c}_t(a) \mid \mathcal{F}_{t-1}] = c_t(a)$, and copies the arm choices of **FPML**. Then for any $\varepsilon > 0$ we have*

$$R_T^*(\textbf{\textit{FPML-partial}}) \leq \ln(N)/\varepsilon + T(1 - \mathrm{e}^{-K\varepsilon})^B$$

$$\textit{For } \varepsilon = \left(\frac{\ln(N)}{TK^B}\right)^{\frac{1}{B+1}},\ R_T^*(\textbf{\textit{FPML-partial}}) \leq \mathcal{O}(T^{\frac{1}{B+1}}(K\ln(N))^{\frac{B}{B+1}}).$$

Random perturbations: Note that, unlike in Theorem 4.1, in Theorem 4.2 it is important for our proof that **FPML** draws independent random samples $p_t(a)$ in each round. This is because the unbiased cost estimates $\hat{c}_t$ may depend on prior action choices of the algorithm, and hence on the random perturbations used in prior rounds. This can lead to non-trivial situations. For example, if **FPML** used a single random perturbation $p(a) = p_t(a)$ for all rounds t, it is then possible for the unbiased cost estimates $\hat{c}_t$ to behave in an adversarial way to ensure that after some round t_0, **FPML** will always choose an arm with cost 1 even though the best arm in hindsight achieves an average cost of 0.5. This would prevent us from arguing that in any fixed round t, provided $|\hat{c}_t| \leq K$, the expected increase in regret during round t is small.

[2] This includes the random perturbations sampled by the algorithm up to round t, and the unbiased cost estimates up to round t.

At a high level, the way to construct this adversarial case is to choose unbiased cost functions $\hat{c}_t$ in a way which, with some constant probability, forces the cumulative perturbed costs of the arms to be approximately equal at some round t_0. This may require the unbiased cost estimates to have a fairly large variance during rounds $t \leq t_0$ (i.e. $|\hat{c}_t| = \Omega\left(\frac{1}{\varepsilon}\right)$ with some small probability). One then picks $\hat{c}_t = c_t$ for all rounds $t > t_0$ (i.e. $K = 1$ during all subsequent rounds). Suppose there are $N = 2$ arms, and arm 1 has $c_t(1) = 1$ for t odd and $c_t(1) = 0$ for t even, and arm 2 has $c_t(2) = 1 - c_t(1)$. It is then possible to force **FPML** to alternate between picking arms 1 and 2, and always pick the arm with cost 1 for rounds $t > t_0$.

We can prove Theorem 4.2 in two steps. The first step is to reduce the problem to the case where **FPML** draws a single random perturbation $p = p_t$ for all rounds $t \geq 1$, and the unbiased cost estimates $\hat{c}_1, \ldots, \hat{c}_t$ are jointly independent of p. When the unbiased cost estimates are independent of the random perturbation, we no longer have the issue of the cost estimates potentially behaving in an adversarial way as discussed in the paragraph above. The second step uses a very similar analysis to the proof of Theorem 4.1, but some subtlety is required when dealing with the random cost estimates. The key property which makes the result possible is that for any $a \in \mathcal{A}$

$$\mathbb{E}[c_i(S_i) - \hat{c}_i(a) \mid \mathcal{F}_{t-1}] = c_i(S_i) - c_i(a) \tag{4.17}$$

$$\leq \mathbb{1}[a \notin S_i] \tag{4.18}$$

i.e. in expectation, we can bound $c_i(S_i) - \hat{c}_i(a_i^*)$ by $\mathbb{1}[a_i^* \notin S_i]$.

Lemma 4.3. *In the setting of Theorem 4.2, let $S(\hat{c}_{:t-1}, p_t)$ be the arm choices of **FPML** after observing cost estimates $\hat{c}_{:t-1} = \hat{c}_1, \ldots, \hat{c}_{t-1}$ and drawing random perturbation p_t in round t. Then*

$$\mathbb{E}\left[\sum_{t=1}^{T} c_t\left(S(\hat{c}_{:t-1}, p_t)\right)\right] = \mathbb{E}_{p_0}\left[\mathbb{E}\left[\sum_{t=1}^{T} c_t\left(S(\hat{c}_{:t-1}, p_0)\right) \mid p_0\right]\right]$$

where the random perturbation $p_0(a) \sim \frac{1}{\varepsilon}$*Exp for* $a \in \mathcal{A}$ *is drawn independently of all other events.*

Proof. Let $p_0(a) \sim \frac{1}{\varepsilon}\text{Exp}$ for $a \in \mathcal{A}$ be perturbations drawn independently from all other events. By repeated application of linearity of expectation, we have

$$\mathbb{E}\left[\sum_{t=1}^{T} c_t\left(S(\hat{c}_{:t-1}, p_t)\right)\right] = \sum_{t=1}^{T} \mathbb{E}\left[c_t\left(S(\hat{c}_{:t-1}, p_t)\right)\right] \tag{4.19}$$

$$= \sum_{t=1}^{T} \mathbb{E}_{p_0}\left[\mathbb{E}\left[c_t\left(S(\hat{c}_{:t-1}, p_0)\right) \mid p_0\right]\right] \tag{4.20}$$

$$= \mathbb{E}_{p_0}\left[\mathbb{E}\left[\sum_{t=1}^{T} c_t\left(S(\hat{c}_{:t-1}, p_0)\right) \mid p_0\right]\right] \tag{4.21}$$

where line 4.20 follows because p_t is drawn independently of $\hat{c}_{:t-1}$ and c_t. □

Lemma 4.3 is subtle and worth discussing. Note that $\hat{c}_t$ can be thought of as a random function of the past actions of **FPML** and the past cost estimates $\hat{c}_1, \ldots, \hat{c}_{t-1}$, i.e. $\hat{c}_t$ is a random function of $p_1, \ldots, p_{t-1}, \hat{c}_1, \ldots, \hat{c}_{t-1}$. Line 4.21 is the formal expectation which samples the random variables $\hat{c}_1, p_1, \ldots, \hat{c}_t, p_t$, then samples p_0, and then evaluates $\sum_{t=1}^{T} c_t\left(S(\hat{c}_{:t-1}, p_0)\right)$. This expectation also happens to be equal to the expected cost of **FPML** where (1) we first sample $\hat{c}_1, p_1, \ldots, \hat{c}_t, p_t$, (2) we sample p_0, and (3) we run **FPML** using the same perturbation p_0 across all rounds and pass it the cost estimates $\hat{c}_1, \ldots, \hat{c}_T$.

Proof of Theorem 4.2

Proof. Fix deterministic cost functions $c_1, \ldots, c_T$. By Lemma 4.3, it suffices to consider the case where **FPML** re-uses random perturbations between rounds, i.e. $p_t(a) = p(a)$ for all $a \in \mathcal{A}, t \in [T]$, and the cost estimates $(\hat{c}_1, \ldots, \hat{c}_T)$ are jointly independent of p. As in the proof of Theorem 4.1, imagine that there is a 'round zero' with costs $(-p(a))_{a \in \mathcal{A}}$ where $p(a) \sim \text{Exp}(\varepsilon)$ independently for each a; define

$\mathcal{F}_0 \coloneqq \sigma((p(a))_{a\in\mathcal{A}}$ to be the σ-algebra generated by these perturbations and include it in each $(\mathcal{F}_t)_{t\geq 1}$.

Writing $\hat{C}_i(\cdot) \coloneqq \sum_{t=1}^{i} \hat{c}_t(\cdot)$ for cumulative estimated reward and $\hat{C}_i^*(\cdot) \coloneqq \hat{C}_i(\cdot) - p(\cdot)$ for the same but including the 'round zero' random initialization, define

$$R_i' \coloneqq \sum_{t=1}^{i} c_t(S_t) - \min_{a\in\mathcal{A}} \hat{C}_i^*(a)$$

for each $i \in [T] \cup \{0\}$. Let S_i be the set of arms chosen by the algorithm at round i, and let a_i^* be the best of these by perturbed estimated cost at the end of round i. We follow the argument from Lemma 4.1:

$$\begin{aligned} R_i' - R_{i-1}' &= c_i(S_i) - \hat{c}_i(a_i^*) + \left(\sum_{t=1}^{i-1} \hat{c}_t(a_{i-1}^*) - \sum_{t=1}^{i-1} \hat{c}_t(a_i^*)\right) && (4.22)\\ &\leq c_i(S_i) - \hat{c}_i(a_i^*) && (4.23) \end{aligned}$$

and so

$$\begin{aligned} \mathbb{E}[R_i' - R_{i-1}' \mid \mathcal{F}_{i-1}] &\leq c_i(S_i) - \mathbb{E}[\hat{c}_i(a_i^*) \mid \mathcal{F}_{i-1}] && (4.24)\\ &= c_i(S_i) - c_i(a_i^*) && (4.25)\\ &\leq \mathbb{1}[a_i^* \notin S_i]. && (4.26) \end{aligned}$$

Hence

$$\begin{aligned} \mathbb{E}[R_T' \mid \mathcal{F}_0] &= \mathbb{E}\left[\sum_{t=1}^{T} \mathbb{E}[R_t' - R_{t-1}' \mid \mathcal{F}_{t-1}] + R_0' \mid \mathcal{F}_0\right] && (4.27)\\ &\leq \mathbb{E}\left[\sum_{t=1}^{T} \mathbb{1}[a_t^* \notin S_t] + R_0' \mid \mathcal{F}_0\right] && (4.28)\\ &\leq \mathbb{E}[|\mathcal{I}| \mid \mathcal{F}_0] + \max_{a\in\mathcal{A}} p(a), && (4.29) \end{aligned}$$

where $\mathcal{I} \coloneqq \{t \in [T] : a_t^* \notin S_t\}$. Using Jensen's inequality in line 4.32 below, we have

$$\mathbb{E}[R'_T \mid \mathcal{F}_0] = \mathbb{E}\left[\sum_{t=1}^{T} c_t(S_t) \mid \mathcal{F}_0\right] + \mathbb{E}\left[-\min_{a\in\mathcal{A}} \hat{C}_T^*(a) \mid \mathcal{F}_0\right] \tag{4.30}$$

$$= \mathbb{E}\left[\sum_{t=1}^{T} c_t(S_t) \mid \mathcal{F}_0\right] + \mathbb{E}\left[\max_{a\in\mathcal{A}} -\hat{C}_T^*(a) \mid \mathcal{F}_0\right] \tag{4.31}$$

$$\geq \mathbb{E}\left[\sum_{t=1}^{T} c_t(S_t) \mid \mathcal{F}_0\right] + \max_{a\in\mathcal{A}} \mathbb{E}[-\hat{C}_T^*(a) \mid \mathcal{F}_0] \tag{4.32}$$

$$= \mathbb{E}\left[\sum_{t=1}^{T} c_t(S_t) \mid \mathcal{F}_0\right] - \min_{a\in\mathcal{A}} \mathbb{E}[\hat{C}_T^*(a) \mid \mathcal{F}_0] \tag{4.33}$$

$$= \mathbb{E}\left[\sum_{t=1}^{T} c_t(S_t) \mid \mathcal{F}_0\right] - \min_{a\in\mathcal{A}} (C_T(a) - p(a)) \tag{4.34}$$

$$= \mathbb{E}\left[\sum_{t=1}^{T} c_t(S_t) \mid \mathcal{F}_0\right] - C_T(a^*) + p(a^*) \tag{4.35}$$

where a^* is the best-in-hindsight arm at the end of round T. Therefore we can bound the expected regret by

$$R_T^* \leq \mathbb{E}\left[\mathbb{E}[|\mathcal{I}| \mid \mathcal{F}_0] + \max_{a\in\mathcal{A}} p(a) - p(a^*)\right] \tag{4.36}$$

Since $\mathbb{E}[p(a^*)] = 1/\varepsilon$ for any fixed action a^*, and $\max_{a\in\mathcal{A}} p(a)$ is the maximum of $|\mathcal{A}|$ i.i.d. $\mathrm{Exp}(\varepsilon)$ random variables and therefore has expectation at most $(1+\ln|\mathcal{A}|)/\varepsilon$ as argued in Kalai and Vempala (2005), taking expectations gives

$$\mathbb{E}[R_T^*] \leq \frac{\ln|\mathcal{A}|}{\varepsilon} + \mathbb{E}[|\mathcal{I}|] \tag{4.37}$$

It remains to upper-bound $\mathbb{E}[|\mathcal{I}|]$. The proof ideas are very similar to those in Lemma 4.2, so we will state the result here and include its proof in the appendix.

Lemma 4.4.

$$E[|\mathcal{I}|] \leq T(1 - \mathrm{e}^{-K\varepsilon})^B \tag{4.38}$$

Proof. See Appendix B.1. □

The result then follows, since

$$\mathbb{E}[R_T^*] \leq \frac{\ln|\mathcal{A}|}{\varepsilon} + \mathbb{E}[|\mathcal{I}|] \tag{4.39}$$

$$\leq \frac{\ln|\mathcal{A}|}{\varepsilon} + T(1-\mathrm{e}^{-K\varepsilon})^B \tag{4.40}$$

□

4.3 Generalized Regret Bounds For Online Submodular Function Maximization

The purpose of this section is to show how the results of Section 4.2 can be used to construct new algorithms to other resource allocation problems in a relatively automatic way. We turn our attention to the problem of Online Submodular Function Maximization (OSFM) first studied by Streeter and Golovin (2008), which captures a number of online resource allocation problems as special cases. For the sake of emphasizing the key ideas, we will restrict attention to the full feedback setting where each action has unit duration; however we believe the same ideas can be used without these assumptions. In this simplified setting, the OSFM problem is as follows:

Definition 4.1. *Define a* schedule *to be a finite* ***ordered*** *sequence of actions*[3] $a \in \mathcal{A}$*, and let* $\mathcal{S}$ *be the set of all schedules. The* length $\ell(S)$ *of a schedule* $S \in \mathcal{S}$ *is the number of actions it contains. Define a* job *to be a function* $f : \mathcal{S} \to [0,1]$ *such that for any schedules* $S_1, S_2 \in \mathcal{S}$ *and any action* $a \in \mathcal{A}$*:*

1. $f(S_1) \leq f(S_1 \oplus S_2)$ *and* $f(S_2) \leq f(S_1 \oplus S_2)$ ***(monotonicity)****;*
2. $f(S_1 \oplus S_2 \oplus \langle a \rangle) - f(S_1 \oplus S_2) \leq f(S_1 \oplus \langle a \rangle) - f(S_1)$ ***(submodularity)****.*

Definition 4.2 (Online Submodular Function Maximization)**.** *The problem consists of a game with* T *rounds. We are given some fixed budget* $B > 0$ *and at each round*

[3] In the full OSFM problem, actions also have a different associated *duration*.

$t \in [T]$ we must choose a schedule $S_t \in \mathcal{S}$ with $\ell(S_t) \leq B$ to be evaluated by a job f_t which is only revealed after our choice. The goal is to maximize the cumulative output Reward$_T := \sum_{t=1}^{T} f_t(S_t)$.

Streeter and Golovin (2008) propose an online greedy algorithm **OG** which achieves the guarantee $(1 - e^{-1})\text{OPT}(B) - \text{Reward}_T \leq \mathcal{O}(\sqrt{TB\ln(N)})$ in expectation, where $\text{OPT}(B)$ is the cumulative reward of the best fixed schedule of length B in hindsight. In this section we explain how to use **FPML** to extend their algorithm to allow a tradeoff between budget resources and regret bounds.

The algorithm **OG** has two key ideas. Suppose that we start with an empty schedule $S_0 \coloneqq \emptyset$. The first idea is that if we knew $f_1, \ldots, f_T$ in advance, we could greedily construct $S_i \coloneqq S_{i-1} \oplus \langle \arg\max_{a_i \in \mathcal{A}} \sum_{t=1}^{T} f_t(S_{i-1} \oplus \langle a_i \rangle) - f_t(S_{i-1}) \rangle$ for $i = 1, \ldots, B$, where a_i is the best greedy arm in hindsight for greedy round i. It can be shown that submodularity then implies $(1 - e^{-1})\text{OPT}(B) \leq \sum_{t=1}^{T} f(S_B)$. Since we don't know $f_1, \ldots, f_T$ in advance, the second idea is to run B copies of a traditional, single arm pulling R_T^* regret minimizer, where the i^{th} copy tries to compete with achieving the same *improvement* of cumulative reward as the best fixed greedy action a_i in hindsight. The regret bound on the i^{th} copy with respect to the best greedy arm in hindsight is $\mathcal{O}(\sqrt{T\ln N})$; across the B copies one can show the net regret compared to the offline greedy solution is bounded by $\mathcal{O}(\sqrt{TB\ln N})$, which is where the final bound comes from. In summary, **OG** works as follows: for each round $t \in [T]$, run B greedy rounds. In greedy round $i = 1, \ldots, B$, pull the arm a_t^i proposed by the i^{th} black box R_T^*-minimizer, set $S_{t,i} \coloneqq S_{t,i-1} \oplus \langle a_t^i \rangle$, and feed back the greedy rewards $r_{t,i}(a) = f_t(S_{t,i-1} \oplus \langle a \rangle) - f_t(S_{t,i-1})$ to the i^{th} black box.

Algorithm 4: OG$(B, \mathcal{E})$

Accept as input $B \geq 1$;
Let $\mathcal{E}_1, \ldots, \mathcal{E}_B$ be instances of a regret minimization algorithm $\mathcal{E}$;
for *rounds* $t = 1, \ldots, T$ **do**
  Let $S_{t,0} = \langle\rangle$ be the empty schedule;
  for $i = 1, \ldots, B$ **do**
    Use $\mathcal{E}_i$ to choose an action $a_t^i \in \mathcal{A}$;
    Set $S_{t,i} \coloneqq S_{t,i-1} \oplus \langle a_t^i \rangle$;
  Set $S_t \coloneqq S_{t,B}$; receive the job f_t;
  For each $i \in [B]$ and each action $a \in \mathcal{A}$ feed back the cost $c_t^{(i)}(a) \coloneqq 1 - (f_t(S_{t,i-1} \oplus \langle a \rangle) - f_t(S_{t,i-1}))$ to regret minimization algorithm $\mathcal{E}_i$;

We propose a hybrid version of **OG**, called $\mathbf{OG_{hybrid}}$, which for any budget B allows us to compete asymptotically well against $\mathrm{OPT}(B')$ for any chosen $B' \leq B/\ln(T)$. The algorithm $\mathbf{OG_{hybrid}}$ is based on the following two changes to **OG**:

1. Instead of having B greedy rounds, we have $B'\ln(T)$ greedy rounds. One can show that the extra factor of $\ln(T)$ allows one to drop the $(1 - e^{-1})$ term in the regret bound. To do this, we need a slight extension of a result from Streeter and Golovin (2008), where they prove the following lemma for $B_1 = B_0$ (proof in Appendix B.1).

   Lemma 4.5. *Let f be any job and let $\bar{G} = \langle \bar{g}_1, \bar{g}_2, \ldots, \bar{g}_{B_1} \rangle$ be a 'greedy' schedule satisfying*

   $$f(\bar{G}_j \oplus \bar{g}_j) - f(\bar{G}_j) \geq \max_{a \in \mathcal{A}} \big(f(\bar{G}_j \oplus \langle a \rangle) - f(\bar{G}_j)\big) - \varepsilon_j, \qquad \forall j \in [B_1]$$

   where $\varepsilon_1, \varepsilon_2, \ldots \geq 0$ are additive errors and $\bar{G}_j = \langle \bar{g}_1, \ldots, \bar{g}_{j-1} \rangle$ for each $j \in [B_1]$. Then for any $B_0 \in \mathbb{N}$,

   $$f(\bar{G}) > \big(1 - \mathrm{e}^{-B_1/B_0}\big)\, f(S^*_{B_0}) - \sum_{j=1}^{B_1} \varepsilon_j$$

   where $S^*_{B_0} \coloneqq \arg\max_{S \in \mathcal{S}: \ell(S) = B_0} f(S)$ is the best schedule of length B_0 for f.

2. Instead of running a one-arm-pulling R^*_T minimizer in each greedy round, we run **FPML** which pulls $\lceil B/B'\ln(T) \rceil$ arms. This allows us to improve the regret

bound for each of the R_T^*-minimizers and directly translates to a tighter overall regret bound. This is due to the following lemma, which is a slight extension of a result from Streeter and Golovin (2008) (proof in Appendix B.1).

Lemma 4.6. *For $B \geq B' \log T$, the algorithm **OG** produces a sequence of schedules $S_1, \ldots, S_T$ with expected regret*

$$\mathbb{E}\left[\sum_{t=1}^{T} f_t(S^*_{B'}) - \sum_{t=1}^{T} f_t(S_t)\right] = \mathcal{O}\left(\mathbb{E}\left[\sum_{j=1}^{B} R_{T,1}(\mathcal{E}_j)\right]\right)$$

*relative to $S^*_{B'} \coloneqq \arg\max_{S \in \mathcal{S}:\ell(S)=B'} \sum_{t=1}^{T} f_t(S)$, the best-in-hindsight fixed schedule of length B', where $R_{T,1}(\mathcal{E}_j)$ is the 1-regret incurred by the j^{th} experts algorithm, i.e. its regret relative to the single best fixed arm in hindsight on the cost sequence given to it by **OG**.*

The key idea here is that the regret of **OG** is directly related to the regret of the subroutines $\{\mathcal{E}_j\}_j$. Traditionally one would use single arm pulling regret minimizers to solve this problem, which has regret at least $\Omega(\sqrt{T})$. But if we plug in multi-arm pulling regret minimizers instead, we can trade off the budget we give to each of these subroutines in return for stronger bounds on $R_{T,1}(\mathcal{E}_j)$, which automatically imply stronger bounds for OSFM.

Besides these two changes, the algorithm is identical to that in Streeter and Golovin (2008). More generally, we let $\mathbf{OG_{hybrid}}(B, \tilde{B})$ denote the algorithm where each **FPML** box has a budget of $\tilde{B}$ and there are $B/\tilde{B}$ greedy rounds, so that the total number of arms pulled in each round is B. This algorithm is a hybrid of **OG** and **FPML** in the sense that $\mathbf{OG_{hybrid}}(B, 1)$ is **OG** with *Follow the Perturbed Leader* as the R_T^*-minimizing subroutine. On the other hand, $\mathbf{OG_{hybrid}}(B, B)$ is **FPML**. Varying $\tilde{B}$ allows us to interpolate between these two algorithms by varying the budget we give to the **FPML** subroutines.

Algorithm 5: $\mathbf{OG_{hybrid}}(B,\widetilde{B})$

Accept as input $B \geq \widetilde{B} \geq 1$ where $\widetilde{B} \mid B$. Define $L \coloneqq B/\widetilde{B}$;
Let $\mathcal{B}_1, \ldots, \mathcal{B}_L$ be instances of **FPML**, each with budget $\widetilde{B}$ and $\varepsilon = ((\ln(N)+1)/T)^{1/(\widetilde{B}+1)}$;
for *rounds* $t = 1, \ldots, T$ **do**
    Let $S_{t,0} = \langle\rangle$ be the empty schedule;
    for $i = 1, \ldots, L$ **do**
        Use $\mathcal{B}_i$ to choose $\widetilde{B}$ actions $a_t^{i,1}, a_t^{i,2}, \ldots, a_t^{i,\widetilde{B}}$;
        Set $S_{t,i} \coloneqq S_{t,i-1} \oplus \langle a_t^{i,1}, \ldots a_t^{i,\widetilde{B}} \rangle$;
    Set $S_t \coloneqq S_{t,L}$ and receive the job f_t;
    for $i = 1, \ldots, L$ **do**
        For each action $a \in \mathcal{A}$ feed back the cost $c_t^{(i)}(a) \coloneqq 1 - (f_t(\langle a_t^{1,*}, \ldots, a_t^{i-1,*}, a\rangle) - f_t(\langle a_t^{1,*}, \ldots, a_t^{i-1,*}\rangle))$ to instance $\mathcal{B}_i$;
        Define $a_t^{i,*} \coloneqq \arg\min_{j \in [\widetilde{B}]} c_t^{(i)}(a_t^{i,j})$;

On a technical note, because we are pulling $\tilde{B} \geq 1$ arms for each greedy choice, we require a slight strengthening on the monotonicity condition which is common to many practical applications including the experiments we consider in the next section:

Assumption 4.1. *In addition to monotonicity and submodularity, each job $f : \mathcal{S} \to [0,1]$ also satisfies $f(S_1 \oplus S_2 \oplus S_3) \geq f(S_1 \oplus S_3)$ for any schedules $S_1, S_2, S_3 \in \mathcal{S}$.*

Using this assumption, we can then give the following guarantees for $\mathbf{OG_{hybrid}}(B, \widetilde{B})$.

Theorem 4.3. *For any $B', \tilde{B} \in \mathbb{N}$, under Assumption 4.1 and with budget $B \coloneqq \widetilde{B}\lceil B' \ln(T)\rceil$, algorithm $\boldsymbol{OG_{hybrid}}(B,\tilde{B})$ experiences expected regret*

$$\mathcal{O}\left(B' \ln(T) T^{1/(\widetilde{B}+1)} \ln(N)^{\widetilde{B}/(\widetilde{B}+1)}\right)$$

relative to the best-in-hindsight fixed schedule of length B'. In particular, if $\widetilde{B} = \lceil \ln(T) \rceil$ (and so $B = B'\lceil \ln(T)\rceil^2$) the regret is bounded by $\mathcal{O}(B' \ln(T) \ln(N))$.

This bound allows us the flexibility of trading off a schedule budget B for how tightly we would like to compete with the best fixed schedule of length $B' \leq B$ in hindsight.

Proof. Let $L = B/\widetilde{B}$ be as in $\mathbf{OG_{hybrid}}$. Suppose for each $i \in [L]$ there is a fictional experts algorithm $\mathcal{E}_i$ which picks the single arm $a_t^{i,*}$ (defined in line 11 of $\mathbf{OG_{hybrid}}$) at each round t, and consider a hypothetical instance of the standard algorithm **OG** run with budget L and these fictional experts algorithms $\mathcal{E}_1, \ldots, \mathcal{E}_L$ as subroutines. Note this is just **OG** where each expert $\mathcal{E}_i$ happens to pick the lowest cost arm found by the ith black box of $\mathbf{OG_{hybrid}}$ in round t. Note also that these instances of **OG** and $\mathbf{OG_{hybrid}}$ pass the same cost functions to their black box regret minimizers. We will argue that (1) the regret of $\mathbf{OG_{hybrid}}$ is bounded by the regret of this hypothetical instance of **OG** and (2) the regret of this hypothetical instance of **OG** satisfies the bounds in Theorem 4.3. To avoid confusion, we will refer to the l-regret as the regret of an algorithm relative to the best fixed subset (schedule) of size (length) l in hindsight, given some cost or payoff function sequence.

Since $L \geq B' \log T$ (by our assumption that $B \geq B'\widetilde{B} \log T$), by Lemma 4.6 the B'-regret (relative to the optimal schedule of length B') of our **OG** instance is upper-bounded in expectation by $\sum_{i=1}^{L} R_{T,1}(\mathcal{E}_i)$, where $R_{T,1}(\mathcal{E}_i)$ is the total 1-regret experienced by $\mathcal{E}_i$ for the costs given to it by **OG**. Moreover, the payoff received by this **OG** instance at each round t is $f(\langle a_t^{1,*}, \ldots, a_t^{L,*} \rangle)$. Let S_t be the schedule chosen by $\mathbf{OG_{hybrid}}$ in round t. Then by Assumption 4.1 and monotonicity, it follows that $f(S_t) \geq f(\langle a_t^{1,*}, \ldots, a_t^{L,*} \rangle)$, because the actions $a_t^{1,*}, \ldots, a_t^{L,*}$ are contained in the schedule S_t with the same relative order. Therefore, the B'-regret of $\mathbf{OG_{hybrid}}$ relative to $\mathrm{OPT}(B')$ must be at most that of our fictional **OG** instance, giving an upper bound on the B'-regret of $\mathbf{OG_{hybrid}}$ as

$$R_{B'}(\mathbf{OG_{hybrid}}) \leq \sum_{i=1}^{L} \mathbb{E}[R_{T,1}(\mathcal{E}_i)]. \tag{4.41}$$

It remains to argue how large the 1-regret of each of these 'fictional' experts algorithms $\mathcal{E}_i$ is. Writing $a_i^{**} = \arg\min_{a \in \mathcal{A}} \sum_{t=1}^{T} c_t^{(i)}(a)$ for the best-in-hindsight fixed action under the costs passed to the ith regret minimizer, the regret incurred by $\mathcal{E}_i$

is therefore

$$R_{T,1}(\mathcal{E}_i) = \sum_{t=1}^{T} c_t^{(i)}(a_t^{i,*}) - \sum_{t=1}^{T} c_t^{(i)}(a_i^{**}) \tag{4.42}$$

$$= \sum_{t=1}^{T} \max_{j\in[\widetilde{B}]} c_t^{(i)}(a_t^{i,j}) - \sum_{t=1}^{T} c_t^{(i)}(a_i^{**}) \tag{4.43}$$

$$= R_{T,1}(\mathcal{B}_i) \tag{4.44}$$

where $R_{T,1}(\mathcal{B}_i)$ is the 1-regret incurred by multiple arm pulling bandit algorithm $\mathcal{B}_i$ in $\mathbf{OG_{hybrid}}$. So by equation 4.41, the B'-regret is bounded by

$$\sum_{i=1}^{L} \mathbb{E}[R_{T,1}(\mathcal{B}_i)] = L\mathbb{E}[R_{T,1}(\mathcal{B})] \tag{4.45}$$

$$= \frac{B\mathbb{E}[R_{T,1}(\mathcal{B})]}{\widetilde{B}} \tag{4.46}$$

where $\mathbb{E}[R_{T,1}(\mathcal{B})]$ is the expected 1-regret of any of the instances $\mathcal{B}_1, \ldots, \mathcal{B}_L$ of $\mathcal{B}$. By Theorem 4.1, since each instance of $\mathcal{B}$ has budget $\widetilde{B}$, we know that setting $\varepsilon = ((\ln(N)+1)/T)^{1/(\widetilde{B}+1)}$ for **FPML** gives

$$\mathbb{E}[R_{T,1}(\mathcal{B})] \leq 2T^{\frac{1}{\widetilde{B}+1}}(1+\ln(N))^{\frac{\widetilde{B}}{\widetilde{B}+1}} \tag{4.47}$$

from which Theorem 4.3 follows. □

Partial feedback: Streeter and Golovin (2008) extend their algorithm to handle partial feedback by replacing the R_T^* regret minimizers with the bandit algorithm **Exp3** (Auer et al., 2002) which only requires feedback on the arms which are pulled. Likewise, one can replace **FPML** with **FPML-partial** in $\mathbf{OG_{hybrid}}$ to get an algorithm which gives regret bounds in a semi-bandit feedback setting (where we can observe $f_t(S)$ for any schedule S consisting of actions which were pulled in round t). We empirically compare the partial feedback versions of **OG** and $\mathbf{OG_{hybrid}}$ in the next section.

4.4 Experiments: Online Hyperparameter Optimization

The problem of *black-box optimization*—where a hidden function is to be minimized using as few evaluations as possible—has recently generated increased interest in the context of hyperparameter selection in deep learning (Snoek et al., 2012; Liu et al., 2020b; Bouthillier and Varoquaux, 2020). As a result, the 2020 NeurIPS BBO Challenge (Turner et al., 2021) invited participants' optimizers to compete to find the best possible configurations of several ML models on a number of common datasets, given a limited budget of training cycles for each. One of the key findings was that sophisticated new algorithms are normally outperformed on average by techniques that ensemble existing methods (Liu et al., 2020a), i.e. optimizers tend to have varying strengths and weaknesses that are suited to different task types. In a scenario where many hyperparameter selection problems are to be processed (e.g. in a data center) and limited computing resources are available, it may thus be desirable to learn over time how best to choose B optimizers to apply independently to each problem (e.g. to run in parallel on B available CPU cores). This is a natural partial feedback application of bandit algorithms that pull multiple arms per round and receive the best score found across the optimizers which are run.

4.4.1 Experimental Setup

We follow a similar approach to the NeurIPS BBO Challenge; the optimization problem at each round $t \in [T]$ is to choose the hyperparameters of either a multi-layer perceptron (MLP) or a lasso classifier for one of 184 classification tasks from the Pembroke Machine Learning Benchmark (Olson et al., 2017) (so $T = 368$). At each round the bandit algorithm must select B Bayesian black-box optimizers from a choice of 9 to run in parallel on the current problem; the received *reward*[4] for each optimizer

[4]Here we use rewards to be consistent with the setting of Online Submodular Function Maximization.

was calculated as a $[0, 1]$-normalized measure of where the best training loss attained sits between **(a)** the expected best loss from a random hyperparameter search and **(b)** an estimate of the best possible loss attainable (the same approach used by the Bayesmark package (Uber, 2020)). Rewards are only observed for the optimizers which are run (semi-bandit feedback). For each budget level $B = 1, \ldots, 6$, we ran a benchmarking study comparing the performance of the following bandit algorithms in this setting:

1. **FPML-partial**(B) from Section 4.2.1, using geometric sampling (Neu and Bartók, 2013) to construct unbiased cost estimates. Geometric resampling constructs a cost estimate by first sampling a_t from **FPML-partial**, observing the cost $c_t(a_t)$, and then estimating the probability probability $\hat{p}_{a_t}$ with which **FPML-partial** chose action a_t by random sampling. It then passes the unbiased cost estimate $\hat{c}_t(a) = \frac{1}{\hat{p}_{a_t}} c_t(a_t)$ if $a = a_t$, $\hat{c}_t(a) = 0$ otherwise. In practice, we need to introduce some bias to this estimate to ensure the cost estimates $\hat{c}$ are bounded. A discussion of this can be found in Appendix B.2.

2. $\mathbf{OG}_{\mathbf{hybrid}}(B, \tilde{B})$ from Section 4.3, with **FPML-partial** as a subroutine. We do this for varying values of $\tilde{B}$ to see how performance changes.

3. **OG**(B), the partial feedback version of the original online greedy algorithm from Streeter and Golovin (2008).

We benchmark performance against **BIH**(B), the score of the best fixed subset of size B in Hindsight. In all cases the ε parameter for the bandit subroutines **FPML-partial** and **Exp3** was set to their theoretically optimal values given B, N and T without any fine-tuning (for **FPML-partial** we use Proposition 4.2). We run each algorithm setting 100 times to estimate the mean and standard deviation of the performance.

4.4.2 Results And Discussion

Table 4.1: Experimental results for $B = 6$ and $B = 4$.

$B = 6$		
Algorithm	Mean Reward	StD
BIH(6)	0.901	NA
FPML-partial(6)	0.888	0.0072
$\mathbf{OG_{hybrid}}(6, 3)$	0.836	0.0111
$\mathbf{OG_{hybrid}}(6, 2)$	0.814	0.0143
$\mathbf{OG_{hybrid}}(6, 1)$	0.785	0.0137
OG	0.767	0.0157
$B = 4$		
Algorithm	Mean Reward	StD
BIH(4)	0.836	NA
FPML-partial(4)	0.813	0.0108
$\mathbf{OG_{hybrid}}(4, 2)$	0.756	0.0149
$\mathbf{OG_{hybrid}}(4, 1)$	0.716	0.0178
OG(4)	0.689	0.0151

As expected, $\mathbf{OG_{hybrid}}(B, 1)$ has very similar performance to $\mathbf{OG}(B)$ in all cases because they implement essentially the same algorithm (the difference being due to different choices of one-arm pulling R_T^* minimizers, **FPML-partial**(1) and **Exp3**). However, we notice that in every instance, allocating more budget to the **FPML-partial** subroutine in $\mathbf{OG_{hybrid}}$ (increasing $\tilde{B}$) while keeping the overall budget constant improved the average score performance. This means that $\mathbf{OG_{hybrid}}$ was always at least as good as **OG** for all parameter settings, and that **FPML-partial** outperformed both of these algorithms in all cases. This is perhaps surprising, because **OG** and $\mathbf{OG_{hybrid}}$ are designed to achieve low regret against the stronger benchmark of the best subset of $B' \geq 1$ arms in hindsight, while **FPML-partial** is only designed to achieve low regret with respect to the single best optimizer in hindsight. Towards explaining this observation, we notice that for this particular dataset, the best subset of B arms in hindsight happens to be very similar to the set of the individual best

performing B arms in hindsight. Moreover, we believe one can show **FPML** achieves low regret with with respect to the latter, using the same techniques as those in Section 4.2.[5] This raises the interesting question of when certain regret objectives might be better than others in practice. Future work might focus on evaluating different regret benchmarks empirically, and deriving new algorithms for the benchmarks which appear to perform best in practice.

Synthetic tasks: We also evaluated these partial feedback algorithms in a number of synthetic environments, exploring examples where **(a)** the optimal subset of B arms, **(b)** the subset of B arms chosen greedily, **(c)** the individual best performing B arms, perform in various ways relative to each other; **OG** approximates **(b)** and **FPML-partial** approximates **(c)**, so the closeness of either of these to **(a)** determines each algorithm's performance. We find that problem instances exist where **OG** outperforms **FPML-partial** and vice versa.

Further experimental details can be found in Appendix B.2.

4.5 Connections To Other Problems

Arora et al. (2012) survey a wide variety of algorithmic problems and shows how they can all be solved using a R_T^* minimizing subroutine in the standard OLwE setting with $B = 1$. The purpose of this section is to highlight the relative ease with which algorithms like **FPML** can sometimes be *plugged in* to existing algorithms which use an R_T^* minimizer as black box subroutine, with little or no alteration to their proofs of correctness. We already saw an example of this in Section 4.3, where the regret minimizing subroutine was replaced with **FPML**. The resulting algorithm gave improved regret guarantees at the cost of higher budget requirements, without changing the structure of the original proof given in Streeter and Golovin (2008). In

[5] We do not do this here, but Orton and Falck (2022) discusses this idea in more detail.

this section we take Linear Programming as an example, and illustrate what its plug-in algorithm looks like. We are unsure whether the resulting algorithm is necessarily useful because it requires a more powerful oracle than the one supposed in Arora et al. (2012); but we do think that exploring the algorithms which result from this plug-in technique more broadly may be an area of interest for future work.

4.5.1 Linear Programming

We consider the Linear Programming (LP) problem from Arora et al. (2012): Given a convex set P, an $n \times m$ matrix A with entries in $\mathbb{R}$, and a vector $b \in \mathbb{R}^n$, the task is to find an $x \in P$ such that $Ax \geq b$, or determine that no such x exists. We assume that we have an oracle which solves the following easier problem (where A_i denotes the i^{th} row of A):

Definition 4.3. *A (ρ, B)-bounded oracle for $\rho \geq 0$ is an algorithm, which when given a joint distribution d over $[n]^B$, finds an $x \in P$ such that*

$$\mathop{\mathbb{E}}_{(i_1,\ldots,i_B)\sim d}\left[\min_{i\in\{i_1,\ldots,i_B\}} A_i x - b_i\right] \geq 0$$

or determines that no such x exists. If an x is found, then $\forall i \in [n], |A_i x - b_i| \leq \rho$.

When $B = 1$ this is a simplified version of the oracle defined in Arora et al. (2012), and it is equivalent to an oracle which can find $x \in P$ which satisfies a single linear constraint $d^\top Ax \geq d^\top b$ (when viewing d as a vector in $\mathbb{R}^n$). When $B = n$, solving this problem can be as hard as solving the original LP problem by jointly choosing i_j to have probability mass 1 on the j^{th} linear constraint. $1 < B < n$ represents an intermediate regime where the oracle needs to find an x which 'fools' the joint distribution d. For simplicity, we assume that we have access to an oracle which takes as input bounded cost functions $c_1, \ldots, c_{t-1}$, and outputs the joint distribution d_t over arms of **FPML** in round t after observing cost functions $c_1, \ldots, c_{t-1}$. In practice such an oracle could be achieved by e.g. sampling arm draws from **FPML** to approximate d_t.

Proposition 4.3. *Let $\varepsilon > 0$. Suppose there exists a (ρ, B)-bounded oracle for the feasibility problem $\exists x \in P$ s.t. $Ax \geq b$. Then there is an algorithm which either finds an $x \in P$ s.t. $\forall i \in [n], A_i x \geq b_i - \varepsilon$, or correctly concludes that the problem is infeasible. The algorithm makes at most $T = \left(\frac{1}{\varepsilon}\right)^{\frac{B+1}{B}} (4\rho)^{\frac{B+1}{B}} (1 + \ln(n))$ calls to the (ρ, B)-bounded oracle and FPML oracle, with a total runtime of $\mathcal{O}(T)$.*

The proof technique for general $B \geq 1$ is essentially identical to the proof given in Arora et al. (2012) for $B = 1$ except that we are able to use a stronger bound on R_T^*.

Proof. We run the FPML oracle with budget B, $N = n$ arms, and $\varepsilon = ((\ln(N) + 1)/T)^{1/(B+1)}$. In round $t \in [T]$ we do the following: Let d_t be the joint distribution over N arms returned the FPML oracle in this round. We pass d_t to the (ρ, B)-bounded oracle, and receive either a vector $x_t \in P$ or that no x_t exists which satisfies the oracle problem. Let us first suppose that we always receive an x_t for each round. Then define the cost function $c_t(i) := A_i x_t - b_i \in [-\rho, \rho]$ and pass this to the FPML oracle. After T rounds, and by scaling and translating the cost functions to lie in $[0, 1]$, Theorem 4.1 implies that $\forall j \in [N]$

$$\frac{\sum_{t=1}^{T} \mathbb{E}_{(i_1,\ldots,i_B)\sim d_t} \left[\min_{i\in\{i_1,\ldots,i_B\}} A_i x - b_i\right]}{T} \leq \frac{4\rho T^{\frac{1}{B+1}} (1 + \ln(N))^{\frac{B}{B+1}}}{T} + \frac{\sum_{t=1}^{T} A_j x_t - b_j}{T}$$

By assumption of the (ρ, B)-bounded oracle, the left hand side is ≥ 0. When $T \geq \left(\frac{1}{\varepsilon}\right)^{\frac{B+1}{B}} (4\rho)^{\frac{B+1}{B}} (1+\ln(N))$, it follows that $x := \frac{\sum_{t=1}^{T} x_t}{T}$ satisfies $\forall j \in [N], A_j x \geq b_j - \varepsilon$. Since P is convex, $x \in P$ and we are done. Now suppose that in some round t we were told the oracle problem was not solvable. We claim that we can conclude that the problem is not feasible and we are done. This is because if $\exists x \in P$ s.t. $Ax \geq b$, then $\mathbb{E}_{(i_1,\ldots,i_B)\sim d} \left[\min_{i\in\{i_1,\ldots,i_B\}} A_i x - b_i\right] \geq \mathbb{E}_{(i_1,\ldots,i_B)\sim d} \left[\min_{i\in\{i_1,\ldots,i_B\}} 0\right] = 0$ and so the oracle problem would be solvable. □

Comment on bound: When $B = 1$, we require $\Omega((\frac{1}{\varepsilon})^2)$ steps in order to find an x which is ε-close to satisfying the constraint. The $(\frac{1}{\varepsilon})^2$ term comes from the fact that when $B = 1$, the average regret for OLwE is $\Omega(\sqrt{T}/T) = T^{-\frac{1}{2}}$, so it takes $T \geq (\frac{1}{\varepsilon})^2$ steps for the average regret to be $\leq \varepsilon$. The quadratic dependence on an accuracy parameter is therefore common in many applications which use OLwE with $B = 1$ as a subroutine (including Boosting (Schapire, 1990) and solving zero sum games (Freund and Schapire, 1999)). For general $B \geq 1$, we only require $\mathcal{O}((\frac{1}{\varepsilon})^{\frac{B+1}{B}})$ steps (suppressing terms related to n and ρ) for the average regret to be $\leq \varepsilon$. In the case of Linear Programming, this is at the expense of requiring a stronger oracle for the problem.

4.6 Chapter Conclusion

This chapter presented a new algorithm, Follow the Perturbed Multiple Leaders, which allows one to directly trade off budget constraints for bounds on regret. We showed how **FPML** can be used as a subroutine to generate new algorithms for Online Submodular Function Optimization and Linear Programming which trade off resources and oracle power for improved performance guarantees. In the specific case where the budget is computational, this chapter gives a new family of algorithms for online problems which trade off compute constraints for performance guarantees. We also highlight two areas for future work:

1. **Plug-in algorithms.** Are there other cases where using **FPML** in existing algorithms can lead to new theoretical results? And when are these new algorithms practically useful?

2. **Which regret benchmarks are useful in practice?** The experiments of Section 4.4 showed that algorithms designed to minimize regret with respect to a single arm can sometimes in practice outperform algorithms designed to

minimize regret with respect to the stronger benchmark of the best subset of arms. Are certain regret objectives better than others in different practical applications?

Chapter 5

Lifelong Learning

Models in computational learning theory (see Kearns and Vazirani (1994)) and reinforcement learning (RL) (see Sutton and Barto (2018)) often define problems as finite, static, and specified by an idealized statistical model. Sometimes these modelling choices can contrast sharply with the way computationally bounded agents actually observe and interact with the world. For example:

1. To an agent with bounded knowledge, new experiences will appear non-stationary even if they come from a well defined stochastic process.

2. To an agent with bounded processing time, it may be intractable to find a solution which fits all observed data. Rather, the agent must try to solve computationally tractable subproblems, building up its knowledge until more difficult problems are within reach.

In large unstructured environments, computationally bounded agents often need to continually learn as the problem changes or more data becomes available, and not rely on idealizations such as data being independently and identically distributed.

Lifelong learning (LL) (see Chen and Liu (2018)) is typically concerned with how to learn in these less structured environments, where an algorithm continually comes across new problems and needs to update its knowledge as it learns. A detailed overview of LL can be found in Section 2.2. Interest in LL has picked up over the last

few years, and the field is still in an early stage of development. There are a number of fairly varied techniques for specific problems in the field, but no unified understanding of "what knowledge is" or "how to think about incremental learning". The primary contribution of this chapter is to attempt to bring some clarity to these questions. We do this by giving incremental learning algorithms which can combine knowledge without distributional assumptions (Section 5.1), building explicit theoretical models of incremental learning which can be analyzed precisely (Section 5.2), and constructing a domain agnostic framework for lifelong reinforcement learning algorithms whose high level learning behaviour can be understood theoretically (Section 5.3). We also discuss more concrete models of incremental knowledge learning in Section 5.4. The chapter is organised as follows:

1. In Section 5.1 we abstractly model the problem of incrementally adding knowledge to a system. We show that by controlling the rate at which new knowledge is added, we can additively accumulate knowledge in a relatively robust way.

2. In Section 5.2, we construct and study a model for incremental problem solving. The model gives a unified way of thinking about how to acquire knowledge and incrementally solve new problems which is domain agnostic. It is built by extending and formalizing ideas in program synthesis from prior work (Ellis et al., 2018, 2020).

3. In Section 5.3, we combine the algorithms in the previous two sections to give a framework for building lifelong reinforcement learning algorithms. The framework allows one to create algorithms whose high level learning behaviour can be understood theoretically. The framework also allows one to plug in existing reinforcement learning algorithms as black boxes, and combine these algorithms in an interpretable way to give a lifelong learning algorithm. This means that we can receive the benefit of e.g. practical neural network architectures found

by practitioners, even if these neural networks are not well understood from a theoretical perspective.

4. In Section 5.4, we give a short discussion of a proposal for a lifelong object-relation world model. It is inspired by a fusion of two recent model based reinforcement learning approaches (Kipf et al., 2020; Hafner et al., 2023).

Sections 5.1 and 5.2 can be read in isolation as independent contributions. Section 5.3 directly depends on both of these sections. Section 5.4 loosely depends on Section 5.3's definition of lifelong reinforcement learning and world models.

5.1 Incremental Improvement With Loss-Estimated Experts

A desirable property of a lifelong learning algorithm is the ability to update itself on new knowledge in a way which leads to incremental improvement. This abstract problem is a central issue in many domains. For example:

1. Open world object detection (cf. Joseph et al. (2021)): Images of new objects and their classification labels are given to the algorithm. How can we update the algorithm on these new images so that it can classify these new objects, without degrading its performance in classifying objects it has already learned to classify?

2. Lifelong reinforcement learning (Thrun and Mitchell, 1995): A reinforcement learning algorithm interacts with new environments. How can we update the algorithm on these new interactions so that it can perform well in these new environments, without degrading its performance on prior environments it has learned to perform well in?

In this section, we model this problem in the general framework of online learning. Over time, we receive new additive knowledge, where we model each piece of

knowledge as an expert e. For example, e could be an image classifier for a subset of objects, or a reinforcement learning policy for a subset of tasks. In each round $r = 1, 2, \ldots$ we are required to perform a specific task by picking an expert e to complete it (e.g. classifying an image, or acting in a reinforcement learning environment). We would like to choose these experts in such a way that our ability to perform new tasks incrementally improves, without degrading our ability to perform old tasks. For example, if at some point in time we've been given an expert e which performs well at navigating mazes in RL environments, then ideally we would like to perform well in all future rounds which correspond to similar maze navigation tasks. This objective is complicated by two factors:

1. The knowledge encoded by an expert e might be appropriate for some rounds, but inappropriate for other rounds. For example, e might be an RL policy good at acting in maze navigation tasks, but bad at object avoidance tasks.

2. Because we have bounded knowledge, we don't necessarily know ahead of time whether e will perform well in the future. For example, e might be an image classifier which does not generalize well on images we happen to encounter. In the lifelong learning setting, we want to avoid making any distributional assumptions about what may happen in the future, because we expect to be surprised as we learn new things (i.e. the environment appears non-stationary). We would therefore like to ensure that even receiving "bad" experts as new knowledge does not significantly degrade our performance.

The key idea in achieving this objective is to strengthen the requirements of what it means for an expert e to be "good". Not only should e have low loss in some rounds, but e should also have epistemic knowledge about its own loss. Towards this end, we force all experts to output an estimate of their loss in the current round before we commit to choosing them. This estimate is only required to be an average-case upper

bound on the loss, and is compatible with outputting the expected value of the loss. For example:

1. If e is an image classifier and x is an image, e outputs an upper bound on its expected classification loss if it were to classify x. If e is highly uncertain about its loss (e.g. x comes from a completely different domain), e can choose to safely output the maximum possible loss.

2. Suppose each round is a reinforcement learning task where the task reward is bounded in $[0, 1]$ and the round loss is $1-$(task reward). If e is a reinforcement learning policy for navigating in a maze, then e can output the loss estimate $1-$(the expected task reward) if it determines the environment is likely a maze environment. Otherwise it can conservatively output 1.

Conceptually, if an expert e incurs low loss in a round, this by itself is not enough for e to receive credit. Instead, e also needs to know that on average (across rounds similar to the current round) it will incur low loss. By requiring experts to not only perform well, but also *know when they will perform well*, we can ensure that on average, our algorithm's performance in each round is close to the lowest loss estimate of "good" experts in each round. The main contributions of this section are as follows:

1. We introduce the **online learning with loss estimates** (OLwLE) problem, in which one maintains a growing set of experts to choose from, and each expert outputs an estimate of its loss in the current round (Definition 5.1).

2. We introduce and motivate the idea of a γ-"good" expert, where the deviations of the loss estimates from the observed losses are controlled by the function γ (Definition 5.2).

3. We propose **FTGE**, a simple and efficient algorithm for this problem, and give an analytical bound on its performance (Definition 6 and Proposition 5.1). From

a practitioner's point of view, Proposition 5.1 gives a concrete way to trade off the rate of learning new knowledge (the rate new experts are given) against the average error, where γ is related to how well you are able to construct loss estimates for good experts in practice.

4. We discuss lower bounds for this problem and alternative approaches to solving it (Propositions 5.2 and 5.3).

Model setting: The model we present for this problem occurs in the online learning setting. The advantage of this setting is that it requires minimal assumptions about the problems one encounters or the experts one learns, which is ideal for lifelong learning problems. On the other hand, the model does not address important questions such as (1) how to learn good experts or (2) how to evaluate whether a proposed expert is reasonable. We leave question (1) for Section 5.2. We do not directly address question (2) in this thesis, but believe it is an interesting area for future work. To make question (2) concrete, imagine receiving two new experts e^1, e^2 for a navigation problem. Imagine that e^1 is an algorithm which always tries to navigate into a volcano, and that e^2 always navigates towards the objective. As a consequence of the minimal assumptions made about the environment in the online learning setting, the model in this section cannot apriori distinguish between e^1 and e^2, because it does not make any assumptions about the losses e^1 and e^2 will actually incur in the future. For example, it could be the case that e^1 actually performs the best in future rounds: maybe it turns out that all volcanoes are dormant, and that they only way to navigate safety to the objective is to travel through a volcano. A more sophisticated model might try to distinguish between e^1 and e^2 by reasoning about the inner workings of each expert instead of treating each expert as a black box. There is likely a challenging balancing act between (1) reasoning about the internal workings of e^1 to avoid catastrophic actions such as navigating into volcanoes and (2)

avoiding rigid models which make strong assumptions about the future (e.g. navigating into volcanoes is always bad) which make them unable to adapt to a constantly changing lifelong learning environment. Because the online learning model does not apriori distinguish between e^1 and e^2, one should think about the learning problems in this section as occurring in environments where losses are bounded, and where we are interested in minimizing an average loss across rounds. In such settings, it is acceptable to perform badly (e.g. navigate towards a volcano) in a small number of rounds provided this does not significantly affect the long run average performance. A natural example of such a setting might be a simulated RL game environment.

Related work: In the online learning literature, to the best of our knowledge, no prior work directly studies the case where both (a) experts estimate their own loss and (b) the goal is to switch between experts based on these (possibly incorrect) estimates.

1. Sleeping Bandits (cf. Kanade et al. (2009)) studies an online learning problem where the set of experts one can choose changes in each round. This framework can model cases where one incrementally receives new experts over time. However, the regret guarantees in these models are analogous to the traditional "single best expert in hindsight" benchmarks in online learning. They are not directly relevant for OLwLE, where we are interested in switching between different experts in each round depending on which expert is most appropriate for the task at hand.

2. Contextual Bandits (cf. Lattimore and Szepesvári (2020)) studies the online learning setting where one competes against the best policy π which chooses experts in each round based on round-specific contextual information. A naive approach to solving OLwLE is to treat the loss estimates of each expert as a context, and reduce this to a contextual bandits problem. A key issue with

this approach is that, in general, the time complexity of implementing a regret minimizer for a contextual bandits problem scales exponentially in the dimensionality of the context. Besides the time complexity issue, it is unclear how to directly apply contextual bandits when the context provided can be adversarial. For example, in OLwLE, we would like to allow some experts to provide "bad" loss estimates while still giving useful guarantees.

3. Another line of work considers how to incorporate machine learning predictions into online learning algorithms (cf. Mitzenmacher and Vassilvitskii (2022)). The goal in this setting is to give online algorithms which give improved regret bounds when the predictions are accurate, while giving optimal worst case regret bounds when the predictions are incorrect. Such algorithms are therefore "robust" to bad estimates. These algorithms have been applied to problems in job scheduling (Purohit et al., 2018), caching (Lykouris and Vassilvitskii, 2021), scheduling problems (Lattanzi et al., 2020), and problems where the loss function in each round is equal to a predictable process plus noise (Rakhlin and Sridharan, 2013). Prior work in this area covers a variety of different online learning settings. However, they generally consider regret measures with respect to the single best fixed expert in hindsight. In contrast, in OLwLE, we are interested in switching between different experts in each round depending on which expert is most appropriate for the task at hand.

4. Learning an ensemble of models for different tasks and selecting between them based on some criteria is a natural approach explored empirically in the literature. For example, it has been studied in the context of image classification (Ahmed et al., 2016), multi-speaker vowel recognition (Jacobs et al., 1991), and lifelong learning (Aljundi et al., 2017). In the context of lifelong learning, an additional appeal of learning multiple models for different tasks is the hope

that this can overcome catastrophic forgetting (Goodfellow et al., 2015), when a single neural network "forgets" past data when it is trained with stochastic gradient descent on new data. For example, Aljundi et al. (2017) propose Expert Gate, which learns an ensemble of neural networks for different tasks, and trains an autoencoder on each task domain. It then uses the reconstruction loss of each autoencoder to select an expert for the current task whose autoencoder reconstruction loss is low relative to the reconstruction loss of the other autoencoders.

Notation: In this section, we intentionally choose to use notation which is distinct from Chapter 4. This is for two reasons:

1. While both Chapter 4 and this section consider problems in the broad domain of online learning, they both deal with fairly different problems. For example, for a subset of options S, the definition of "cost" $c_t(S)$ in Chapter 4 is different to the "loss" $l_r(S)$ in this section. In order to avoid any confusion, we intentionally use different names for these entities.

2. The results of this section will be integrated into a reinforcement learner in Section 5.3. In this section we denote the current round by r and refer to the algorithm choices as *experts* $e \in E$. These notational choices will allow us to make the integration between this section and section 5.3 seamless, while still allowing us to keep the notational norms in reinforcement learning, namely t for the current time step in an environment and $a \in \mathcal{A}$ for the action of an agent.

We begin by formally defining the online learning with loss estimates problem.

Definition 5.1. *The* ***online learning with loss estimates (OLwLE)*** *problem consists of rounds* $r = 1, 2, 3, \ldots$. *Let* E_r *be the set of experts in round* r *where initially* $E_0 = \emptyset$ *and* $|E_1| \geq 1$. *In round* r, *the following occurs:*

1. *$k \in \mathbb{N}_0$ experts $e^1, \ldots, e^k$ are added to the expert set, i.e. $E_r = E_{r-1} \cup \{e^1, \ldots, e^k\}$.*

2. *Each expert $e \in E_r$ outputs a loss estimate $b_r(e) \in [0,1]$.*

3. *The algorithm chooses an expert $e_r \in E_r$ and incurs loss $l_r(e_r) \in [0,1]$.*

4. *In the **bandit** setting of this problem, the algorithm only observes the loss $l_r(e_r)$. In the **full feedback setting**, the algorithm observes $l_r(e)$ for every $e \in E_r$.*

The goal of the algorithm is to choose experts in such a way that we can bound the total loss $\sum_{r=1}^{R} l_r(e_r)$ of the algorithm relative to a benchmark.

Naively, one might think to consider all the loss estimates $\{b_r(e)\}_{e \in E_r}$ in each round, and pick the expert with the lowest loss estimate. The issue with this is that there may be some expert $e \in E_r$ (e.g. an over-fitted classifier) who consistently estimates a low loss $b_r(e) = 0$, even though their actual loss is very high. On the other hand, we cannot simply label experts as "bad" if there is sometimes a discrepancy between the estimate $b_r(e)$ and the observed loss $l_r(e)$. For example, if $l_r(e)$ is random and $b_r(e)$ is equal to the expectation of $l_r(e)$, we still expect there to be a discrepancy between $b_r(e)$ and $l_r(e)$ in any particular round.

To get a sense of how large we should expect this discrepancy to be, fix an expert e and a subset of rounds $\mathcal{I}_e^{\leq R} \subset [R]$. If $b_r(e)$ is the expected value of $l_r(e) \in [0,1]$ for all rounds r, then a standard application of Azuma's inequality (see Theorem C.1 in appendix) tells us that $\mathbb{P}\left[\sum_{r \in \mathcal{I}_e^{\leq R}} (l_r(e) - b_r(e)) \geq k\sqrt{|\mathcal{I}_e^{\leq R}|}\right] \leq e^{-k^2/2}$. This means that we expect the total deviation between the loss estimates and the observed losses to not exceed $k\sqrt{|\mathcal{I}_e^{\leq R}|}$ for a sufficiently large value of k. This observation motivates the following definition: we will call experts "good" if the deviation between the loss estimates and the observed losses *during the rounds they were chosen by the*

algorithm[1] is always bounded by some function γ. If for good experts we think loss estimates are equal to the expected value of the observed loss, then a reasonable choice for γ might be $\gamma(x) = 10\sqrt{x}$. In general, we can choose γ to be larger or smaller depending on empirical performance. For example, if it is very challenging to estimate the loss due to non-stationarity, we can pick γ to be larger or make the loss estimates more conservative. We will only require $\gamma(x) = o(x)$ to be nontrivial in order to get asymptotic guarantees on performance, but the slower γ grows the tighter the guarantees will be.

Definition 5.2. *Let* $\boldsymbol{ALG}$ *be an algorithm for the online learning with loss estimates problem. Let* $\mathcal{I}_e^{\leq R} = \{r \in [R] | \boldsymbol{ALG}$ *chose expert* e *in round* $r\}$*. Let* $\gamma : \mathbb{R} \to \mathbb{R}_+$ *be a monotone-increasing concave function. Then expert* e *is* (γ, R)**-good** *if*

$$\forall R' \in [R], \sum_{r \in \mathcal{I}_e^{\leq R'}} (l_r(e) - b_r(e)) \leq \gamma(|\mathcal{I}_e^{\leq R'}|)$$

We say expert e *is* γ*-good if it is* (γ, R)*-good for all* $R \in \mathbb{N}$*. For all* $r \in \mathbb{N}$*, we denote the set of all* (γ, r)*-good experts at time* r *by* $G_r := \{e \in E_r | e$ *is* (γ, r)*-good*$\}$ *and likewise* $(G_r)^c := E_r \setminus G_r$*.*

We now introduce a very simple algorithm, Follow The Good Experts (**FTGE**), for solving the OLwLE problem. In round r, **FTGE** will simply pick the expert with the smallest loss estimate from the set $E_r \setminus (G_{r-1})^c$, i.e. we exclude experts which are not $(\gamma, r-1)$-good. Because the quality of the loss estimates for experts in G_{r-1} is controlled by γ, we are able to give an overall performance bound for this strategy in terms of γ, the total number of rounds R, and the size of E_R. A key advantage of the simplicity of this approach is that it only requires bandit feedback, i.e. if **FTGE** chooses e_r in round r, it only needs to observe the loss for expert e_r in this round.

[1] There is a subtlety between fixing a subset of rounds in advance, and letting those rounds depend on the choices of the algorithm (which may depend on the loss estimates themselves). See Section C.1 in the appendix for a more detailed discussion.

This is especially useful when getting loss estimates is cheap, but computing actual losses is expensive. For example, imagine that E_r is a set of image classifiers, and x is an image to classify in this round. To get the classification loss of each classifier for x one would need to evaluate each classifier, so the number of model evaluations scales with the size of E_r. On the other hand, to estimate classifier losses, one might learn a cheap image embedding which maps an image x to a local neighbourhood of classifiers for which x is likely to be in their domains. Then one can conservatively estimate $b_r(e) = 1$ for every expert e outside of this neighbourhood, and then run more computationally expensive estimates for experts inside the neighbourhood. It is therefore plausible that computing loss estimates for all experts can be significantly cheaper than needing to compute actual losses.

Algorithm 6: FTGE(γ)

Initialize $(G_0)^c, E_0 = \emptyset$;
for *round* $r = 1, 2, 3, \ldots$ **do**
  Receive experts $e^1, \ldots, e^k$. Set $E_r = E_{r-1} \cup \{e^1, \ldots, e^k\}$ and $\forall i \in [k]$ initialize count_{e^i}=0,error_{e^i}=0,bounds_{e^i}=0;
  Compute $e_r = \arg\min_{e \in E_r \setminus (G_{r-1})^c} b_r(e)$. If $E_r \setminus (G_{r-1})^c = \emptyset$ then let e_r be any $e \in E_r$;
  Pick expert e_r and observe $l_r(e_r)$;
  $\text{count}_{e_r} += 1$;
  $\text{error}_{e_r} += l_r(e_r)$;
  $\text{bounds}_{e_r} += b_r(e_r)$;
  if $error_{e_r}$-$bounds_{e_r} > \gamma(count_{e_r})$ **then**
    $(G_r)^c = (G_{r-1})^c \cup e_r$;
  else
    $(G_r)^c = (G_{r-1})^c$;

We now give a performance bound for **FTGE**. We show that the actual loss incurred by the algorithm $\sum_{r=1}^{R} l_r(e_r)$ over R rounds is bounded by $\sum_{r=1}^{R} \min_{e \in G_r} b_r(e) + |E_R| \left(\gamma\left(\frac{R}{|E_R|}\right) + 1\right)$. Note that $\min_{e \in G_r} b_r(e)$ is equal to the lowest loss estimate over all (γ, r)-good experts in round r. This means that if any expert in $e \in E_r$ is (a) (γ, r)-good and (b) estimates its loss will be small in round r, then $\min_{e \in G_r} b_r(e)$ will

be small. This captures our intuition that in round r, if at any point in the past we had learned a piece of knowledge e which can achieve low loss in round r, then we should be able to perform as well as e. The additional term $|E_R|\left(\gamma\left(\frac{R}{|E_R|}\right)+1\right)$ can be controlled by ensuring our knowledge size $|E_R|$ does not grow too large. For example, if $|E_R| = \mathcal{O}(\sqrt{R})$ and $\gamma(x) = \mathcal{O}(\sqrt{x})$, then $|E_R|\left(\gamma\left(\frac{R}{|E_R|}\right)+1\right) = \mathcal{O}(R^{\frac{3}{4}})$ which means that the average loss $\frac{1}{R}\sum_{r=1}^{R} l_r(e_r)$ converges to the average lowest loss estimate $\frac{1}{R}\sum_{r=1}^{R} \min_{e\in G_r} b_r(e)$ as $R \to \infty$.

Proposition 5.1. *Let e_r be the expert chosen by Algorithm 6 in round r. Then for any $R \in \mathbb{N}$, we have*[2]

$$\sum_{r=1}^{R} l_r(e_r) - \sum_{r=1}^{R} \min_{e\in G_r} b_r(e) \leq |E_R|\left(\gamma\left(\frac{R}{|E_R|}\right)+1\right)$$

In particular, if $|E_R| = \mathcal{O}\left(R^{\alpha}\right)$ and $\gamma(x) = \mathcal{O}\left(x^{\frac{1}{2}}\right)$, then

$$\sum_{r=1}^{R} l_r(e_r) - \sum_{r=1}^{R} \min_{e\in G_r} b_r(e) = \mathcal{O}\left(R^{\frac{1}{2}+\frac{\alpha}{2}}\right)$$

And if $\gamma(x) = o(x)$ and $|E_R| = o(R)$, then

$$\limsup_{R\to\infty}\left[\frac{1}{R}\sum_{r=1}^{R} l_r(e_r) - \frac{1}{R}\sum_{r=1}^{R} \min_{e\in G_r} b_r(e)\right] \leq 0$$

Proof. First note that Algorithm 6 computes $(G_r)^c$ correctly at the end of each round: if $r \in \mathbb{N}$ is the first round in which $e \in E_r$ is not (γ, r)-good, then we add e to the algorithm's calculation of $(G_r)^c$. Likewise, if $e \in E_r$ is added to the algorithm's calculation of $(G_r)^c$, it follows that e is not (γ, r')-good for any $r' \geq r$.

For the sake of clarity of the proof, we will assume that $\forall r \in [R]$, G_r is nonempty. This assumption is not necessary for the result to hold[3], but it does simplify the

[2] We define $\min_{e\in G_r} b_r(e) = 1$ if $G_r = \emptyset$.

[3] If $G_r = \emptyset$ then $\min_{e\in G_r} b_r(e) = 1$ by definition, so the bound only becomes easier when this is the case.

notation. It is also easy to achieve this assumption by adding a "sentinel" expert $e \in E_1$ in round 1 which always estimates $b_r(e) = 1$. Such an expert is trivially γ-good for all non-trivial choices of γ.

We begin by decomposing the total loss in terms of the loss from (γ, R)-good experts and the remaining experts. We have:

$$\sum_{r=1}^{R} l_r(e_r) = \sum_{e \in G_R} \sum_{r \in \mathcal{I}_e^{\leq R}} l_r(e) + \sum_{e \in (G_R)^c} \sum_{r \in \mathcal{I}_e^{\leq R}} l_r(e) \tag{5.1}$$

$$\leq \sum_{e \in G_R} \left(\gamma(|\mathcal{I}_e^{\leq R}|) + \sum_{r \in \mathcal{I}_e^{\leq R}} b_r(e) \right) + \sum_{e \in (G_R)^c} \sum_{r \in \mathcal{I}_e^{\leq R}} l_r(e) \tag{5.2}$$

$$\tag{5.3}$$

which follows by definition of (γ, R)-good. Let $r^*(e) = \max_{r \in \mathbb{N}_0} e$ is (γ, r)-good. Then we have

$$\sum_{e \in (G_R)^c} \sum_{r \in \mathcal{I}_e^{\leq R}} l_r(e) \leq \sum_{e \in (G_R)^c} \left(1 + \sum_{r \in \mathcal{I}_e^{\leq r^*(e)}} l_r(e) \right) \tag{5.4}$$

$$\leq \sum_{e \in (G_R)^c} \left(1 + \gamma(|\mathcal{I}_e^{\leq r^*(e)}|) + \sum_{r \in \mathcal{I}_e^{\leq r^*(e)}} b_r(e) \right) \tag{5.5}$$

$$\leq \sum_{e \in (G_R)^c} \left(1 + \gamma(|\mathcal{I}_e^{\leq R}|) + \sum_{r \in \mathcal{I}_e^{\leq R}} b_r(e) \right) \tag{5.6}$$

where line 5.4 follows because if Algorithm 6 chooses e at time $r > r^*(e)$, then by definition of r^* the algorithm must set $(G_r)^c = (G_{r-1})^c \cup e$, and so e is never again chosen in the execution of the algorithm. Here we are using the simplifying assumption that G_r is never empty. Line 5.5 follows because e is $(\gamma, r^*(e))$-good during the period $[1, r^*(e)]$. The final inequality follows because γ is a monotone-increasing function. Moving back to the original bound, we get

$$\sum_{r=1}^{R} l_r(e_r) \leq \sum_{e\in G_R}\left(\gamma(|\mathcal{I}_e^{\leq R}|) + \sum_{r\in\mathcal{I}_e^{\leq R}} b_r(e)\right) + \sum_{e\in(G_R)^c}\left(1+\gamma(|\mathcal{I}_e^{\leq R}|) + \sum_{r\in\mathcal{I}_e^{\leq R}} b_r(e)\right) \tag{5.7}$$

$$\leq |(G_R)^c| + \sum_{e\in E_R}\gamma(|\mathcal{I}_e^{\leq R}|) + \sum_{e\in E_R}\sum_{r\in\mathcal{I}_e^{\leq R}} b_r(e) \tag{5.8}$$

$$\leq |E_R| + \sum_{e\in E_R}\gamma(|\mathcal{I}_e^{\leq R}|) + \sum_{r=1}^{R} b_r(e_r) \tag{5.9}$$

$$= |E_R| + \sum_{e\in E_R}\gamma(|\mathcal{I}_e^{\leq R}|) + \sum_{r=1}^{R}\min_{e\in E_r\setminus(G_{r-1})^c} b_r(e) \tag{5.10}$$

$$\leq |E_R| + \sum_{e\in E_R}\gamma(|\mathcal{I}_e^{\leq R}|) + \sum_{r=1}^{R}\min_{e\in G_r} b_r(e) \tag{5.11}$$

$$\leq |E_R|\left(1+\gamma\left(\frac{R}{|E_R|}\right)\right) + \sum_{r=1}^{R}\min_{i\in G_r} b_r(e) \tag{5.12}$$

which is what was to be shown. Line 5.10 follows from the fact that in round r Algorithm 6 picks expert e where $e = \arg\min_{e\in E_r\setminus(G_{r-1})^c} b_r(e)$. Line 5.11 follows because $G_r \subset E_r \setminus (G_{r-1})^c$. Line 5.12 follows from an application of Jensen's inequality: by assumption γ is concave, so $-\gamma$ is convex. If we consider the random variable X which uniformally takes a value from the list $\{|\mathcal{I}_e^{\leq R}|\}_{e\in E_R}$, then $\mathbb{E}[-\gamma(X)] \geq -\gamma(\mathbb{E}[X])$, so

$$\mathbb{E}[\gamma(X)] \leq \gamma(\mathbb{E}[X]) \tag{5.13}$$

$$\implies \frac{1}{|E_R|}\sum_{e\in E_R}\gamma(|\mathcal{I}_e^{\leq R}|) \leq \gamma\left(\frac{1}{|E_R|}\sum_{e\in E_R}|\mathcal{I}_e^{\leq R}|\right) = \gamma\left(\frac{R}{|E_R|}\right) \tag{5.14}$$

$$\implies \sum_{e\in E_R}\gamma(|\mathcal{I}_e^{\leq R}|) \leq |E_R|\gamma\left(\frac{R}{|E_R|}\right) \tag{5.15}$$

□

Proposition 5.1 gives an upper bound on $\sum_{r=1}^{R} l_r(e_r) - \sum_{r=1}^{R}\min_{e\in G_r} b_r(e)$ which becomes $\mathcal{O}(R^{\frac{3}{4}})$ if $\gamma(x) = \mathcal{O}(\sqrt{x})$ and $|E_R| = \mathcal{O}(\sqrt{R})$. It is natural to ask if there is a corresponding lower bound. A heuristic argument says that a lower bound of

$\mathcal{O}(\min(|E_R|, R))$ is unavoidable when the losses are chosen by an adversary: every time we are given a new expert e, it is always possible that e incurs high loss on the first round we use it, even though e estimates low loss. More formally, suppose we are given a new expert e in round r. Suppose $\gamma(1) > 0$, so that $c := \max(1-\gamma(1), 0) < 1$. Then suppose $b_r(e) = c$, and that for every expert $e' \in E_r, l_{e',r} = 1$. Then e is (γ, r)-good because if it is chosen in round r, then $l_r(e) - b_r(e) \leq 1 - (1 - \gamma(1)) = \gamma(1)$. Thus $\min_{e' \in G_r} b_{e',r} \leq c < 1$, but $l_r(e_r) = 1$ by construction, so $(l_r(e_r) - \min_{e \in G_r} b_r(e))$ increases by at least $1 - c$ in this round. This argument can be repeated for every round in which a new expert is added.

Likewise, if the losses are random variables, then even if $b_r(e)$ is a perfect loss estimator (equal to the expected value of $l_r(e)$), in the worst case we should still expect $\sum_{r=1}^{R} l_r(e_r) - \sum_{r=1}^{R} \min_{e \in G_r} b_r(e)$ to be as large as $\Omega(R^{\frac{1}{2}})$ with some constant probability. Proposition 5.2 makes this observation formally. To add the maximum degree of flexibility, we imagine that in round r one can pick any set $S_r \subset E_r$ and benchmark performance against $\min_{e \in S_r} b_r(e)$. Picking $S_r = G_r$ results in a lower bound in the setting of Proposition 5.1, but other choices of S_r can cover other problem settings. For example, a classical choice is to consider regret with respect to the best fixed subset $S^* \subset E_R$ in hindsight, i.e. bound the regret $\max_{S \subset E_R} \sum_{r=1}^{R} l_r(e_r) - \sum_{r=1}^{R} \min_{e \in (S \cap E_r)} b_r(e)$. In this case, one can simply choose $S_r = (S^* \cap E_r)$ in Proposition 5.2 to cover this problem setting.

Proposition 5.2. *Let **ALG** be an algorithm for the online learning with loss estimates problem. Let e_r be the expert chosen by the algorithm in round r. Let $\mathcal{S}$ be the set of all R-tuples where the rth tuple element is a non-empty subset of E_r. Suppose that $\forall r \in [R]$ and $e \in E_r$, $b_r(e)$ is equal to the expectation of $l_r(e)$. Then there exists a problem instance such that for sufficiently large R,*

$$\mathbb{P}\left[\forall(S_1,\ldots,S_R)\in\mathcal{S},\sum_{r=1}^{R}l_r(e_r)-\sum_{r=1}^{R}\min_{e\in S_r}b_r(e)\geq\frac{1}{10}R^{\frac{1}{2}}\right]\geq\frac{1}{3}$$

Proof. For each round r, and for each expert $e\in E_r$, let $l_r(e)\sim$ Bernoulli $\left(\frac{1}{2}\right)$ i.i.d. and let $b_r(e)=\frac{1}{2}$ be the expected value of $l_r(e)$.

We have

$$\mathbb{P}\left[\sum_{r=1}^{R}l_r(e_r)-\sum_{r=1}^{R}\min_{e\in S_r}b_r(e)\geq\frac{1}{10}R^{\frac{1}{2}}\right]=\mathbb{P}\left[\text{Binomial}\left(R,\frac{1}{2}\right)-\frac{1}{2}R\geq\frac{1}{10}R^{\frac{1}{2}}\right] \tag{5.16}$$

By the Central Limit Theorem, $\frac{\text{Binomial}\left(R,\frac{1}{2}\right)-\frac{1}{2}R}{\frac{1}{2}\sqrt{R}}$ converges in distribution to the standard normal distribution $N(0,1)$. Let $Z\sim N(0,1)$. Then

$$\lim_{R\to\infty}\mathbb{P}\left[\text{Binomial}\left(R,\frac{1}{2}\right)-\frac{1}{2}R\geq\frac{1}{5}\times\frac{1}{2}R^{\frac{1}{2}}\right]=\mathbb{P}\left[Z\geq\frac{1}{5}\right]>\frac{1}{3} \tag{5.17}$$

and the result follows. □

We finish this section by discussing another natural approach to solving the OLwLE problem. If we think of some experts e as being valid (e.g. generalized correctly, having reasonable loss estimates) and others as being invalid (e.g. overfitted to training data), then it is natural to try and compete with the algorithm which always picks the expert with lowest estimated loss from the subset of valid experts. **FTGE** essentially defines valid as being (γ,R)-good, but in general one could have other criteria for what valid means. A natural way to automatically encapsulate all possible definitions of valid is to try and compete with the algorithm which always picks the expert with lowest estimated loss from the best subset $S\subset E_R$ of experts in hindsight. More precisely, this approach would try to bound the regret

$$\text{Regret} := \max_{S \subset E_R} \sum_{r=1}^{R} l_r(e_r) - \sum_{r=1}^{R} l_r(S \cap E_r) \tag{5.18}$$

where $l_r(S \cap E_r) = 1$ if $S \cap E_r = \emptyset$, otherwise $l_r(S \cap E_r) = l_r(e)$ where e is the expert in $S \cap E_r$ with smallest loss estimate $b_r(e)$ in round r (if there are ties we can break them with some fixed rule). In the full-feedback setting, if one knew E_R in advance, it is straightforward to give a randomized algorithm which guarantees $\mathbb{E}[\text{Regret}] \leq \mathcal{O}\left(\sqrt{R|E_R|}\right)$ by reducing this problem to the online learning with experts setting where each expert corresponds to a subset of E_R. However, a naive implementation of this idea runs in time $\Omega(2^{|E_R|})$. Prior work in online-learning to offline-learning reductions (Hazan and Koren, 2016) shows that this can be sped up provided one can efficiently solve the corresponding offline optimization problem, i.e. for a fixed R and expert sequence $E_1 \subset \ldots E_R$, given $\{l_r(e), b_r(e) \in [0,1]\}_{r \in [R], e \in E_r}$, find $S \subset E_R$ which minimizes $\sum_{r=1}^{R} l_r(S \cap E_r)$. Proposition 5.3 shows that this offline problem is NP-hard, and so this approach is not necessarily tractable.

Proposition 5.3. *Fix $N, R \in \mathbb{N}$. $\forall i \in [N], r \in [R]$ let $b_r(i), l_r(i) \in [0,1]$, and for every $r \in [R]$, suppose that $\{b_r(i)\}_{i \in [N]}$ only contains distinct elements. For $S \subset [N]$, define $l_r(S) = 1$ if $S = \emptyset$ and otherwise $l_r(S) = l_r(\arg\min_{i \in S} b_r(i))$. Then it is NP-hard to compute*

$$\min_{S \subset [N]} \sum_{r=1}^{R} l_r(S)$$

Proof. We will reduce from 3SAT to this problem. Let ϕ be a 3SAT instance with $C > 0$ clauses and $V > 0$ variables $x_1, \ldots, x_V$. We will construct an instance of the minimization problem with $2V+1$ experts and $C+4V(C+1)+[V(C+1)+1] = \mathcal{O}(VC)$ rounds which has a minimum value of $V(C+1)$ if and only if ϕ is satisfiable. For each $i \in [V]$, associate the literal x_i with the expert labeled $2i-1$ and the literal

$\neg x_i$ with the expert labeled $2i$. The expert labeled $2V+1$ will act as a "dummy" expert to ensure that S is nonempty. Suppose S is a solution to the minimization problem. If $2i-1 \in S$ then this corresponds to x_i being true in ϕ. If $2i \in S$ this corresponds to $\neg x_i$ being true. Our first gadget will ensure that exactly one of $2i-1$ and $2i$ is in S if S minimizes the expression. We do this as follows: For each variable x_i, add $2(C+1)$ identical "variable constraint rounds" of the following form (group 1): $b_r(2i-1)=0, b_r(2i)=0.1, l_r(2i-1)=l_r(2i)=0$, $b_r(i') \in (0.1,1), l_r(i')=1$ for $i' \in [2V] \setminus \{2i-1, 2i\}$, and $b_r(2V+1)=1, l_r(2V+1)=1$. By construction, $l_r(S)=0$ if either $2i-1$ or $2i$ are contained in S, and $l_r(S)=1$ otherwise. The purpose of duplicating these rounds $2(C+1)$ times is to make the total loss scale by a factor of $2(C+1)$ depending on this condition. Next, we add a further identical $(C+1)$ rounds (group 2) of the form $b_r(2i-1)=0, l_r(2i-1)=1$, $b_r(i') \in (0,1), l_r(i')=0$ for $i' \in [2V] \setminus \{2i-1\}$, and $b_r(2V+1)=1, l_r(2V+1)=0$. Assume that $2V+1 \in S$ (needed to ensure S is nonempty). Then in these rounds, $l_r(S)=1$ if $2i-1 \in S$, and $l_r(S)=0$ otherwise. Finally, we do the same for $2i$, adding an identical $(C+1)$ rounds (group 3) of the form $b_r(2i)=0, l_r(2i)=1$, $b_r(i') \in (0,1), l_r(i')=0$ for $i' \in [2V] \setminus \{2i\}$, and $b_r(2V+1)=1, l_r(2V+1)=0$. In total, we add $4(C+1)$ "variable constraint rounds" for each variable. For each group of "variable constraint rounds" for a variable x_i, we can calculate the total loss across these rounds according to the following criteria (where total loss = (group 1 loss) + (group 2 loss) + (group 3 loss)):

1. $2V+1 \in S \wedge 2i-1, 2i \notin S \implies$ total loss $= 2(C+1)+0+0 = 2(C+1)$.
2. $2i-1 \in S \wedge 2i \notin S \implies$ total loss$= 0+(C+1)+0 = C+1$.
3. $2i-1 \notin S \wedge 2i \in S \implies$ total loss$= 0+0+(C+1) = C+1$.
4. $2i-1 \in S \wedge 2i \in S \implies$ total loss$= 0+(C+1)+(C+1) = 2(C+1)$.

Next, we add C rounds corresponding to each of the clauses. For cause j, if the literal x_i appears in this clause, then set $b_r(2i-1) \in \{0, 0.1, 0.2\}$ (i.e. ensure all b_r values are distinct) and $l_r(2i-1) = 0$. Likewise, if $\neg x_i$ appears then set $b_r(2i) \in \{0, 0.1, 0.2\}$ and $l_r(2i) = 0$. For all other experts i' which have not been assigned after considering all variables which appear in the current clause, set $b_r(i') > 0.2$ and $l_r(i') = 1$. Then for this round, $l_r(S) = 0$ if and only if there is some literal x_i (or $\neg x_i$) which appears in the current clause whose corresponding expert $2i-1$ (resp. $2i$) is contained in S. Otherwise, $l_r(S) = 1$.

Up until now, we have added $4V(C+1) + C$ rounds. We finish by adding a further $V(C+1)+1$ identical rounds of the form $b_r(2V+1) = l_r(2V+1) = 0$ and $b_r(i') > 0, l_r(i') = 1$ for $i' \in [2V]$. The total loss of these rounds is 0 if $2V+1 \in S$, and $V(C+1)+1$ otherwise.

We are now ready to argue that ϕ is satisfiable if and only if the minimum of this problem is equal to $V(C+1)$. First suppose that ϕ is satisfiable. For each variable x_i, if $x_i = 1$ in the satisfying assignment then let $2i-1 \in S$. Otherwise let $2i \in S$. Lastly, let $2V+1 \in S$. Then the total loss for the "variable constraint rounds" is $V(C+1)$ because only exactly one of experts $2i-1$ and $2i$ appear in S. The total loss for the clause rounds is 0 because for any clause j, there is some literal x_i (or $\neg x_i$) which evaluates to true, so $2i-1$ (resp $2i$) is in S, so $l_r(S) = 0$ for this round. Thus the total loss is $V(C+1)$.

For the other direction, suppose that S is a solution to the minimization problem with total loss $V(C+1)$. Clearly $2V+1 \in S$, otherwise the loss would be larger than $V(C+1)$. For $i \in [V]$, it follows that exactly one of experts $2i-1$ and $2i$ is in S. For the sake of contradiction, suppose that $2i-1, 2i \notin S$. Then by adding $2i-1$ to S, the loss across the "variable constraint rounds" decreases by at least $2(C+1) - (C+1) = C+1$, and the total loss for the clause rounds increases by at most C, so the total loss decreases by at least 1, a contradiction. Likewise, suppose

$2i-1, 2i \in S$. Then by removing $2i$, the total loss across the "variable constraint rounds" decreases by at least $2(C+1) - (C+1) = C+1$, and the total loss across the clause rounds increases by at most C, so the total loss decreases by at least 1, a contradiction. Therefore exactly one of $2i-1$ and $2i$ are in S. It follows that the loss across the clause rounds is 0, because the loss across the 'variable constraint rounds" is equal to the total loss $V(C+1)$. Consider the variable assignment that sets $x_i = 1$ if $2i-1 \in S$, and $x_i = 0$ if $2i \in S$. We claim this is a satisfying assignment for ϕ. For any clause j, its corresponding clause round loss is 0, which means that there exists some literal v in the clause whose corresponding expert is in S. By construction, v evaluates to true, so clause j is satisfied. It follows that ϕ is satisfiable. $\square$

5.2 Incremental Problem Solving

In this section we study the following abstract problem: how can algorithms learn new knowledge *as a function of previously learned knowledge*? This problem is of central importance in lifelong learning, because every lifelong learning algorithm needs some procedure for improving its ability to perform tasks. Despite a reasonably large literature of algorithms which attempt to address this problem in some way, this problem remains challenging for two reasons:

1. Many existing approaches to this problem occur in domains we don't understand well from a theoretical perspective (e.g. optimizing neural networks). As a consequence, they are often based on heuristics and lack theoretical guarantees. This can make it challenging to see if certain techniques proposed in the literature actually move us closer to solving a particular problem. To give a specific example, a widely cited paper claims to show that many of the fairly complicated algorithms proposed for solving catastrophic forgetting in neural networks (Subsection 2.2.2) are outperformed by a naive baseline (Prabhu et al., 2020).

2. Many existing approaches are specific to particular learning domains. As a result, there is currently no standardized definition of "knowledge" or what it means to "update knowledge" in the context of lifelong learning (Chen and Liu, 2018). This can make it challenging to understand how incremental learning techniques in different domains compare to each other. One can find a survey of the varied ways researchers have modelled knowledge for lifelong learning problems in different domains in Section 2.2.

In this section, we aim to partially address both of these challenges. We introduce an abstract theoretical framework for incrementally learning knowledge which is

domain agnostic. We then give an incremental learning algorithm within this framework, and give some mathematically precise statements about the learning behaviour of the algorithm. More specifically:

1. We define the Incremental Problem Solving problem within an abstract framework of Turing machines and program advice strings, similar in spirit to Algorithmic Information Theory (Definitions 5.3 and 5.4). We show with examples how this framework can be flexibly applied to model different kinds of knowledge and domains.

2. We motivate and define **IPS** (Algorithm 7), the "simplest" constant-time algorithm for solving problems and updating knowledge. We prove high level guarantees about the learning behaviour of the algorithm (Propositions 5.4 and 5.5), and show that **IPS** is "universal" in the sense that it can simulate any other incremental learning algorithm (Proposition 5.6).

3. We show how to generalize **IPS** to the settings where one has black box oracle optimizers (Algorithm 8 and Proposition 5.7), as well as when problems are not explicitly given to the algorithm but need to be inferred from data (Algorithm 9). These generalizations allow us to use the algorithm in practical applications, while still maintaining theoretical guarantees.

The algorithms we present should not be thought of as a complete solution to incremental learning. Rather, they should be thought of as a generic framework where one can plug in "black box" tools such as neural networks. The framework provides a clean way of thinking about the operation of the algorithm with high level performance guarantees. However, the guarantees are only practically meaningful if the black boxes are useful in specific ways. If the algorithm fails to solve problems, then the learning guarantees give a transparent high level understanding of why this might be the case. This understanding can then be used to decide which practical

changes are needed to improve the problem solving power of the system. Such choices include the solution weighting function, which problems to give to the system to help it incrementally learn, and which kinds of new black boxes to modularly add to the system. Alternatively, if one is not concerned with practical implementations, the content in this section can be thought of as an idealized theoretical model of time-bounded incremental learning whose properties can be studied analytically.

Related work: The field of Algorithmic Information Theory (AIT) studies how to reason about knowledge in the context of the theory of computation. A definitive resource is "An Introduction to Kolmogorov Complexity and Its Applications" (Li and Vitányi, 2008). Incremental learning has also been considered in the context of AIT, but these are usually related to specific applications or are not intended to be practical. To the best of our knowledge, the framework and algorithms we present in this section are novel. We give some examples of prior work in AIT and incremental learning below:

1. Early work by Solomonoff (1989) sketches, at a high level, how one might build an incremental learning system based on algorithmic probability. It proposes finding solutions to algorithmic problems by starting with a list of primitive "concepts", and combing them together to form possible solutions. After doing this for several problems, the best solution for each problem is taken. One then tries to extract new concepts from these solutions, by looking at the relative frequencies of combinations of concepts. For example, if the concept combination AB occurs more frequently than the relative frequency of the concept A times the relative frequency of the concept B, then one might create the new concept $W = AB$. These ideas are later developed more rigorously in Solomonoff (2003). The main themes of the ideas are conceptually similar to DreamCoder (Ellis et al., 2020) and EC^2 (Ellis et al., 2018).

2. Franz et al. (2021) show how to incrementally compress strings in a near optimal way. The key idea is to greedily find "the shortest feature" of the string which leads to non-zero compression. Such features can be shown to have intuitively nice properties like feature independence.

3. Garcia-Piqueras and Hernández-Orallo (2021) introduce the idea of conditional teaching set size, which measures the algorithmic information complexity of successively teaching new concepts via curricula.

4. Murena et al. (2017) propose a method of incremental learning which can be viewed as a form of transfer learning. Here, a new model M_r is penalized by the Kolmogorov complexity $K(M_r|M_{r-1}, \ldots, M_1)$ of the new model conditional on the prior learned models $M_1, \ldots, M_{r-1}$.

The framework and algorithm we give can be viewed as a natural abstraction of the DreamCoder algorithm (Ellis et al., 2020). We therefore begin by reviewing DreamCoder as a means of motivating the framework.

5.2.1 Background: DreamCoder

DreamCoder is an incremental learning algorithm for solving problems whose solutions are algorithmic or symbolic in nature. This includes list processing problems, symbolic regression, recursive programming, and graphics drawing. Problems are given to the algorithm in rounds. In any particular round r, the algorithm receives problems $p_1, \ldots, p_k$ it needs to find solutions for. A problem consists of a collection of input-output pairs generated from some unknown function, and the goal is to output the function as a solution. For example, if DreamCoder is applied to the list processing domain, concrete examples of problems include:

1. List sorting: the algorithm is given many examples where the input is a list and the output is the list in sorted order, e.g. $[4, 3, 7, 1] \to [1, 3, 4, 7]$.

2. List reversal: the algorithm is given many examples where the input is a list and the output is the list in reversed order, e.g. $[4, 3, 7, 1] \to [1, 7, 3, 4]$.

DreamCoder maintains a library of learned functions ("concepts") L which it has learned from solving problems in previous rounds. Initially, L is initialized with a small set of Turing complete function primitives specific to the problem domain. For example, L might include function primitives which manipulate a pencil if the problem domain is graphics drawing. If the problem domain is list processing, one might initialize $L \supseteq \{\mathrm{Y}, \lambda, \mathrm{nil}, \mathrm{if}, \mathrm{cons}\}$. The algorithm also maintains a neural network Q which it uses to propose solutions to problems. In round r, the algorithm goes through three stages:

1. Wake: In this step, the algorithm tries to find a solution to each problem. For each problem $i \in [k]$, the algorithm draws solution proposals $s_i \sim Q$ from the neural network. A solution is a symbolic expression of a new function, where each symbol corresponds to a function in L. For example, $s_i(z) =$ fold cons (cons z nil). The algorithm picks the solution s_i which maximizes $\mathbb{P}(p_i|s_i)\mathbb{P}(s_i|L)$. Here $\mathbb{P}(p_i|s_i)$ is problem specific and measures how good the solution s_i is. $-\log_2 \mathbb{P}(s_i|L)$ is equal to the number of bits needed to encode the expression s_i given the library L. It can be thought of as the "length" of s_i. Solutions are therefore penalized by their length, or how "complex" they are.

2. Abstract (Sleep): In this step, the algorithm tries to find function subroutines which are common across several solutions, and adds these functions to L as new atomic "concepts". For example, imagine problem 1 is to take a list and double each element (e.g. $[1, 2, 3] \to [2, 4, 6]$) and problem 2 is to take a list and subtract 1 from each element (e.g. $[1, 2, 3] \to [0, 1, 2]$). Imagine the solution found for problem 1 is s_1=(Y (λ (r l) (if (nil? l) nil (cons ((car l) (car l))(r (cdr l)))))) and for problem 2 is s_2=(Y (λ (r l)(if (nil? l) nil(cons (- (car l) 1)(r

(cdr l))))))). If we define the new concept map=(λ (f) (Y (λ (r l) (if (nil? l) nil (cons (f (car l))(r (cdr l))))))), it turns out that each of these solutions can be refactored as s_1=map (λ (z) (+ z z))) and s_2=map (λ (z) (- z 1))), so the subroutine map is a common factor of both solutions. DreamCoder searches for and adds such common solution factors to its library of functions L. This has two effects. Firstly, solutions s_1 and s_2 can now be expressed in a more compact (shorter) form by referencing the single symbol map instead of the multiple symbols needed to implement map, so the solutions become "simpler" when expressed in terms of this new concept. Secondly, it becomes easier to find new solutions in step 1 (Wake) to problems which make use of map as a subroutine. For example, imagine the next problem is to apply two functions $f, g \in L$ to a list (e.g. $[1, 2, 3] \to [g(f(1)), g(f(2)), g(f(3))]$). We can now express a solution with the simple expression s =map g map f. But before map was added to L, we would need to "guess" a very long, complicated solution where each map is replaced by its individual symbols. While guessing this solution might have been too difficult before map was added to L, it now becomes tractable. To make this abstraction step more precise, DreamCoder tries to expand L using the Minimum Description Length (MDL) principle (Rissanen, 1978) (cf. Grünwald and Roos (2019)), by choosing an L which maximally compresses the solutions found. For two function expressions a, b, write $a =_r b$ if a and b are equivalent up to refactoring of their expressions as in the previous example. Note this is strictly stronger than saying a and b are functionally identical (determining functional identity is an undecidable problem). We will discuss the consequences of this in more detail shortly. DreamCoder adds new functions to L in a way which attempts to maximize $\mathbb{P}(L) \prod_{i \in [k]} \max_{a_i =_r s_i} \mathbb{P}(p_i|a_i)\mathbb{P}(a_i|L)$. Here $-\log_2 \mathbb{P}(L)$ is the number of bits needed to encode the library L. Informally, this objective function is maximized when one can find ways of rewriting

each of the solutions $s_1, \ldots, s_k$ so that there are long common factors, each of which occur in large subsets of solutions. Adding these common factors to L decreases $\mathbb{P}(L)$, but increases $\mathbb{P}(a_i|L)$ because the solutions can be written more compactly in terms of symbols from the library L. Practically, DreamCoder uses heuristics and an efficient refactoring algorithm to approximately maximize this expression.

3. Dream (Sleep): DreamCoder trains Q to better predict solutions when given problem instances. This is done by training Q on problems already solved, as well as generating hypothetical problems. Hypothetical problems are generated by randomly sampling functions from L, connecting them together to form a new random function (the target solution), and using this program to generate a problem instance.

The idea of factoring out common function subroutines between solutions is intuitive. Indeed, empirical results show that the library of functions learned by DreamCoder often corresponds to subroutines humans might use. For example, if the problem domain is to find programs which draw abstract shapes, then DreamCoder learns a subroutine arc(x) which draws a semi-circle with radians proportional to x. On the other hand, there are some technical criticisms one can make of the algorithm. We discuss these criticisms here, and foreshadow how the algorithm **IPS** proposed in the next subsection overcomes these challenges.

1. DreamCoder separates the step of finding solutions (wake) from the step of finding common concepts (abstract). To see why this is problematic, let op(x)=(x+1) mod 2, and suppose the concept x mod y is in L. We will set up a situation where op(x) is a common subroutine for two problems, but DreamCoder fails to find it. Suppose problem 1 is to learn the function 2×((x+1) mod 2), and problem 2 is to learn the function 3×((x+1) mod 2). Suppose DreamCoder finds

solution string $s_1 = 2\times((\text{x}+1) \bmod 2)$ for problem 1, but $s_2 = 3\times(\mathbf{(x-1)\ mod\ 2})$ for problem 2. Note that solutions $3\times(\mathbf{(x-1)\ mod\ 2})$ and $3\times(\mathbf{(x+1)\ mod\ 2})$ have the exact same probability $\mathbb{P}(p_2|s)\mathbb{P}(s|L)$ for problem 2 because they are functionally equivalent and have the same length. Therefore DreamCoder considers both solutions to be equally good. However, s_1 and s_2 do not share common factor **(x+1) mod 2** (it requires knowledge of modular arithmetic to deduce that **(x+1) mod 2** and **(x-1) mod 2** are functionally equivalent on $\mathbb{N}$). Therefore, during the Abstract phase, when DreamCoder tries to find a common factor between solutions $s_1 = 2\times((\text{x}+1) \bmod 2)$ and $s_2 = 3\times(\mathbf{(x-1)\ mod\ 2})$, it will not find the shared subroutine $\text{op}(\text{x}) = (\text{x}+1) \bmod 2$. The fix to this problem is to merge waking and sleeping into a single step, so that functionally equivalent subroutines shared across many problems are forced to use the same symbolic implementation. The algorithm we propose in the next subsection, **IPS**, makes this change.

2. The neural network Q encodes the meta knowledge of how to efficiently find solutions in terms of functions in L. This is problematic, because how to update neural networks with new information is not well understood. It would be much more ideal if this meta knowledge was encoded in a more concrete form, like the library of functions L. We solve this problem by abstracting the notion of solutions and knowledge, allowing solutions to receive their own concept library as an advice string. This choice will allow **IPS** to encode meta knowledge in a way similar to the "library of concepts" in DreamCoder.

3. The Minimum Description Length (MDL) principle for learning new knowledge as common subroutines shared between solutions is intuitive and elegant. But under which conditions can we expect this approach to work, and when can we expect it to fail? How do we know there are not better incremental learning

algorithms which use different principles? The algorithm we present in the next subsection will be closely related to the MDL principle, but its operation will be precisely defined within a simple mathematical model. This will allow us to give concrete guarantees which answer the above questions.

5.2.2 An Incremental Problem Solver

The primary purpose of this section is to give a clean, mathematically precise abstraction for thinking about incremental learning. We want this abstraction to be domain agnostic, while also capturing the intuitions behind incremental learning algorithms like DreamCoder. Some technical preliminaries are needed to do this precisely. A more thorough discussion of them can be found in Chapter 2 of Hutter (2005). The two core ideas of these preliminaries are as follows. Firstly, given any specific object in any specific problem domain, we can always imagine encoding this object as a binary string. This allows us to convert all problems, regardless of their domain, into a common language. Secondly, we can also think of any binary string $s \in \{0,1\}^*$ as encoding a Turing Machine. This allows us to reason about computation in an abstract way.

Technical preliminaries: string encoding Given a finite alphabet Σ, we let Σ^* denote the set of all finite length strings consisting of characters from Σ (including the empty string). We denote the set of strings of length $\leq C$ as $\Sigma^C = \{x \in \Sigma^* | |x| \leq C\}$. For $x, y \in \Sigma^*$, we let $xy \in \Sigma^*$ denote string concatenation of x and y. We take for granted that we can map strings $x \in \Sigma^*$ injectively into a prefix-free binary code, and we let $\langle x \rangle \in \{0,1\}^*$ denote the prefix-free binary representation of x. Precise details are given in Definition C.1 in the appendix. We will interchangeably associate $x \in (\Sigma)^*$ with its corresponding prefix-free binary string representation when the context is clear, and write $l(x) = |\langle x \rangle|$ for the length of the prefix-free binary representation of x. Let $x, y, \ldots, z$ be a finite sequence of strings with each

element a member of Σ^*, and denote such an ordered sequence by $x; y; \ldots; z$ to avoid overloading notation. We associate $x; y; \ldots; z$ uniquely[4] with the binary string given by $\langle x \rangle \langle y \rangle \ldots \langle z \rangle =: \langle x; y; \ldots; z \rangle$. Similarly, we define $l(x; y; \ldots; z) = |\langle x; y; \ldots; z \rangle|$. We denote the set of all finite length sequences of strings from Σ^* as $(\Sigma^*)^*$.

Technical preliminaries: universal Turing machines Let $\mathcal{TM}$ be the set of two-input tape, single output tape, binary Turing machines. For any $T \in \mathcal{TM}$, we can encode the transition function of T as a string encoding$(T) \in \Sigma^*$. There exists a fixed, three input tape, single output tape, binary universal Turing machine U such that for any $a, b \in \{0,1\}^*$ and $T \in \mathcal{TM}$, if a, b, and $\langle$encoding$(T)\rangle$ are input into the first, second and third input tapes of U respectively, then U simulates the Turing machine T when fed inputs a and b on its first and second input tapes respectively. In particular the output of U is equal to the output of T on (a, b) if T halts on (a, b), and U does not halt if T does not halt on (a, b). We write $U(a, b, \langle \text{encoding}(T) \rangle) = T(a, b)$ for the output of U and T on these respective inputs, where $T(a, b) = \perp$ if T does not halt on (a, b). Without loss of generality, we can assume that U associates every binary string $s \in \{0,1\}^*$ with the encoding of some two input tape Turing machine. *Unless otherwise stated, we will assume all Turing machines to be deterministic.* If a Turing machine is randomized, we model this as a deterministic Turing machine which is given read-only access to an infinite tape with random bits.

Definition 5.3. *A **program** is a string $s \in \{0,1\}^*$. For $y \in \{0,1\}^*$, we define the function $s(\cdot|y) \in (\{0,1\}^* \to \{0,1\}^* \cup \{\perp\})$ as $\forall x \in \{0,1\}^*$, $s(x|y) := U(x, y, s)$. If s is a randomized Turing machine, we define $s(\cdot|y)$ to be a deterministic function which is randomly chosen and fixed the moment it is defined, depending on the random bits the Turing machine encoded by s is initialized with.*

We now define a game where an algorithm receives problems in batches online, and

[4]Uniqueness follows from the fact that the binary code is prefix free.

must output solutions in the form of *functions* $\tilde{s} \in (\{0,1\}^* \to \{0,1\}^* \cup \{\bot\})$. We use the terms "problem" and "solution" in a very abstract sense. The only requirement is that one should be able to specify a weight $\mathbb{P}(p|\tilde{s}) \in \mathbb{R}_+$ which indicates how well solution $\tilde{s}$ solves problem p. For example:

1. Supervised Learning: p is a regression problem with an associated dataset, and $\tilde{s}$ is a regression model. $\mathbb{P}(p|\tilde{s})$ is the likelihood of the data given model $\tilde{s}$. Here we rely on the fact that, using a binary encoding, any regression model $m : A \to B$ can be associated with a function $\tilde{s}$ with binary input/output and vice versa.

2. Model Based Reinforcement Learning (cf. Moerland et al. (2023)): p is a reinforcement learning problem with an associated sequence of environment observations. $\tilde{s}$ encodes both a model of the environment and a reinforcement learning policy. $\mathbb{P}(p|\tilde{s})$ is the likelihood of the environment observations given the data, times a term which depends on the RL policy loss. We give more details of how to do this in Section 5.3 when we discuss lifelong reinforcement learning.

3. Clustering (cf. Ezugwu et al. (2022)): p is a clustering problem with an associated set of points $x_1, \ldots, x_N \in \mathbb{R}^d$, $\tilde{s}$ encodes an assignment of each point x_i to a cluster (e.g. $\tilde{s}(\langle i \rangle)$ denotes the group label of x_i), and $-\log \mathbb{P}(p|\tilde{s})$ is an unsupervised clustering loss function.

Remark 5.1. *The careful reader will notice that we have not said anything about how long it takes to compute solutions $\tilde{s}(x) = s(x|y)$ on any particular input x. For example, a regression model $\tilde{s}$ might be a bad solution if it takes a long time to evaluate. One can model this by redefining $s(x|y) = \langle t(x,y,s) \rangle U(x,y,s)$, where $t(x,y,s)$ is the time taken by the Turing machine encoded by s to halt when run on input (x,y) (or some large maximum cutoff value). One can then modify the weighting function*

$\mathbb{P}(p|\tilde{s})$ *to penalize solutions which output high values of* $t(x, y, s)$*. In the definition given, we chose not to explicitly model these considerations in favour of simplicity. But they can easily be incorporated with minimal changes.*

Definition 5.4. *The* ***incremental problem solving game*** *consists of rounds* $r = 1, 2, \ldots$*. In round* r*, the algorithm is given* $k_r \in \mathbb{N}$ ***problems*** $p_1, \ldots, p_{k_r}$ *which come from some set PROBLEMS. The algorithm must then output* k_r ***solutions*** *in the form of functions* $\tilde{s}_1, \ldots, \tilde{s}_{k_r} \in (\{0,1\}^* \to \{0,1\}^* \cup \{\perp\})$*, where* $\forall i \in \{1, \ldots, k_r\}$*,* $\tilde{s}_i$ *is the proposed solution to problem* p_i*. For each* $i \in [k_r]$*, the goal of the algorithm is to output a good solution* $\tilde{s}_i \in S_i$ *from the set of solutions* S_i *which "solve" problem* p_i*.*

The incremental learning algorithm we propose will maintain a growing **knowledge base** of strings $\mathcal{KB} = kb_1; kb_2; \cdots \in (\Sigma^*)^*$. We will model *using* knowledge as passing the binary encoded form of this knowledge base as an advice string to a Turing machine. More precisely, given a binary string $s \in \{0,1\}^*$ and a knowledge base $\mathcal{KB}$, we can specify a solution $\tilde{s} = s(\cdot|\langle\mathcal{KB}\rangle)$. Similar to DreamCoder, and following the principles of AIT, we will think of the complexity of a solution $\tilde{s}$ being related to the length of the program s. We want to find short programs s which lead to good solutions. We will now show through examples how our knowledge framework is able to capture existing incremental learning systems similar to DreamCoder, while also being capable of modelling more abstract forms of meta knowledge.

Example 5.1

Consistency with knowledge models in prior work. Given a collection of input-output examples $\{(x_i, y_i)\}_{i=1}^N$, DreamCoder learns a function $\tilde{s}$ which is consistent with the data. $\tilde{s}$ is expressed as a composition of functions coming from a list $L = \{f_1, f_2, \ldots f_z\}$ (L has z elements). For example, $\tilde{s}$ could be $f_7(f_1(f_2(x), f_3(x)), x)$. If w is the number of times a function appears in $\tilde{s}$, then following the MDL prin-

ciple as in DreamCoder, the "length" of $\tilde{s}$ given L is approximately $w \log(z)$ bits, because we need approximately $\log(z)$ bits to specify which of the z functions in L a particular symbol corresponds to. We now show how to encode essentially the same solution within the model of this section, while also highlighting subtle differences between the two approaches. We want to find a function $\tilde{s}$ which is consistent with the examples $\{(x_i, y_i)\}_{i=1}^{N}$. Suppose we have a knowledge base $\mathcal{KB} = f_1; f_2; \ldots; f_z$. Importantly, each $f_i \in \Sigma^*$ is a string of code which implements the function f_i; this contrasts with DreamCoder, where each f_i is an atomic "concept" or black-box function which can form part of a functional expression. Recall that in our model, a solution $\tilde{s}$ is represented by $s \in \{0,1\}^*$ which represents code for a Turing machine[5]; in general this allows greater flexibility than requiring the solution to be a composition of functions. If the mathematical solution is $f_7(f_1(f_2(x), f_3(x)), x)$, we can express this solution in our model as follows, written in pseudo-code: $s = \langle \text{EvalCodeString}(kb_7(kb_1(kb_2(x), kb_3(x)), x)) \rangle$, where kb_i is a primitive which looks up the ith string in the knowledge base $\mathcal{KB}$, and EvalCodeString is a function which interprets raw code (analogous to "eval" in Python). The solution essentially constructs the raw code which expresses the mathematical expression, and then evaluates this code. $\tilde{s} = s(\cdot|\langle \mathcal{KB} \rangle)$ is then functionally equivalent to $f_7(f_1(f_2(x), f_3(x)), x)$. Note that the length of the string s in bits which encodes this solution is at most $\mathcal{O}(1)$ longer than in the case of DreamCoder: we only need to specify an additional call to "EvalCodeString"[6], and approximately $\log(z)$ bits for each kb_i. This means that if one assigns a prior probability to solutions according to $2^{-(\text{binary encoding length of solution})}$, then the prior probability assigned to a solution by this knowledge model is at most a constant factor less than the probability assigned to the equivalent solution by DreamCoder. We can therefore model approaches like

[5]Any other reasonable model of computation could be chosen instead, e.g. a Python interpreter or WordRam model.

[6]Either this function is a primitive instruction in the universal Turing machine, or it can be coded in a constant number of bits.

DreamCoder in this knowledge framework.

EXAMPLE 5.2

The knowledge model is domain agnostic. Because knowledge in our framework is *interpreted* (i.e. a program reads it as input) rather than evaluated (e.g. as a function), we can easily express a description of knowledge from any domain by encoding it as a binary string, without having to consider domain specific details like how that knowledge might actually be *implemented* as e.g. a function. For example, in the context of meta learning, MaML (Finn et al., 2017) learns a parameter $\theta_0 \in \mathbb{R}^d$ such that if you have a regression dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ coming from a certain class of problems, a few iterations of stochastic gradient descent on a model m_θ with initial parameters $\theta = \theta_0$ will quickly converge to a good solution. In our framework, we can encode this knowledge by setting $\langle \mathcal{KB} \rangle = \langle \theta_0 \rangle$. The corresponding solution $\tilde{s}$ can be written in pseudo-code as $s(\cdot|\langle \mathcal{KB} \rangle)$ where $s =$ "StochasticGradientDescent$(kb_1, \mathcal{D})(x)$". By viewing $\mathcal{KB}$ as consisting of raw strings interpreted by a program, we are therefore able to express very different kinds of knowledge inside a unified framework: functions in Example 5.1, and raw parameter vectors in this example. Moreover, mixing knowledge from different domains in the same knowledge base doesn't affect the solution strings (besides possibly requiring a few more bits to encode longer pointers to library elements). For example, adding a string encoding θ_0 to the end of $\mathcal{KB}$ in Example 5.1 will not change the solution string of Example 5.1.

EXAMPLE 5.3

Meta-knowledge can be expressed in solutions. As in the previous examples, suppose we are given a regression problem with dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$. Let $\mathcal{KB} = kb_1; \ldots; kb_z$. We now show how to concretely encode "knowledge of how to use knowledge", analogous to the role of the neural network Q in DreamCoder. Suppose that kb_1 is (a string encoding of) a program which is very effective at finding solutions

to regression problems when given a knowledge base $\mathcal{KB}$. For concreteness, one can assume kb_1 is a linear regression algorithm, and kb_i for $i > 1$ are functions which compute non-linear features which are useful for linear regression. Then a "meta" solution $\tilde{s}$ to the regression problem can be written in pseudo-code as $\tilde{s} = s(\cdot|\langle\mathcal{KB}\rangle)$ where $s =$ "EvalCodeString$(kb_1)(\mathcal{D}, \mathcal{KB})(x)$", where $\mathcal{KB}$ is a variable reference to the knowledge base input. The solution s simulates the program kb_1 (linear regression) by passing it both the dataset $\mathcal{D}$ to do regression on as well as the knowledge base $\mathcal{KB}$ (functions for computing non-linear features). The length of s is $\mathcal{O}(1)$ bits. We are therefore able to express meta-knowledge inside $\mathcal{KB}$ of how to solve problems. This kind of solution expression is only possible because we explicitly feed in $\mathcal{KB}$ as an additional string input, allowing a self-referential behaviour. If we did not do this, the length of the string encoding of the above solution would be $\Omega(z)$, because we would need to hard-code $\mathcal{KB}$ into the solution string s instead of being able to pass it as a reference. Frameworks like DreamCoder are therefore unable to express these kinds of meta solutions as short strings.

Suppose now that someone gives you a knowledge base $\mathcal{KB}$ and k problems $p_1, \ldots, p_k$ to find solutions for in round r. Suppose further that you are only allowed a constant amount of time to solve each problem. What is the "simplest" strategy one could try? One approach is to naively enumerate through a constant number of (a) knowledge strings α shared across all solutions and (b) solutions s for each problem, starting from the simplest (shortest) solutions/knowledge and continuing until the time budget is reached. We formalize this idea in Algorithm 7 (**IPS**). Fix constants $C_1, C_2 \in \mathbb{N}$. For each $\alpha \in \{0,1\}^{C_1}$, and for each problem p_i, we will enumerate through each program $s_i \in \{0,1\}^{C_2}$ and weight these choices by their performance $\mathbb{P}(p_i|s_i(\cdot|\langle\mathcal{KB}\rangle\alpha))$ and their description length complexity $w(|\alpha|)\prod_{i=1}^{k} w(|s_i|)$, where we define $w(n) = \frac{2^{-n}}{n^2}$. We will then draw solutions s and new shared knowledge α in proportion to their weights. We pick this choice of w so that we use the same

prior over strings which is commonly used in the context of MDL. As we will see later, weighting strings in this way will give desirable properties from the perspective of meta learning. $-\log_2(w(|s|))$ approximately corresponds to the number of bits needed to encode s in a prefix free code and ensures that $\sum_{s\in\{0,1\}^*} w(|s|)$ is finite, so that we can have a probability distribution over finite length binary strings.

IPS runs in time $\mathcal{O}(k2^{C_1}2^{C_2}) = \mathcal{O}(k)$ (abstracting the weighting function $\mathbb{P}(p|\tilde{s})$ as a constant time oracle). From a Bayes perspective, it can be shown that **IPS** draws solutions according to

$$\mathbb{P}(s_1,\ldots,s_k,\alpha) \propto w(|\alpha|)\mathbb{1}[|\alpha| \leq C_1] \prod_{i\in[k]} \mathbb{P}(p_i|s_i(\cdot|\langle\mathcal{KB}\rangle\alpha))w(|s_i|)\mathbb{1}[|s_i| \leq C_2]$$

This distribution can be viewed as a truncated form of the distribution implied by the MDL principle, where the truncation comes from the idea that our time-bounded problem solver is only allowed to try a constant number of solutions per problem. The behaviour of **IPS** captures the following two intuitions:

1. Suppose we are given a problem p_i and a knowledge base $\mathcal{KB}$. If there is a simple (short) solution for p_i in terms of the knowledge we have, we will find this solution. Otherwise, the solution is too complicated to find given our current compute budget, so we will not find a good solution.

2. In round r, suppose there is a shared structure α which is useful for many solutions in the current round. Analogously to DreamCoder, one can think of α representing common subroutines or information which is used by many different solutions. We will then add α to $\mathcal{KB}$ in the hope it may be useful for solving future problems as well. By adding α to $\mathcal{KB}$, we reduce the length of future solutions which make use of the subroutines/knowledge encoded in α. This expands the horizon of problems which have simple solutions in terms of $\mathcal{KB}$, which results in an improved capacity to solve more complicated problems.

Algorithm 7: IncrementalProblemSolver (**IPS**)

Receive problems $p_1, \ldots, p_k$ and knowledge base $\mathcal{KB}$ as input;
for $\alpha \in \{0,1\}^{C_1}$ **do**
    $w_\alpha = 0$;
    if α *is a valid encoding of some* $x \in (\Sigma^*)^*$ **then**
        for $i = 1, \ldots, k$ **do**
            $w_{\alpha,i} = 0$;
            for $s \in \{0,1\}^{C_2}$ **do**
                $w_{\alpha,i,s} = \mathbb{P}(p_i | s(\cdot, \langle \mathcal{KB} \rangle \alpha)) w(|s|)$;
                $w_{\alpha,i} += w_{\alpha,i,s}$;
        $w_\alpha = w(|\alpha|) \prod_{i=1}^{k} w_{\alpha,i}$;
Draw $\hat{\alpha} \in \{0,1\}^{C_1}$ in proportion to the weights $\{w_\alpha\}_{\alpha \in \{0,1\}^{C_1}}$;
for $i = 1, \ldots, k$ **do**
    Draw $\hat{s}_i \in \{0,1\}^{C_2}$ in proportion to the weights $\{w_{\hat{\alpha},i,s}\}_{s \in \{0,1\}^{C_2}}$;
Return solution proposals $\hat{s}_1(\cdot, \langle \mathcal{KB} \rangle \hat{\alpha}), \ldots, \hat{s}_k(\cdot, \langle \mathcal{KB} \rangle \hat{\alpha})$ and new knowledge base $\langle \mathcal{KB}_{\text{new}} \rangle = \langle \mathcal{KB} \rangle \hat{\alpha}$;

IPS gives us a concrete, constant time algorithm for incremental problem solving which is domain agnostic. We can now study some simple properties of this algorithm.

The first question one might ask is "will **IPS** find a solution to a particular problem"? Fix a problem p_i, and suppose $S \subset (\{0,1\}^* \to \{0,1\}^* \cup \{\bot\})$ is a set of target solutions which are considered to "solve" p_i. In order for **IPS** to find a solution $\tilde{s}_i \in S$, there are two requirements. Firstly, the solutions $\tilde{s} \in S$ need to have short representations in terms of programs $s \in \{0,1\}^{C_2}$ (condition 1). Secondly, the weighting function $\mathbb{P}(p_i|\tilde{s})$ needs to identify solutions in S as being superior to alternative solutions (condition 2). More formally, we imagine a sequence of groups of problems $\{(p_1^{(j)}, \ldots, p_k^{(j)})\}_{j=1}^{\infty}$ of increasing complexity. For example, $p_i^{(j)}$ could be a regression task with j data points, or a reinforcement learning task of a certain class with complexity controlled by j. We will then require that for any $\tilde{s}' \notin S$ which has a short program, there is some $\tilde{s} \in S$ such that

$$\lim_{j \to \infty} \frac{\mathbb{P}(p_i^{(j)}|\tilde{s})}{\mathbb{P}(p_i^{(j)}|\tilde{s}')} \to \infty \qquad (*)$$

i.e. $\mathbb{P}$ asymptotically prefers solution $\tilde{s}$ to solution $\tilde{s}'$. Under these two conditions we can guarantee that the probability **IPS** returns a solution in S goes to 1 as $j \to \infty$. In practice, this result means that if we want **IPS** to learn a particular solution to a problem, we need to give it problems which allow the weighting function $\mathbb{P}$ to identify the solution. In the i.i.d. regression setting $(*)$ is often automatic if $\mathbb{P}(p_i^{(j)}|\tilde{s})$ is the likelihood of the data given model $\tilde{s}$. However, for more complicated domains, determining how to satisfy $(*)$ may require careful consideration. We will give a concrete example of this after the next proposition.

Proposition 5.4. *Fix a knowledge base $\mathcal{KB} \in (\Sigma^*)^*$, and let $S \subset (\{0,1\}^* \to \{0,1\}^* \cup \{\bot\})$ be a set of solutions which satisfy condition (1) below. Suppose that $\{(p_1^{(j)}, \ldots, p_k^{(j)})\}_{j=1}^{\infty}$ is a sequence of groups of problems which satisfy condition (2). Let E_j be the event that the solution returned for problem $p_i^{(j)}$ by Algorithm 7 when given input $(p_1^{(j)}, \ldots, p_k^{(j)}, \mathcal{KB})$ is an element of S. Then we have $\lim_{j\to\infty} \mathbb{P}[E_j] = 1$.*

Conditions:

1. *$\forall \tilde{s} \in S, \exists s \in \{0,1\}^{C_2 - 1.1\log_2(|\langle \mathcal{KB} \rangle|) - c}$ s.t. $\tilde{s} = s(\cdot|\langle \mathcal{KB} \rangle)$, where c is some constant which depends on the Turing machine encoding scheme.*

2. *$\forall s' \in \{0,1\}^{C_2}, \alpha \in \{0,1\}^{C_1}$, $s'(\cdot|\langle \mathcal{KB} \rangle \alpha) \notin S \implies \exists \tilde{s} \in S$ s.t.*

$$\lim_{j\to\infty} \frac{\mathbb{P}(p_i^{(j)}|\tilde{s})}{\mathbb{P}(p_i^{(j)}|s'(\cdot|\langle \mathcal{KB} \rangle \alpha))} = \infty$$

Proof. Suppose the algorithm draws $\hat{\alpha} = \alpha$. We will show $\lim_{j\to\infty} \mathbb{P}[E_j|\hat{\alpha} = \alpha] = 1$. Since $\mathbb{P}[E_j] = \sum_{\alpha \in \{0,1\}^{C_1}} \mathbb{P}[E_j|\hat{\alpha} = \alpha]\mathbb{P}[\hat{\alpha} = \alpha]$ this will immediately imply the result. Let $A = \{s \in \{0,1\}^{C_2} | s(\cdot|\langle \mathcal{KB} \rangle \alpha) \in S\}$ and $B = \{0,1\}^{C_2} - A$. We have

$$\mathbb{P}[E_j|\hat{\alpha}=\alpha] = \frac{\sum_{s\in A} w_{\alpha,i,s}}{\sum_{s\in A} w_{\alpha,i,s} + \sum_{s\in B} w_{\alpha,i,s}} \tag{5.19}$$

$$= \frac{1}{1+\frac{\sum_{s\in B} w_{\alpha,i,s}}{\sum_{s\in A} w_{\alpha,i,s}}} \tag{5.20}$$

so it suffices to show that $\lim_{j\to\infty} \frac{\sum_{s\in A} w_{\alpha,i,s}}{\sum_{s\in B} w_{\alpha,i,s}} = \infty$.

Let $\tilde{s} \in S$. By condition (1), $\exists s \in \{0,1\}^{C_2 - 1.1\log_2(|\langle\mathcal{KB}\rangle|)-c}$ s.t. $\tilde{s} = s(\cdot|\langle\mathcal{KB}\rangle)$. This implies that $\exists s' \in \{0,1\}^{C_2}$ such that $s'(\cdot|\langle\mathcal{KB}\rangle\alpha) = \tilde{s}$, which we will denote by $s' = h(\tilde{s})$ for any $\tilde{s} \in S$. To see why this is true, consider the two-input tape Turing machine T which simulates the Turing machine encoded by s as follows: T copies s, except when s tries to read past $|\langle\mathcal{KB}\rangle|$ bits on its second input tape, in which case T pretends as if all positions to the right of this point have no input. Then by construction $\langle\text{encoding}(T)\rangle(\cdot|\langle\mathcal{KB}\rangle\alpha) = s(\cdot|\langle\mathcal{KB}\rangle)$. Moreover, we can choose T such that $|\langle\text{encoding}(T)\rangle| \leq C_2$. This is because we need at most $|s| \leq C_2 - 1.1\log_2(|\langle\mathcal{KB}\rangle|) - c$ bits to specify the Turing machine being simulated, and at most an additional $1.1\log_2(|\langle\mathcal{KB}\rangle|)+\mathcal{O}(1)$ bits to hard-code the length $|\langle\mathcal{KB}\rangle|$, and $\mathcal{O}(1)$ bits to code the simulation logic.

Therefore, we have

$$\frac{\sum_{s\in A} w_{\alpha,i,s}}{\sum_{s\in B} w_{\alpha,i,s}} = \frac{\sum_{s\in A} \mathbb{P}[p_i^{(j)}|s(\cdot|\langle\mathcal{KB}\rangle\alpha)]w(|s|)}{\sum_{s\in B} \mathbb{P}[p_i^{(j)}|s(\cdot|\langle\mathcal{KB}\rangle\alpha)]w(|s|)} \tag{5.21}$$

$$\geq \frac{\sum_{\tilde{s}\in S} \mathbb{P}[p_i^{(j)}|\tilde{s}]w(|h(\tilde{s})|)}{\sum_{s\in B} \mathbb{P}[p_i^{(j)}|s(\cdot|\langle\mathcal{KB}\rangle\alpha)]w(|s|)} \tag{5.22}$$

Fix any $\varepsilon > 0$. By condition (2), for any $s \in \{0,1\}^{C_2}$ such that $s(\cdot|\langle\mathcal{KB}\rangle\alpha) \notin S$, there exists a $\tilde{s} \in S$ and $j_s \in \mathbb{N}$ such that $\forall j \geq j_s$

$$\frac{\mathbb{P}[p_i^{(j)}|\tilde{s}]}{\mathbb{P}[p_i^{(j)}|s(\cdot|\langle\mathcal{KB}\rangle\alpha)]} \geq \frac{1}{\varepsilon} 2^{C_2}(w(C_2)^{-1}) \tag{5.23}$$

we will write $\tilde{s} = g(s) \in S$ for this $\tilde{s}$, and pick any $j \geq \max_{\substack{s\in\{0,1\}^{C_2} \\ s(\cdot|\langle\mathcal{KB}\rangle\alpha)\notin S}} j_s$. Then we have

$$\begin{aligned}
\frac{\sum_{\tilde{s}\in S}\mathbb{P}[p_i^{(j)}|\tilde{s}]w(|h(\tilde{s})|)}{\sum_{s\in B}\mathbb{P}[p_i^{(j)}|s(\cdot|\langle\mathcal{KB}\rangle\alpha)]w(|s|)} &\geq \frac{\sum_{\tilde{s}\in S}\mathbb{P}[p_i^{(j)}|\tilde{s}]w(|h(\tilde{s})|)}{\sum_{s\in B}\mathbb{P}[p_i^{(j)}|g(s)]\varepsilon 2^{-C_2}w(C_2)w(|s|)} && (5.24)\\
&\geq \frac{\sum_{\tilde{s}\in S}\mathbb{P}[p_i^{(j)}|\tilde{s}]w(C_2)}{\sum_{s\in B}\mathbb{P}[p_i^{(j)}|g(s)]\varepsilon 2^{-C_2}w(C_2)} && (5.25)\\
&\geq \frac{\sum_{\tilde{s}\in S}\mathbb{P}[p_i^{(j)}|\tilde{s}]}{|B|2^{-C_2}\max_{s\in B}\mathbb{P}[p_i^{(j)}|g(s)]\varepsilon} && (5.26)\\
&\geq \frac{\sum_{\tilde{s}\in S}\mathbb{P}[p_i^{(j)}|\tilde{s}]}{\max_{s\in B}\mathbb{P}[p_i^{(j)}|g(s)]\varepsilon} && (5.27)\\
&\geq \frac{1}{\varepsilon} && (5.28)
\end{aligned}$$

where the last line follows since $g(s) \in S$ for any $s \in B$. It follows that that

$$\lim_{j\to\infty} \frac{\sum_{s\in A} w_{\alpha,i,s}}{\sum_{s\in B} w_{\alpha,i,s}} = \infty$$

□

Example 5.4

Condition 2 in Proposition 5.4 highlights the need to use a weighting function which, for a specific sequence of problems, asymptotically prefers target solutions. While this is often automatic in i.i.d. regression settings, it can require careful thought in more complicated or less structured problem domains. We illustrate this with the following example. We will construct a deterministic reinforcement learning environment where the best solution does not minimize a typical model based reinforcement learning loss

function (i.e. condition 2 does not hold). We will do this by making it infeasible to simulate the environment accurately across many steps. Concretely, consider an environment with 101 steps. In the first step, you observe $(0, q, x_0)$, where $q \in \mathbb{N}$ is a query, $t = 0$ is the current step, and $x_0 \in \mathbb{N}$ is an integer. You must then decide whether to exit the game (reward 0), or continue playing all of the remaining 100 steps. If you continue playing, in each successive step, if x_t is the integer observed in the previous step, then you will observe $(t{+}1, q, x_{t+1})$ where $x_{t+1} \in \{2x_t, 2x_t{+}1\}$ according to some hidden deterministic function $x_{t+1} = \Gamma(x_t)$. One can view this as traversing a binary tree, where $2x_t$ and $2x_t + 1$ are resp. the left and right children of x_t. At the end of 101 steps, you receive reward 1 if $x_{100} = q$, and -1 otherwise, i.e. you should choose to continue playing the game if you think the final integer will be $x_{100} = q$, and choose to exit the game otherwise. A model based reinforcement learning approach might try to learn a transition function f which predicts the next observation x_{t+1} conditional on past observations, and a reward function Reward which predicts the reward observed in the current step conditional on past observations. A typical loss function for a model might look like $\mathcal{L} = \text{error}(f) + \text{complexity}(f) + \text{error}(\text{Reward}) + \text{complexity}(\text{Reward})$ where error and complexity measure the supervised error (on some training dataset) and model complexity respectively. One would then learn a policy π on top of this model of the environment. For simplicity, suppose that π is the optimal policy for an environment described by f and Reward. One can then choose a weighting function $\mathbb{P}(p|(f, \text{Reward})) = \exp(-\mathcal{L})$. The function Reward which minimizes $\mathcal{L}$ is mostly likely (for any reasonable choice of complexity measure) the function which returns 0 if the current step $t \neq 100$, and otherwise -1 if $x_{100} \neq q$, and 1 if $x_{100} = q$. This reward function has 0 error, and low complexity. Consider a function f which minimizes $\mathcal{L}$. Suppose the true transition function Γ is complex, so that for any given x, $f(x) = \Gamma(x)$ with probability $1 - \varepsilon$ (i.e. f can only approximate Γ). Now suppose the model observes $(0, q, x_0)$ in the first step. The model-based

method will then roll out a simulation of the environment using f for 100 steps, computing $\hat{x}_{t+1} = f(\hat{x}_t)$, and pass these estimated observations to Reward to get a reward estimate. However, the probability that the model will predict $\hat{x}_{100} = x_{100}$ correctly is at most $(1-\varepsilon)^{100} \approx 0$ even for small ε. This means that even if $q = x_{100}$, the model of the environment will always conclude it is better to exit the game. In this environment, it is therefore infeasible to predict the reward by simulating every step of the environment. On the other hand, imagine that the majority of the distribution of x_0 is on a small set of integers. In this case, a more practical strategy is to memorize the x_{100} which results from the initial condition x_0 (denote this by $x_{100}(x_0)$), and then use this to predict the reward at step $t = 100$ depending on whether $q = x_{100}(x_0)$. This corresponds to learning a function Reward* which returns 0 if the current round $t \neq 100$, and otherwise returns 1 if the initial condition x_0 was memorized and $x_{100}(x_0) = q$, and -1 otherwise. Note that unlike Reward, Reward* is not a function of the simulated predictions of f; it is only a function of the initial integer x_0 and the current round number. This new function Reward* has non-zero error and higher complexity than Reward, but it clearly results in a more effective policy π for this environment in certain situations. This shows that condition 2 is not always obvious. For this particular problem, two ways to try and overcome it are (1) use a non-model based reinforcement learning loss function or (2) add a loss which penalizes the environment model for inducing policies π which are not consistent with the data (e.g. choose to exit the game when the resulting reward is 1).

The next question we might ask of **IPS** is "will **IPS** learn a particular piece of knowledge α"? Informally, we expect this to occur if knowledge α is useful for improving the performance of many solutions. We can use our model to make this condition mathematically precise. Fix a subset of knowledge $\Gamma \subset \{0,1\}^{C_2}$, and suppose that $\{(p_1^{(j)}, \ldots, p_{k_{(j)}}^{(j)})\}_{j=1}^{\infty}$ is a sequence of groups of problems. Note we now allow the number of problems $k_{(j)}$ to vary as well. For example, one can imagine that Γ

is the set of strings α which encode a linear regression algorithm, and each $p_i^{(j)}$ is a linear regression problem for some dataset $\mathcal{D}_i^{(j)}$, where j is the number of points in the dataset, and the number of problems $k_{(j)}$ increases as a function of j. We can show that as $j \to \infty$, **IPS** will return $\hat{\alpha} \in \Gamma$ (i.e. learn linear regression) if the knowledge in Γ is "asymptotically better" on average than alternative knowledge, in a way which we make precise in Proposition 5.5 below. The proof is similar to that of Proposition 5.4 and can safely be skipped without impacting understanding.

Proposition 5.5. *Fix a knowledge base $\mathcal{KB} \in (\Sigma^*)^*$, and let $\Gamma \subset \{0,1\}^{C_1}$ be subset of strings where $\forall \alpha \in \Gamma$, α is a valid binary encoding of some string $x \in (\Sigma^*)^*$. Suppose that $\{(p_1^{(j)}, \ldots, p_{k_{(j)}}^{(j)})\}_{j=1}^{\infty}$ is a sequence of groups of problems which satisfy condition (1). Let E_j be the event that the new knowledge base returned by Algorithm 7 when given input $(p_1^{(j)}, \ldots, p_k^{(j)}, \mathcal{KB})$ is of the form $\langle \mathcal{KB} \rangle \alpha$, where $\alpha \in \Gamma$. Then we have $\lim_{j\to\infty} \mathbb{P}[E_j] = 1$.*

Condition:

1. $\forall \gamma \in \{0,1\}^{C_1}$ *where γ is a valid encoding of some $x \in (\Sigma^*)^*$, $\gamma \notin \Gamma \implies \exists \alpha \in \Gamma$ s.t.*

$$\lim_{j\to\infty} \frac{\sum_{s_1,\ldots,s_{k_{(j)}} \in \{0,1\}^{C_2}} \prod_{i=1}^{k_{(j)}} \mathbb{P}(p_i^{(j)} | s_i(\cdot | \langle \mathcal{KB} \rangle \alpha)) w(|s_i|)}{\sum_{s_1,\ldots,s_{k_{(j)}} \in \{0,1\}^{C_2}} \prod_{i=1}^{k_{(j)}} \mathbb{P}(p_i^{(j)} | s_i(\cdot | \langle \mathcal{KB} \rangle \gamma)) w(|s_i|)} = \infty$$

Proof. Let $B = \{\langle x \rangle | x \in (\Sigma^*)^* \wedge |\langle x \rangle| \leq C_1\} - \Gamma$. Then we have

$$\mathbb{P}[E_j] = \frac{\sum_{\alpha\in\Gamma} w_\alpha}{\sum_{\alpha\in\Gamma} w_\alpha + \sum_{\alpha\in B} w_\alpha} \tag{5.29}$$

$$= \frac{1}{1 + \frac{\sum_{\alpha\in B} w_\alpha}{\sum_{\alpha\in\Gamma} w_\alpha}} \tag{5.30}$$

so it suffices to show

$$\lim_{j\to\infty} \frac{\sum_{\alpha\in\Gamma} w_\alpha}{\sum_{\alpha\in B} w_\alpha} = \infty \tag{5.31}$$

We have

$$\frac{\sum_{\alpha\in\Gamma} w_\alpha}{\sum_{\alpha\in B} w_\alpha} = \frac{\sum_{\alpha\in\Gamma} w(|\alpha|)\prod_{i=1}^{k_{(j)}}\sum_{s_i\in\{0,1\}^{C_2}}\mathbb{P}(p_i^{(j)}|s_i(\cdot|\langle\mathcal{KB}\rangle\alpha))w(|s_i|)}{\sum_{\alpha\in B} w(|\alpha|)\prod_{i=1}^{k_{(j)}}\sum_{s_i\in\{0,1\}^{C_2}}\mathbb{P}(p_i^{(j)}|s_i(\cdot|\langle\mathcal{KB}\rangle\alpha))w(|s_i|)} \tag{5.32}$$

$$= \frac{\sum_{\alpha\in\Gamma} w(|\alpha|)\sum_{s_1,\ldots,s_{k_{(j)}}\in\{0,1\}^{C_2}}\prod_{i=1}^{k_{(j)}}\mathbb{P}(p_i^{(j)}|s_i(\cdot|\langle\mathcal{KB}\rangle\alpha))w(|s_i|)}{\sum_{\alpha\in B} w(|\alpha|)\sum_{s_1,\ldots,s_{k_{(j)}}\in\{0,1\}^{C_2}}\prod_{i=1}^{k_{(j)}}\mathbb{P}(p_i^{(j)}|s_i(\cdot|\langle\mathcal{KB}\rangle\alpha))w(|s_i|)} \tag{5.33}$$

Fix $\varepsilon > 0$. By condition (1), $\forall\gamma \in B$, $\exists\alpha \in \Gamma$ and $j_\gamma \in \mathbb{N}$ such that $\forall j \geq j_\gamma$

$$\frac{\sum_{s_1,\ldots,s_{k_{(j)}}\in\{0,1\}^{C_2}}\prod_{i=1}^{k_{(j)}}\mathbb{P}(p_i^{(j)}|s_i(\cdot|\langle\mathcal{KB}\rangle\alpha))w(|s_i|)}{\sum_{s_1,\ldots,s_{k_{(j)}}\in\{0,1\}^{C_2}}\prod_{i=1}^{k_{(j)}}\mathbb{P}(p_i^{(j)}|s_i(\cdot|\langle\mathcal{KB}\rangle\gamma))w(|s_i|)} \geq \frac{1}{\varepsilon}2^{C_1}(w(C_1)^{-1}) \tag{5.34}$$

Denote α by $\alpha = g(\gamma)$ for any such $\gamma \in B$, and pick any $j \geq \max_{\gamma\in B} j_\gamma$. Then we have

$$\frac{\sum_{\alpha\in\Gamma} w(|\alpha|)\sum_{s_1,\ldots,s_{k_{(j)}}\in\{0,1\}^{C_2}}\prod_{i=1}^{k_{(j)}}\mathbb{P}(p_i^{(j)}|s_i(\cdot|\langle\mathcal{KB}\rangle\alpha))w(|s_i|)}{\sum_{\alpha\in B} w(|\alpha|)\sum_{s_1,\ldots,s_{k_{(j)}}\in\{0,1\}^{C_2}}\prod_{i=1}^{k_{(j)}}\mathbb{P}(p_i^{(j)}|s_i(\cdot|\langle\mathcal{KB}\rangle\alpha))w(|s_i|)} \tag{5.35}$$

$$\geq \frac{\sum_{\alpha\in\Gamma} w(|\alpha|)\sum_{s_1,\ldots,s_{k_{(j)}}\in\{0,1\}^{C_2}}\prod_{i=1}^{k_{(j)}}\mathbb{P}(p_i^{(j)}|s_i(\cdot|\langle\mathcal{KB}\rangle\alpha))w(|s_i|)}{\sum_{\alpha\in B} w(|\alpha|)\varepsilon 2^{-C_1}w(C_1)\sum_{s_1,\ldots,s_{k_{(j)}}\in\{0,1\}^{C_2}}\prod_{i=1}^{k_{(j)}}\mathbb{P}(p_i^{(j)}|s_i(\cdot|\langle\mathcal{KB}\rangle g(\alpha)))w(|s_i|)} \tag{5.36}$$

$$\geq \frac{\sum_{\alpha\in\Gamma}\sum_{s_1,\ldots,s_{k_{(j)}}\in\{0,1\}^{C_2}}\prod_{i=1}^{k_{(j)}}\mathbb{P}(p_i^{(j)}|s_i(\cdot|\langle\mathcal{KB}\rangle\alpha))w(|s_i|)}{\sum_{\alpha\in B}\varepsilon 2^{-C_1}\sum_{s_1,\ldots,s_{k_{(j)}}\in\{0,1\}^{C_2}}\prod_{i=1}^{k_{(j)}}\mathbb{P}(p_i^{(j)}|s_i(\cdot|\langle\mathcal{KB}\rangle g(\alpha)))w(|s_i|)} \tag{5.37}$$

$$\geq \frac{\sum_{\alpha\in\Gamma}\sum_{s_1,\ldots,s_{k_{(j)}}\in\{0,1\}^{C_2}}\prod_{i=1}^{k_{(j)}}\mathbb{P}(p_i^{(j)}|s_i(\cdot|\langle\mathcal{KB}\rangle\alpha))w(|s_i|)}{\varepsilon|B|2^{-C_1}\max_{\alpha\in B}\sum_{s_1,\ldots,s_{k_{(j)}}\in\{0,1\}^{C_2}}\prod_{i=1}^{k_{(j)}}\mathbb{P}(p_i^{(j)}|s_i(\cdot|\langle\mathcal{KB}\rangle g(\alpha)))w(|s_i|)} \tag{5.38}$$

$$\geq \frac{1}{\varepsilon} \tag{5.39}$$

where the last line follows since $g(\alpha) \in \Gamma$ for any $\alpha \in B$. Therefore

$$\lim_{j \to \infty} \frac{\sum_{\alpha \in \Gamma} w_\alpha}{\sum_{\alpha \in B} w_\alpha} = \infty \tag{5.40}$$

□

We consider one final question about **IPS** before we consider how to generalize it to more practical settings. "Is there a better incremental learning algorithm?" Perhaps there is some incremental learning program $\alpha \in \{0,1\}^*$ which uses an entirely different technique to **IPS** and performs much better at incremental problem solving in certain cases. For example, maybe α uses a specific heuristic to find *long* programs (e.g. large neural networks) $s \in \{0,1\}^*$ which make make good solutions for the problems it is given. We will now show that **IPS** is universal in the sense that if α is in the knowledge base $\mathcal{KB}$, then **IPS** can simulate α to get solutions which are at least as good as the solutions α proposes. The key observation is that regardless of the kinds of solutions α proposes, the following "meta" solution has a short program: s_i^* ="when given input x, run code α from $\mathcal{KB}$ and get the solution f proposed by α for the ith problem. Then output $f(x)$".[7] If α proposes very good solutions (the weighting function assigns them high weight), then so will the meta program, so **IPS** will return these solutions with high probability. What if α is not in $\mathcal{KB}$? If α comes from a class of "good incremental problem solvers" $\Gamma \subset \{0,1\}^{C_2}$ which are effective at finding good solutions to many problems, then including any $\alpha \in \Gamma$ into $\mathcal{KB}$ will allow us to write very short meta solutions s_i^* to each problem, and these solutions will have relatively high weight $\mathbb{P}(p_i|s_i^*)$. In this case, one can show that the single term $\prod_{i=1}^k \mathbb{P}(p_i|s_i^*(\cdot|\langle \mathcal{KB} \rangle \alpha))w(|s_i^*|)$ will dominate the sum in the denominator of condition (2) in Proposition 5.5, meaning that we would expect condition (2) in Proposition 5.5

[7] Re-evaluating α to compute $f(x)$ is quite wasteful computationally. One can consider more sophisticated models where we allow programs to cache their previous computation, but this is beyond the scope of the model presented here. In practice we would just cache f once it is computed instead of re-evaluating α to get f for every new input x.

to hold. This means that **IPS** will add a string $\alpha \in \Gamma$ to its knowledge base $\mathcal{KB}$. Note that this argument only follows because of the description length weight w we chose over programs, which is one justification for a minimum description length principle approach. For example, if we had chosen a uniform weight $w(|s_i|) = 1$ instead, there would not be a bias towards short meta programs, and it would not follow that strings $\alpha \in \Gamma$ are preferred.

The consequence of these observations is that there are no "short counterexamples" of incremental learning algorithms α which are significantly better than **IPS**, in the precise sense of Propositions 5.5 and 5.6. If there were, then **IPS** would learn them, add them to its knowledge base, and simulate them to learn new problems. In order for these ideas to work formally, we need to make slight modifications to our model of computation which we previously omitted for simplicity.

Definition 5.5. *Let ALG be a (randomized) algorithm for the incremental problem solving game. If* $(p_1^{(j)}, \ldots, p_{k_{(j)}}^{(j)})$ *are the problems given in round* j*, then let* $ALG(\{(p_1^{(j)}, \ldots, p_{k_{(j)}}^{(j)})\}_{j=1}^{r})$ *be the distribution of solutions output by ALG in round* r *after receiving the problems for all rounds up to* r *as input. The expanded variant of the computation model is one where in round* r *(1) all Turing machines are given read-only access to problems from prior rounds* $\{(p_1^{(j)}, \ldots, p_{k_{(j)}}^{(j)})\}_{j=1}^{r}$ *and (2) programs are now allowed to encode randomized Turing machines (as described in Definition 5.3). Given a string* $s \in \{0,1\}^{C_2}$ *encoding a randomized Turing machine, and random bits* rnd_s *used to initialize the Turing machine, we will write* $s_{rnd_s}(\cdot|y)$ *to denote the corresponding deterministic function acquired by initializing this Turing machine with random bits* rnd_s*.*

We can now state these ideas more precisely. Fix a round r and problem $p_i^{(r)}$. Suppose that S is a subset of solutions for $p_i^{(r)}$ which are "obviously superior" in the sense that the weighting function $\mathbb{P}(p_i^{(r)}|s)$ assigns $s \in S$ significantly higher weight than other solutions. Suppose further that there is some algorithm ALG for the IPS

game which is good at finding these superior solutions, i.e. ALG proposes a solution $s \in S$ to problem $p_i^{(r)}$ with probability γ. Then we will show that if an implementation $alg \in \{0,1\}^*$ of ALG is included in $\mathcal{KB}$, then **IPS** will return a solution $s \in S$ to problem $p_i^{(r)}$ with probability $\frac{\gamma}{1+\varepsilon}$, where $\varepsilon > 0$ is a parameter which controls how superior the solutions in S are.

Proposition 5.6. *In the setting of Definition 5.5, suppose that the knowledge base $\mathcal{KB}$ in round r is of the form $\langle \mathcal{KB} \rangle = \ldots alg \ldots$, where $alg \in \{0,1\}^*$ encodes a Turing machine which simulates ALG, i.e. the Turing machine outputs solutions according to distribution $ALG(\{(p_1^{(j)}, \ldots, p_{k_{(j)}}^{(j)})\}_{j=1}^r)$ when given input $\{(p_1^{(j)}, \ldots, p_{k_{(j)}}^{(j)})\}_{j=1}^r$. Let $S \subset \{0,1\}^* \to \{0,1\}^* \cup \{\perp\}$ be a subset of "obviously superior" solutions such that condition (1) holds, and let γ be the probability that $ALG(\{(p_1^{(j)}, \ldots, p_{k_{(j)}}^{(j)})\}_{j=1}^r)$ returns a solution $\tilde{s} \in S$ for problem $p_i^{(r)}$ in round r. Let E be the event that Algorithm 7 returns a solution $\tilde{s} \in S$ for problem $p_i^{(r)}$ in round r. Suppose that condition (2) holds. Then we have*

$$\mathbb{P}[E] \geq \frac{\gamma}{1+\varepsilon}$$

Conditions:

1. *Let $\varepsilon > 0$. Let $\alpha \in \{0,1\}^{C_1}, s' \in \{0,1\}^{C_2}$, and $rnd_{s'}$ be the randomness used to initialize s'. If $s'_{rnd_{s'}}(\cdot|\langle \mathcal{KB} \rangle \alpha) \notin S$, then $\forall \tilde{s} \in S$*

$$\frac{\mathbb{P}(p_i^{(r)}|\tilde{s})}{\mathbb{P}(p_i^{(r)}|s'_{rnd_{s'}}(\cdot|\langle \mathcal{KB} \rangle \alpha))} \geq \frac{1}{\varepsilon} 2^{C_2} (w(C_2))^{-1})$$

2. *$C_2 = 1.1 \log(|\langle \mathcal{KB} \rangle|) + \mathcal{O}(1)$ is sufficiently large and k_r is bounded by a constant for all rounds r.*

Proof. Consider the following pseudo-code: "Given input x, run the program described by alg to get the solution f output by alg for the ith problem. Then output $f(x)$". This program can be encoded in $\mathcal{O}(1) + 1.1\log(|\langle\mathcal{KB}\rangle|) + 1.1\log(k_r)$ bits by encoding (a) the position of alg in $\mathcal{KB}$ and (b) the index i of the solution which should be returned. By condition (2), we may assume this program can be encoded in C_2 bits, and we denote this program by s^*. Condition on the event $\hat{\alpha} = \alpha$. For $s \in \{0,1\}^{C_2}$, let rnd_s denote the random bits used to initialize the Turing machine described by s, and denote initializing the TM with these random bits by s_{rnd_s}. Let E_1 be the event that $s^*_{\text{rnd}_{s^*}}(\cdot|\langle\mathcal{KB}\rangle\alpha) \in S$, which by assumption happens with probability γ. Let rnd denote the randomness initialization for all $s \in \{0,1\}^{C_2} - s^*$. Conditional on rnd and rnd_{s^*}, let $A = \{s \in \{0,1\}^{C_2} | s_{\text{rnd}_s}(\cdot|\langle\mathcal{KB}\rangle\alpha) \in S\}$ and $B = \{0,1\}^{C_2} - A$. Then following the proof of Proposition 5.4 we have

$$\mathbb{P}[E|\hat{\alpha} = \alpha, \text{rnd}] \geq \gamma\mathbb{P}[E|\hat{\alpha} = \alpha, \text{rnd}, E_1] \tag{5.41}$$

$$\geq \inf_{\substack{\text{rnd}_{s^*} \\ s^*_{\text{rnd}_{s^*}}(\cdot|\langle\mathcal{KB}\rangle\alpha)\in S}} \mathbb{P}[E|\hat{\alpha} = \alpha, \text{rnd}, \text{rnd}_{s^*}] \tag{5.42}$$

$$= \inf_{\substack{\text{rnd}_{s^*} \\ s^*_{\text{rnd}_{s^*}}(\cdot|\langle\mathcal{KB}\rangle\alpha)\in S}} \frac{1}{1+R} \tag{5.43}$$

where

$$R = \frac{\sum_{s\in B}\mathbb{P}(p_i^{(r)}|s_{\text{rnd}_s}(\cdot|\langle\mathcal{KB}\rangle\alpha))w(|s|)}{\sum_{s\in A}\mathbb{P}(p_i^{(r)}|s_{\text{rnd}_s}(\cdot|\langle\mathcal{KB}\rangle\alpha))w(|s|)} \tag{5.44}$$

$$\leq \frac{2^{C_2}\max_{s\in B}\mathbb{P}(p_i^{(r)}|s_{\text{rnd}_s}(\cdot|\langle\mathcal{KB}\rangle\alpha))}{\mathbb{P}(p_i^{(r)}|s^*_{\text{rnd}_{s^*}}(\cdot|\langle\mathcal{KB}\rangle\alpha))w(C_2)} \tag{5.45}$$

$$\leq \varepsilon \tag{5.46}$$

Where the last line follows from condition (1). It follows that $\mathbb{P}[E] \geq \frac{\gamma}{1+\varepsilon}$.

□

5.2.3 Practical Extension: Oracles

We now consider an extension of **IPS**, **IPS-Oracle**, which brings it closer to modelling practical scenarios. We imagine that instead of naively enumerating strings $\alpha \in \{0,1\}^{C_1}, s \in \{0,1\}^{C_2}$ to find shared knowledge and solutions, we have access to an oracle which explicitly proposes which strings to try. We will weight proposals according to their complexity and solution weight as before. This allows us to avoid impractical enumeration, while also make use of the algorithms which practitioners use (e.g. specific neural network architectures and optimization procedures) for finding good programs. Abstractly, we think of the oracle as consisting of a number of different black box methods created by a practitioner, and we let C_1 and C_2 denote the maximum size (in bits) of the new knowledge and solutions proposed by these black boxes[8]. Given new problems $p_1, \ldots, p_r$, we will feed these problems into each of the black boxes, and receive their proposals for program solutions and new knowledge. A key insight is that we can view this oracle-based problem solver as identical to **IPS**, except that we remove distributional support from solutions and knowledge which the oracles do not propose. As long as there are some black boxes which propose "good" solutions $s \in S$ and "good" knowledge $\gamma \in \Gamma$, then removing this support doesn't hurt us (it only increases the probability of these good solutions), and we can get essentially the same guarantees of **IPS**. We formalize these ideas in Algorithm 8. As an example, Proposition 5.7 below shows how to get essentially the same guarantee for **IPS-Oracle** as **IPS** in Proposition 5.4.

[8]In keeping with our intuition that we only have a finite amount of compute to solve each new problem, we imagine that each black box is some finite learning procedure. For example, a black box might try to fit a neural network of a fixed size with an upper bound on the number of stochastic gradient descent steps it uses.

Algorithm 8: IncrementalProblemSolver-Oracle (**IPS-Oracle**)

Receive problems $p_1, \ldots, p_k$ and knowledge base $\mathcal{KB}$ as input;
Initialize $\forall \alpha \in \{0,1\}^{C_1}, \forall i \in [k], w_\alpha = 0,\ w_{\alpha,i} = 0$;
Receive oracle proposals $OP_1 = \{\alpha_{m_1}\}_{m_1=1}^{M_1}$, $OP_2 = \{(s_{1,m_2}, \ldots, s_{k,m_2})\}_{m_2=1}^{M_2}$, $OP_1, OP_2 \sim$Oracle, where $\forall m_1 \in [M_1], \alpha_{m_1} \in \{0,1\}^{C_1}$ and $\forall m_2 \in [M_2], \forall i \in [l], s_{i,m_2} \in \{0,1\}^{C_2}$;
for $m_1 = 1, \ldots, M_1$ **do**
    $\alpha = \alpha_{m_1}$;
    for $m_2 = 1 \ldots, M_2$ **do**
        for $i = 1, \ldots, k$ **do**
            $s = s_{i,m_2}$;
            $w_{\alpha,i,s} = \mathbb{P}(p_i | s(\cdot | \langle \mathcal{KB} \rangle \alpha)) w(|s|)$;
            $w_{\alpha,i} += w_{\alpha,i,s}$;
for $m_1 = 1, \ldots, M_1$ **do**
    $\alpha = \alpha_{m_1}$;
    $w_\alpha = w(|\alpha|) \prod_{i=1}^{k} w_{\alpha,i}$;
Draw $\hat{\alpha} \in \{0,1\}^{C_1}$ in proportion to the weights $\{w_\alpha\}_{\alpha \in \{0,1\}^{C_1}}$;
for $i = 1, \ldots, k$ **do**
    Draw $\hat{s}_i \in \{0,1\}^{C_2}$ in proportion to the weights $\{w_{\hat{\alpha},i,s}\}_{s \in \{0,1\}^{C_2}}$;
Return solution proposals $\hat{s}_1(\cdot, \langle \mathcal{KB} \rangle \hat{\alpha}), \ldots, \hat{s}_k(\cdot, \langle \mathcal{KB} \rangle \hat{\alpha})$ and new knowledge base $\langle \mathcal{KB} \rangle \hat{\alpha}$;

Proposition 5.7. *Fix a knowledge base $\mathcal{KB} \in (\Sigma^*)^*$, and let $S \subset (\{0,1\}^* \to \{0,1\}^* \cup \{\perp\})$ be a set of solutions. Suppose that $\{(p_1^{(j)}, \ldots, p_k^{(j)})\}_{j=1}^{\infty}$ is a sequence of groups of problems which satisfy condition (1) below. Let E_j be the event that the solution returned for problem $p_i^{(j)}$ by Algorithm 8 when given input $(p_1^{(j)}, \ldots, p_k^{(j)}, \mathcal{KB})$ is an element of S. Let E'_j be the event that when **IPS-Oracle** is given input $(p_1^{(j)}, \ldots, p_k^{(j)}, \mathcal{KB})$, there is some oracle proposal $\tilde{s} \in S$ for the ith problem, i.e. there exists some $m_2^* \in [M_2]$ such that $\forall \alpha \in \{0,1\}^{C_1}$, $s_{i,m_2^*}(\langle \mathcal{KB} \rangle \alpha) \in S$. Let $P_S = \liminf_{j \to \infty} \mathbb{P}[E']$. Then we have $\liminf_{j \to \infty} \mathbb{P}[E_j] \geq P_S$.*

Condition:

1. $\forall s' \in \{0,1\}^{C_2}, \alpha \in \{0,1\}^{C_1},\ s'(\cdot | \langle \mathcal{KB} \rangle \alpha) \notin S \implies \forall \tilde{s} \in S$

$$\lim_{j\to\infty} \frac{\mathbb{P}(p_i^{(j)}|\tilde{s})}{\mathbb{P}(p_i^{(j)}|s'(\cdot|\langle\mathcal{KB}\rangle\alpha))} = \infty$$

Proof. Condition on the event E'_j, and condition on $\hat{\alpha} = \alpha$ for any fixed $\alpha \in \{0,1\}^{C_1}$ (without loss of generality, we will assume the event $E'_j \cap \{\hat{\alpha} = \alpha\}$ has probability greater than 0). We will show $\lim_{j\to\infty} \mathbb{P}[E_j|E'_j, \hat{\alpha} = \alpha] = 1$, and the result will follow. Let Proposals $= \{s_{i,m_2}|m_2 \in [M_2]\}$ be the set of solution proposals for the ith problem, $A = \{s \in \text{Proposals}|s(\cdot|\langle\mathcal{KB}\rangle\alpha) \in S\}$ and $B = \text{Proposals} - A$. For simplicity, we will assume that two different oracles do not give identical proposals. We have

$$\mathbb{P}[E_j|E'_j, \hat{\alpha} = \alpha] = \frac{\sum_{s\in A} w_{\alpha,i,s}}{\sum_{s\in A} w_{\alpha,i,s} + \sum_{s\in B} w_{\alpha,i,s}} \tag{5.47}$$

$$= \frac{1}{1 + \frac{\sum_{s\in B} w_{\alpha,i,s}}{\sum_{s\in A} w_{\alpha,i,s}}} \tag{5.48}$$

so it suffices to show that $\lim_{j\to\infty} \frac{\sum_{s\in A} w_{\alpha,i,s}}{\sum_{s\in B} w_{\alpha,i,s}} = \infty$.

Since we are conditioning on the event E'_j, let $\tilde{s} = s_{j,m_2^*}(\cdot|\langle\mathcal{KB}\rangle\alpha)$ as in the assumption of the proposition. We have

$$\frac{\sum_{s\in A} w_{\alpha,i,s}}{\sum_{s\in B} w_{\alpha,i,s}} = \frac{\sum_{s\in A} \mathbb{P}[p_i^{(j)}|s(\cdot|\langle\mathcal{KB}\rangle\alpha)]w(|s|)}{\sum_{s\in B} \mathbb{P}[p_i^{(j)}|s(\cdot|\langle\mathcal{KB}\rangle\alpha)]w(|s|)} \tag{5.49}$$

$$\geq \frac{\mathbb{P}[p_i^{(j)}|\tilde{s}]w(|s_{j,m_2^*}|)}{\sum_{s\in B} \mathbb{P}[p_i^{(j)}|s(\cdot|\langle\mathcal{KB}\rangle\alpha)]w(|s|)} \tag{5.50}$$

Fix any $\varepsilon > 0$. By condition (1), for any $s \in \{0,1\}^{C_2}$ such that $s(\cdot|\langle\mathcal{KB}\rangle\alpha) \notin S$, there exists a $j_s \in \mathbb{N}$ such that $\forall j \geq j_s$

$$\frac{\mathbb{P}[p_i^{(j)}|\tilde{s}]}{\mathbb{P}[p_i^{(j)}|s(\cdot|\langle\mathcal{KB}\rangle\alpha)]} \geq \frac{1}{\varepsilon} 2^{C_2}(w(C_2)^{-1}) \tag{5.51}$$

Pick any $j \geq \max_{\substack{s \in \{0,1\}^{C_2} \\ s(\cdot|\langle \mathcal{KB} \rangle \alpha) \notin S}} j_s$. Then we have

$$\frac{\mathbb{P}[p_i^{(j)}|\tilde{s}]w(|s_{j,m_2^*}|)}{\sum_{s \in B} \mathbb{P}[p_i^{(j)}|s(\cdot|\langle \mathcal{KB} \rangle \alpha)]w(|s|)} \geq \frac{\mathbb{P}[p_i^{(j)}|\tilde{s}]w(|s_{j,m_2^*}|)}{\sum_{s \in B} \mathbb{P}[p_i^{(j)}|\tilde{s}]\varepsilon 2^{-C_2} w(C_2) w(|s|)} \tag{5.52}$$

$$\geq \frac{\mathbb{P}[p_i^{(j)}|\tilde{s}]w(C_2)}{\sum_{s \in B} \mathbb{P}[p_i^{(j)}|\tilde{s}]\varepsilon 2^{-C_2} w(C_2)} \tag{5.53}$$

$$= \frac{\mathbb{P}[p_i^{(j)}|\tilde{s}]}{|B|2^{-C_2}\varepsilon \mathbb{P}[p_i^{(j)}|\tilde{s}]} \tag{5.54}$$

$$\geq \frac{\mathbb{P}[p_i^{(j)}|\tilde{s}]}{\mathbb{P}[p_i^{(j)}|\tilde{s}]\varepsilon} \tag{5.55}$$

$$= \frac{1}{\varepsilon} \tag{5.56}$$

which shows that $\lim_{j \to \infty} \frac{\sum_{s \in A} w_{\alpha,i,s}}{\sum_{s \in B} w_{\alpha,i,s}} = \infty$. □

Note that both **IPS** and **IPS-Oracle** can be fully parallelized: solution weights can be computed in parallel, and black box solution proposers can be run in parallel. We finish this subsection by discussing how the framework of **IPS-Oracle** might be used in practice. The key high level idea is that one can start with a very simple oracle, run **IPS-Oracle**, and see which problems are not solved well. If a problem is not solved well, then a plausible reason for this is that "good" solutions or knowledge do not have distributional support on the oracle proposals. One can then try to model how to solve a specific class of these problems better, and modularly add this model as a new black box to the oracle. If the model captures this class of problems well, then the support on "good" solutions and knowledge will monotonically increase (in the limit, the oracle proposes all programs $s \in \{0,1\}^{C_2}$ and knowledge $\alpha \in \{0,1\}^{C_1}$, and **IPS-Oracle** becomes equivalent to **IPS**). We therefore have an ad-hoc way of modularly improving the power of **IPS-Oracle**, by looking at the problems it cannot solve, adding models for these problems as black boxes, and repeating. Alternatively, one can think of the oracle as a "bag of tools" for solving different kinds of problems conditional on previously learned knowledge $\mathcal{KB}$. We conclude this subsection by

discussing practical considerations of computing the complexity of proposed solutions. This is practically important, because the complexity of solutions like neural networks (and their overfitting behaviour) is not always well represented by the number of parameters in the network.

Choosing binary encoding lengths: Suppose you are given a regression problem with dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$, and you fit this dataset with a neural network n. You need to encode the network in binary so you can compute the complexity term $2^{-|\langle n \rangle|}$ of the network. A naive choice is to truncate the weights $\theta \in \mathbb{R}^d$ of the network to a finite precision and serialize them, but this choice might not represent the true complexity of the network. In practice, you have several other choices, and you can pick whichever gives the minimum length:

1. If θ can be compressed (e.g. it's very sparse), one can serialize the compression $\tilde{\theta}$ of θ instead of θ itself. More formally, you would be serializing the program $s =$"decompress $\tilde{\theta}$ and then use this model for prediction" which has a shorter binary encoding than θ. In practice, you would just serialize θ (which is functionally equivalent to s), but only charge a complexity cost $2^{-|s|}$ for the shorter representation s. In a second case, if the weights θ are based on fine-tuning existing neural network weights $\theta' \in \mathcal{KB}$ as is done in some continual learning methods (see Subsection 2.2.2), the difference $\theta - \theta'$ may be small and easier to compress than θ.

2. Suppose you are in the few shot learning setting, so the number of data points N is small, and you learned θ by initializing the neural network with a parameter θ_0 and then performing a few steps of gradient descent on $\mathcal{D}$. In this case, if θ_0 is in $\mathcal{KB}$, then encoding the dataset $\mathcal{D}$ (which can be used with θ_0 to reconstruct θ) can be shorter than encoding θ. More formally, you would be serializing the program $s =$"Apply 10 steps of SGD on $\theta_0 = kb_i$ with dataset $\mathcal{D}$, and then use

this model for prediction".

3. Suppose you are in the online regression setting, so we need to predict x_i after seeing $\{(x_j, y_j)\}_{j=1}^{i-1}$. Suppose the neural network n has an architecture A. Instead of returning the network n as a solution, one can instead consider the solution which performs a rolling regression of a neural network with architecture A, and uses this rolling regression to predict the next y_j. After enough online datapoints, one would expect this solution to perform as well as n, while possibly requiring much fewer bits to encode. More formally, you would be serializing the program $s =$"Fit a network with architecture A to previously seen data, and use this network to predict y_j given x_j". The length of s is related to the number of bits needed to encode A, rather than the number of bits needed to encode the neural network parameters. Note s does not need to encode the previously seen data, because this is given to it as input by the problem.

4. Suppose the neural network was trained with natural selection instead of stochastic gradient descent. In this case, one can encode the neural network n very efficiently with pseudo-random number generator seeds instead of the network parameters θ, as detailed in Salimans et al. (2017) (see Subsection 2.2.3 for a more detailed account of these kinds of techniques).

5.2.4 Practical Extension: Unstructured Problems

So far we have assumed that in round r, individual problems $p_1, \ldots, p_k$ are neatly given to us. However, we can also imagine settings where we are only given raw data $\mathcal{D}$, and we need to figure out how to decompose this into problems ourselves. As a concrete simple example, imagine that $\mathcal{D}$ consists of many tuples of the form (task name, $x \to y$). For example, in the case of list processing problems in DreamCoder, we could imagine task name $\in$ {reverse list, double elements of list, add 1 to every element of list}, and each $x \to y$ is a single example of that task. The single program s which,

when given input (task name, x) performs the specified task on input x, is very long (i.e. complex) and may be intractable to find. However, we could also separate the data $\mathcal{D}$ into simpler isolated problems $p_1, \ldots, p_k$ where p_i corresponds to the ith task. Then problem p_i is, given input (task name, x), correctly decide whether this input belongs to the ith task type, and if so perform the ith task on x. The program s_i to an isolated task p_i is significantly shorter (i.e. less complex) than the solution which solves all tasks simultaneously, and so may be more tractable to find. On the other hand, if we can solve each individual problem $p_1, \ldots, p_k$, we can combine their solutions together to solve the full problem.

How can we modify **IPS-Oracle** to model this idea? The issue is nuanced, because knowing how to break data up into subproblems is a problem in itself; our system should incrementally learn to better decompose data into subproblems, as well as how to better solve these subproblems. Thankfully, the abstract knowledge framework of **IPS** is flexible enough to make modelling this relatively simple. Just like how programs specify solutions to problems, we will define a *problem encoder* $pe \in \{0,1\}^{C_3}$ to be a program which, when given a knowledge base $\mathcal{KB}$ and data $\mathcal{D}$, outputs a list of problems $p_1, \ldots, p_k = pe(\langle \mathcal{D} \rangle | \langle \mathcal{KB} \rangle)$ to be solved. The task of decomposing problems is therefore treated on equal footing as the task of solving the problems themselves. We will need to ensure the output of pe comes from a set of valid encodings $VE(\mathcal{D})$ so that pe can only output valid decompositions of the data $\mathcal{D}$. For example, in the list processing example, pe needs to decide how to partition the datapoints in $\mathcal{D}$ into individual problems which can be solved separately and later combined. pe is only allowed to output problems $p_1, \ldots, p_k$ which (a) are of the form described previously and (b) partition the entire dataset $\mathcal{D}$. For example, if $pe(\langle \mathcal{D} \rangle | \langle \mathcal{KB} \rangle)$ did not include some of the datapoints in $\mathcal{D}$, this would not be a valid decomposition of $\mathcal{D}$. We formalize this discussion below:

Definition 5.6. *Fix a collection of valid problem encodings* $\{VE(x)\}_{x \in \{0,1\}^*}$*, where*

$V(x) \subset \{0,1\}^*$ *is the set of valid problem encodings for data* x*. A* ***problem encoder*** *is a program* $pe \in \{0,1\}^*$ *such that* $\forall x, y \in \{0,1\}^*$*,* $pe(x|y)$ *encodes a list of problems* $p_1, \ldots, p_{k_x}$*, where* $p(x|y) \in VE(x)$ *and* $k_x \in \mathbb{N}$*.*

We can now extend **IPS-Oracle** to **IPS-General**. Instead of receiving problems $p_1, \ldots, p_k$, **IPS-General** will receive raw data $\mathcal{D}$ it must decompose into problems. Like **IPS-Oracle**, **IPS-General** will receive oracle proposals for problem encoders which it will weight according to (a) their complexity and (b) how well the system can find solutions which solve these problem decompositions. From a Bayes perspective, it can be shown that if $\alpha \in \{0,1\}^{C_1}, pe \in \{0,1\}^{C_3}, s_1, \ldots, s_{k_{pe}} \in \{0,1\}^{C_2}$ are each proposed by the oracle where $p_1, \ldots, p_{k_{pe}} = pe(\langle \mathcal{D} \rangle | \langle \mathcal{KB} \rangle \alpha)$, then **IPS-General** draws samples from a probability distribution given by

$$\mathbb{P}(\alpha, pe, s_1, \ldots, s_{k_{pe}}) \propto w(|\alpha|) w(|pe|) \prod_{i=1}^{k_{pe}} \mathbb{P}(p_i|s_i) w(|s_i|)$$

At a high level, we expect the core behaviour of **IPS-General** to be captured by the propositions given for **IPS** and **IPS-Oracle**, modulo marginalizing over the set of problem encodings $\{pe\}_{pe \in Oracle}$. We therefore find it more instructive to focus on those propositions to understand the behaviour of the algorithm, as opposed to re-deriving similar but more complicated expressions. We conclude the Incremental Problem Solving section by giving the algorithm for **IPS-General**.

Algorithm 9: IncrementalProblemSolver-General (**IPS-General**)

Receive data $\mathcal{D}$ and knowledge base $\mathcal{KB}$ as input;
Pass data $\mathcal{D}$ to oracle and receive proposals $OP_1 = \{\alpha_{m_1}\}_{m_1=1}^{M_1}$, $OP_3 = \{pe_{m_3}\}_{m_3=1}^{M_3}$, $OP_1, OP_3 \sim$**Oracle**, where $\forall m_1 \in [M_1], \alpha_{m_1} \in \{0,1\}^{C_1}$, and $\forall m_3 \in [M_3], pe_{m_3} \in \{0,1\}^{C_3}$;
for $m_1 = 1, \ldots, M_1$ **do**
    $\alpha = \alpha_{m_1}$;
    for $m_3 = 1, \ldots, M_3$ **do**
        $pe = pe_{m_3}$;
        $p_1, \ldots, p_{k_{pe}} = pe(\langle\mathcal{D}\rangle|\langle\mathcal{KB}\rangle\alpha)$;
        Pass problems $p_1, \ldots, p_{k_{pe}}$ to oracle and receive proposals $OP_2 = \{(s_{1,m_2}, \ldots, s_{k_{pe},m_2})\}_{m_2=1}^{M_2}$, $OP_2 \sim$**Oracle**, where $\forall i \in [k_{pe}], \forall m_2 \in [M_2], s_{i,m_2} \in \{0,1\}^{C_2}$;
        for $m_2 = 1 \ldots, M_2$ **do**
            for $i = 1, \ldots, k_{pe}$ **do**
                $s = s_{i,m_2}$;
                $w_{\alpha,pe,i,s} = \mathbb{P}(p_i|s(\cdot, \langle\mathcal{KB}\rangle\alpha))w(|s|)$;
                $w_{\alpha,pe,i} += w_{\alpha,pe,i,s}$;
        $w_{\alpha,pe} = w(|pe|)\prod_{i=1}^{k_{pe}} w_{\alpha,pe,i}$
for $m_1 = 1, \ldots, M_1$ **do**
    $\alpha = \alpha_{m_1}$;
    $w_\alpha = w(|\alpha|)\sum_{m_3=1}^{M_3} w_{\alpha,pe_{m_3}}$;
Draw $\hat{\alpha} \in \{0,1\}^{C_1}$ in proportion to the weights $\{w_\alpha\}_{\alpha\in\{0,1\}^{C_1}}$;
Draw $\widehat{pe} \in \{0,1\}^{C_3}$ in proportion to the weights $\{w_{\hat{\alpha},pe}\}_{pe\in\{0,1\}^{C_3}}$;
Let $p_1, \ldots, p_{k_{pe}} = \widehat{pe}(\langle\mathcal{D}\rangle|\langle\mathcal{KB}\rangle\hat{\alpha})$;
for $i = 1, \ldots, k_{pe}$ **do**
    Draw $\hat{s}_i \in \{0,1\}^{C_2}$ in proportion to the weights $\{w_{\hat{\alpha},\widehat{pe},i,s}\}_{s\in\{0,1\}^{C_2}}$;
Return problem decomposition $(p_1, \ldots, p_{k_{pe}})$, solution proposals $\hat{s}_1(\cdot, \langle\mathcal{KB}\rangle\hat{\alpha}), \ldots, \hat{s}_{k_{pe}}(\cdot, \langle\mathcal{KB}\rangle\hat{\alpha})$ and new knowledge base $\langle\mathcal{KB}\rangle\hat{\alpha}$;

5.3 A Lifelong Reinforcement Learner

In this section we combine the algorithms in Sections 5.1 and 5.2 to make Abstract Lifelong Reinforcement Learner (**ALRL**), an algorithmic framework for lifelong reinforcement learning (LRL). The results of Sections 5.1 and 5.2 will imply specific high-level guarantees on the behaviour of **ALRL**: its performance in hypothetical scenarios, which tasks it will learn to solve, and which knowledge it will accumulate over time. An overview of prior work in LRL can be found in Subsection 2.2.5. Existing approaches to LRL are generally either black box optimization approaches with neural networks (Ring, 1998; Tessler et al., 2017; Mendez et al., 2022), or based on principled idealized statistical models (Wilson et al., 2007; Brunskill and Li, 2014; Lecarpentier et al., 2021; Wang et al., 2022). While the black box methods can work nicely in practice for specific problems, they lack any interpretable guarantees which allow them to be used as general lifelong reinforcement learners. On the other hand, the more rigorous methods tend to suffer from at least one of three weaknesses:

1. The guarantees rely on very idealized assumptions. For example, a common model is to assume the environment consists of separate stationary Markov decision processes (MDPs) one can repeatedly sample from (Lecarpentier et al., 2021).

2. The model is unable to represent meta knowledge in the same way as **IPS** from Section 5.2. For example, Wilson et al. (2007) model the environment as a hierarchical mixture of MDPs. Their model can learn new environments, but its method for learning environments more quickly (via grouping similar environments together) is fixed by the algorithm. One can construct hypothetical environments where there exists a meta learning procedure for learning them more quickly, but the hierarchical MDP mixture model will never learn this meta procedure.

3. In order to be interpretable, the method uses classical methods which are not necessarily more performant than black box methods in practice. For example, the algorithm proposed in Wang et al. (2022) learns a mixture of models using the EM algorithm. But in some applications, end to end training with a single neural network may be more performant. Ideally we would be able to switch between techniques depending on the problem at hand.

ALRL is intended to make progress on merging the strengths of less understood black box techniques and with more interpretable theoretical models. The algorithm inherits high level theoretical guarantees from the algorithms derived in sections 5.1 and 5.2, and is able to model meta knowledge. On the other hand, **ALRL** is built on top of an oracle where practitioners can implement their own black box problem solvers (e.g. neural networks) to search for programs which are effective at solving RL tasks. The framework therefore attempts to make the best of interpretable guarantees from formal methods as well as heuristic black box techniques which can perform well in practice.

We begin by defining the LRL problem. It can be helpful to contrast LRL with RL in order to appreciate the difficulties of defining LRL. In RL, one typically assumes that one receives repeated samples from a markov decision process (MDP), and one needs to learn a policy which performs well in this MDP. In contrast, in the general case, LRL is typically thought of as consisting of a *single* interaction with an environment over an infinite number of time steps. There is no built-in repeated sampling of a well defined object like a MDP which allows us to learn regular structure in a clean way. We need to carefully balance making the LRL problem tractable by adding enough regularity, while also keeping it general enough to capture the spirit of the problem. There is no standardized definition of LRL, so we give our own definition below.

Model motivation: We imagine the LRL problem as consisting of rounds $r = 1, 2, 3, \ldots$. In each round, an agent interacts with the environment for T_r steps, making actions and receiving observations and rewards. The goal of the agent in round r is to maximize the sum of the rewards it receives in round r. In round r, the agent is therefore only concerned with doing well in the rth task. While tasks are "independent" of each other in the sense that their rewards are maximized independently, the tasks themselves may not necessarily be independent. For example, imagine in round r that a user is instructing an agent about which objects the agent must collect in a puzzle environment. The agent scores high reward if it listens attentively, and correctly answers questions which probe the agent's understanding about the user's instructions. In round $r + 1$, the agent must collect the objects, and it scores high reward if it successfully collects them and returns to the user. The agent relies on the information it learned in round r, but in round r the agent was optimizing to "listen well" rather than "listen so objects can be correctly collected in round $r + 1$". One may reasonably wonder if this "independent round" design is really necessary. Note that in the LRL setting, it is impossible to have a naive objective like "maximize the sum of all rewards across all time", because this sum might diverge. Hutter (2005) gives a discussion of the ways this problem is typically overcome, including (1) discounting future rewards so that the sum is finite and (2) at any time, maximizing the sum of rewards over T steps into the future, where T is some large constant. When (1) is done by exponentially decaying rewards, it is effectively equivalent to (2) because rewards become negligible past a certain future horizon. Our choice of using "rounds" can be thought of as a variant of (2) where we effectively choose the length of T dynamically depending on the task and power of the agent. For example, we can initially give the agent short tasks like "listen to object collection instructions" and "collect objects", and later give it tasks with a longer time horizon such as "listen to object collection instructions and collect objects" as its knowledge improves. The

second restriction we will make is that the sum of the rewards in each round will be bounded in $[0, 1]$. This ensures that the algorithm is allowed to "make mistakes" in rounds, without having the mistake from one round dominate the sum of the rewards across rounds.

Model assumptions and limitations: Before we give a formal definition, we give a more focused discussion on two particular aspects of this learning model.

1. Independent finite rounds: in this model, each round has finite length, and the problem of maximizing reward in each round is treated independently. A significant consequence of this design is that the algorithm is guaranteed *fast feedback* – its optimization objective (the reward in the current round) is observed immediately after the current round is finished. One can contrast this to RL problems where one might be optimizing for rewards which occur far into the future. In such cases, the algorithm will only observe much later if the actions taken towards this objective actually resulted in high reward. In the context of lifelong learning, where we only get one trajectory from an infinitely long environment, it appears to be extremely challenging to get meaningful learning guarantees without some kind of fast feedback assumption. At the same time, the way fast feedback is encoded into the model presented here is perhaps relatively simplistic; there are likely more sophisticated (and more realistic) ways to encode this assumption.

2. Bounded round rewards and the online learning setting: in this model, each round has bounded reward, and the learning problem is treated very similarly to the distribution free adversarial online learning setting. In this setting, it is always possible to make mistakes; because of this, we necessarily need to bound the size of these mistakes (i.e. have bounded reward). At the same time, this means that we also allow algorithms to make arbitrary mistakes (i.e. achieve

the worst reward score possible) in some rounds. The power of the online learning setting is that one does not need to make distributional assumptions about the future, which is ideal in the context of lifelong learning. A major drawback of this setting is that it does not distinguish between (1) an agent making reasonable mistakes in environments it could not have foreseen, and (2) an agent randomly making safety critical mistakes or non-sensical decisions. This is because in both cases (1) and (2) the agent receives the same reward (the minimum possible reward). Future work might explore how to build models which are able to better distinguish between cases (1) and (2), and give useful guarantees which limit the occurrences of case (2).

Definition 5.7. *The **incremental lifelong reinforcement learning game** consists of a sequence of rounds $r = 1, 2, 3, \ldots$. Each round $r \in \mathbb{N}$ consists of a sequence of timesteps $t = 1, 2, 3, \ldots, T_r$. The t-th timestep consists of the following sequence of events:*

1. *The agent receives as input an observation $obs_{r,t} \in OBS$, a reward $reward_{r,t} \in \mathbb{R}$, and a flag $end_{r,t} \in \{0, 1\}$ which is equal to 1 iff this is the last timestep of the current round.*

2. *The agent outputs an action $a_{r,t} \in \mathcal{A}$.*

Denote the total reward for the r-th round as $\text{total-reward}_r = \sum_{t=1}^{T_r} reward_{r,t}$. It is guaranteed that $\text{total-reward}_r \in [0, 1]$. The goal of the algorithm in round r is to choose actions which achieve high total reward for the current round.

We will combine the algorithms in Sections 5.1 and 5.2 as follows. Experts in Section 5.1 will correspond to agent policies. At the beginning of round r, each expert $e \in E$ will conservatively predict the expected reward in round r if its policy e^π were used for the current round. We will use **FTGE** to pick which expert to follow.

After a sufficient number of rounds have passed, we will collect data $\mathcal{D}$ consisting of the environment interactions (observations, rewards, actions) of previous rounds. Following Section 5.2, we will decompose this data into *problems*, where each problem corresponds to a subset of the data $\mathcal{D}$. A solution to problem p will be a model of the world which explains the observed data, and gives rise to an expert policy e^π which achieves high reward within this model. We will use **IPS-General** from Section 5.2 to find these solutions, feeding new experts to **FTGE** and maintaining a growing knowledge base $\mathcal{KB}$ of accumulated knowledge for solving problems. In order to do this precisely, we will now define the notation, model based solutions, and problem decompositions described above.

Notation: Let $(r,t)-1$ denote the round and timestep immediately preceding (r,t) if $(r,t) \neq (1,1)$. Let $X_{r,t} = (\text{obs}_{r,t}, \text{end}_{r,t}, a_{r,t})$, and $\tilde{X}_{r,t} = (\text{obs}_{r,t}, \text{reward}_{r,t}, \text{end}_{r,t}, a_{r,t})$. Let $X_{\leq(r,t)} = X_{1,1}, \ldots, X_{1,T_1}, X_{2,1} \ldots X_{2,T_2}, \ldots X_{(r,t)}$ denote the corresponding ordered sequence of data up to (r,t), and $X_r = X_{r,1}, \ldots, X_{r,T_r}$. Define $\tilde{X}_{\leq(r,t)}$ and $\tilde{X}_r$ similarly.

Notation: induced functions and extended domains Given a function $e \in \{0,1\}^* \to (\{0,1\}^* \cup \{\bot\})$, for $a \in \Sigma^*$, we will define $e^a \in \{0,1\}^* \to (\{0,1\}^* \cup \{\bot\})$ to be the function defined by $\forall x \in \{0,1\}^*$, $e^a(x) := e(\langle a \rangle x)$. If A and B are sets, we write $e : A \to B$ to indicate the function e redefined on the domains A and B by associating binary strings (or $\bot$) to elements of A and B in some reasonable way. For example, we can define $e : \{0,1\}^d \to \mathbb{R}$ by evaluating the binary string $x \in \{0,1\}^d$, receiving $y = e(x)$, and then interpreting y as a binary encoding of a real number.

We now define world models within the framework of Section 5.2. In the context of model based reinforcement learning, world models are learned simulators of the environment which can be used to learn agent policies. There are different ways of constructing world models. Definition 5.8 below is an abstraction of a broad class of world models. DreamerV3 (Hafner et al., 2023), Structured World Models (Wu et al.,

2021) and work by Ha and Schmidhuber (2018) are concrete practical examples of this class. The reader may want to reference Section 5.4 which reviews DreamerV3 and Structured World Models in more detail. If MuZero (Schrittwieser et al., 2020) is restricted to one step predictions, then it is also a concrete example of Definition 5.8; however extending Definition 5.8 to accommodate multi-step predictions should be relatively straightforward. At a high level, these world models work as follows. For each timestep t in round r, after $(\text{obs}_{r,t}, \text{end}_{r,t})$ has been observed, a representation function e^{rep} takes in past observations as input, and outputs a posterior distribution $\mathcal{AS}_{r,t}$ over abstract model states. An abstract state $as_{r,t} \sim \mathcal{AS}_{r,t}$ is then sampled from the distribution $\mathcal{AS}_{r,t}$. Conversely, a transition predictor e^{pred} takes in the current abstract state and a hypothetical agent action, and tries to predict the posterior distribution $\mathcal{AS}_{r,t}$ over abstract states one step into the future, i.e. it outputs a distribution $\widehat{as}_{r,t} = e^{\text{pred}}(as_{(r,t)-1}, a_{(r,t)-1})$. One can think of e^{pred} as defining a simulator on top of an abstract representation of the world given by e^{rep}. Given the abstract state $as_{r,t}$, one can learn predictors for $\text{reward}_{r,s}, \text{obs}_{r,s}$ and $\text{end}_{r,s}$. Learning a predictor for $\text{obs}_{r,s}$ is not strictly necessary, but is sometimes done to encourage the model to learn useful representations. After these components are learned, one can learn a policy e^{π} and Q-function e^{Q} entirely within the learned model. Here, $e^{Q}(as_{r,t}, a_{r,t})$ predicts the expected remaining reward of the current round when in abstract state $as_{r,t}$, when the agent chooses the action $a_{r,t}$ followed by using the policy e^{π}. Unique to our framework, we also learn a domain indicator e^{I}. At the beginning of round r, each expert will evaluate $e^{I}(as_{r,1}) \in [0, 1]$ which indicates the probability that the current round is in the domain of expert e. Each expert will then output the conservative value estimate $e^{I}(as_{r,1}) \times e^{V}(as_{r,1})$, where $e^{V}(as_{r,1})$ is the predicted expected value of using policy e^{π} in the current round. The domain indicator e^{I} is like an N vs 1 classifier in open world learning (Bendale and Boult, 2015) (see Subsection 2.2.4).

In the definition below, the careful reader will notice that we model the represen-

tation function e^{rep} as taking all prior experiences $X_{\leq(r,t)-1}$ as input. This is done to maximize the flexibility of the definition, and to accommodate algorithms such as MuZero. However, the practical implementation of the expert may be much more efficient. For example, the oracle in **IPS-General** might only propose experts whose representation function can be evaluated using the previous abstract state $as_{(r,s)-1}$ instead of the entire history $X_{\leq(r,t)-1}$, as is done in DreamerV3. In this case e^{rep} is still formally a function of the entire history $X_{\leq(r,t)-1}$, but in practice one only needs to keep the previous abstract state $as_{(r,s)-1}$ in memory in order to evaluate e^{rep}.

Definition 5.8. *Given a solution $e \in \{0,1\}^* \to (\{0,1\}^* \cup \{\bot\})$, we define the following world model components:*

1. ***A representation function:*** *let AS be an abstract state domain, and define $\mathcal{AS}_{r,t} = e^{rep}(X_{\leq(r,t)-1}, \{as_{r',t'}\}_{(r',t')<(r,t)}, (obs_{r,t}, end_{r,t})) \in \Delta(AS)$. Let $as_{r,t} \sim \mathcal{AS}_{r,t}$ be a fixed sample from $\mathcal{AS}_{r,t}$.*

2. ***A transition predictor:*** *let $\widehat{as}_{r,t} = e^{pred}(as_{(r,t)-1}, a_{(r,t)-1}) \in \Delta(AS)$ and let $l_{pred} : \Delta(AS) \times \Delta(AS) \to \mathbb{R}$ be an associated loss function.*

3. ***Statistic predictors:*** *For $Y \in \{reward, obs, end\}$, with associated domains $D_{reward} = \mathbb{R}, D_{obs} = OBS, D_{end} = \{0,1\}$, let $\hat{Y}_{r,t} = e^Y(as_{r,t}) \in \Delta(D_Y)$ and let $l_Y : \Delta(D_Y) \times D_Y \to \mathbb{R}$ be an associated loss function.*

4. ***Policy predictors:***

   (a) *Let $e^Q : AS \times \mathcal{A} \to \Delta(\mathbb{R})$ denote the Q function with loss function which depends on e^Q and round data $\tilde{X}_r$, denoted by $l_Q(e^Q, \tilde{X}_r) \in \mathbb{R}$.*

   (b) *Let $\hat{a}_{r,t} = e^\pi(as_{r,t}) \in \Delta(\mathcal{A})$ be the policy function with loss function which depends on the distribution $e^\pi(as_{r,t})$ over actions and the Q function on abstract state $as_{r,t}$, denoted by $l_\pi(e^\pi(as_{r,t}), e^Q(as_{r,t}, \cdot)) \in \mathbb{R}$.*

(c) *Let* $\hat{I}_r = e^I(as_{r,1}) \in [0,1]$ *denote the domain indicator with loss function* $l_I : [0,1] \times \{0,1\} \to \mathbb{R}$.

5. *In round* r *and timestep* t*, after* $(obs_{r,t}, end_{r,t})$ *has been observed, define the expert policy choice* $e^\pi(X_{\leq(r,t)}) := e^\pi(as_{r,t})$ *and expert value prediction* $e^V(X_{\leq(r,t)}) = \hat{I}_r \times \mathbb{E}_{a\sim e^\pi(as_{r,t})}[e^Q(as_{r,t}, a)]$.

Suppose now that we have interacted with the environment for some rounds, and let $\mathcal{D} = \{\tilde{X}_r\}_{r\in W}$ denote the data associated with the interactions which occurred during rounds in the set W. Suppose we have a set of experts $E = \{e_n\}_{n=1}^N$ which we have already learned. We now want to find solutions (experts) which improve our ability to model data $\mathcal{D}$, given that we already have the experts in E. We will solve this problem by passing $(\mathcal{D}, E)$ to **IPS-General**. In order to use **IPS-General**, we need to define (1) the set of valid problem encodings and (2) the solution weighting function $\mathbb{P}(p|e)$. A valid problem encoding will consist of problems $p_1, \ldots, p_k$ where problem $p_i = (W_i, \mathcal{D})$ corresponds to finding a world model for the rounds $W_i \subset W$. We include a dummy problem p_0 which corresponds to using existing experts E in rounds W_0. Together, $W_0, W_1, \ldots, W_k$ form a partition of W. We therefore try to find a way of partitioning the rounds W into subproblems W_i, where each subproblem has its own world model e and policy e^π. The weighting function for solution (expert) e on problem p_i will be derived from the model based reinforcement learning loss of e for the rounds W_i. This loss encourages e to make accurate predictions. It also encourages the domain predictor e^I to output 1 on W_i and 0 on all other rounds.

Definition 5.9. *Given a dataset* $\mathcal{D} = \{\tilde{X}_r\}_{r\in W}$ *consisting of rounds from some set* W*, and a set of expert functions* $E = \{e_n : \{0,1\}^* \to (\{0,1\}^* \cup \{\perp\})\}_{n=1}^N$*, we define the following set of valid problem encodings* $VE((\mathcal{D}, E))$ *and solution weighting function as follows:*

Set of valid problem encodings:

1. *Fix a constant $k_{max} \in \mathbb{N}$ to be the maximum number of problems which can be proposed.*

2. *A valid problem encoding is of the form $p_0, p_1, \ldots, p_k$, where $k \leq k_{max}$ and for $i \in [k]$, $p_i = (W_i, \mathcal{D})$, and for $i = 0$, $p_0 = (W_0, f : W_0 \to [N], \mathcal{D}, E)$, where $W_0, \ldots, W_k$ form a partition of the rounds W of $\mathcal{D}$.*

Solution weighting function: *Given a round r and a function e, the world model loss of e in round r and timestep t is*

$$l_{r,t}(e) = l_{pred}(\widehat{as}_{r,t}, \mathcal{AS}_{r,t}) + \left(\sum_{Y \in \{reward, obs, end\}} l_Y(\hat{Y}_{r,t}, Y_{r,t}) \right) + l_\pi(e^\pi(as_{r,t}), e^Q(as_{r,t}, \cdot)) + l_Q(e^Q, \tilde{X}_r)$$

and in round r we define the round loss as $l_r(e) = \sum_{t=1}^{T_r} l_{r,t}(e)$. For $i \in [k]$, given a problem $p_i = (W_i, \mathcal{D})$ and solution e we define the solution loss as

$$l(p_i|e) = \sum_{r \in W_i} \Big[l_r(e) + l_I(\hat{I}_r, 1) \Big] + \sum_{r \in W \setminus W_i} l_I(\hat{I}_r, 0)$$

and the solution weighting function as $\mathbb{P}(p_i|e) := \exp(-l(p_i|e))$. For $i = 0$, we define the dummy loss (which does not depend on e) as

$$l(p_0|e) = \sum_{r \in W_0} \Big[l_r(e_{f(r)}) + l_I(\hat{I}_{r,e_{f(r)}}, 1) \Big]$$

where $\hat{I}_{r,e_{f(r)}}$ is the corresponding $\hat{I}_r$ for expert $e_{f(r)} \in E$, and we define $\mathbb{P}(p_0|e) = \exp(-l(p_0|e))$.

We can now define the lifelong reinforcement learning algorithm **ALRL**. The algorithm will maintain a set of experts E_r and a knowledge base $\mathcal{KB}_r$, where we initialize $E_1 = \{\text{random-expert}\}$ to contain the world model whose policy samples actions randomly and always estimates reward $e^V = 0$. At the beginning of any round r, we will use **FTGE** to pick an expert $e \in E_r$ based on each expert's reward prediction for the current round. We will then follow the policy e^π for steps $1, \ldots, T_r$, observe the round reward total-reward$_r$, and pass the observed loss 1-total-reward$_r$ to **FTGE** for expert e. For every round r of the form $k_{\max}^2 x^2$ for $x \in \mathbb{N}$, we will pass the previous rounds $\mathcal{D}$ and expert set E_r to **IPS-General**, and get back new solutions $e_{r,1}, \ldots, e_{r,k}$ which we will add to E_{r+1}. Note this ensures that $|E_r| \leq \sqrt{r}$ which is required to control the guarantees of **FTGE**.

Algorithm 10: Abstract Lifelong Reinforcement Learner

Initialize: Expert set $E_1 = \{\text{random-expert}\}$, expert selector ES=**FTGE**(γ) with initial expert set E_1, knowledge base $\mathcal{KB}_1 =$ "", and observed interactions $\mathcal{D}_0 = \emptyset$;
for *round* $r = 1, 2, \ldots$ **do**
  Let e be the choice of ES for the current round;
  for *timesteps* $t = 1, 2, \ldots, T_r$ *of round* r **do**
    Pick action $a_{r,t} \sim e^\pi(X_{\leq(r,t)})$;
  Feed observed loss $l_r = 1 - \text{total-reward}_r$ to ES for expert e in round r;
  $\mathcal{D}_r = \mathcal{D}_{r-1} \cup \{\tilde{X}_r\}$;
  if $r = k_{max}^2 x^2$ *for some* $x \in \mathbb{N}$ **then**
    Draw $e_{r,0}, e_{r,1}, \ldots, e_{r,k_r}, \mathcal{KB}_{r+1} \sim$ **IPS-General**$((\mathcal{D}_r, E_r^*), \mathcal{KB}_r)$ where $E_r^* \subset E_r$ is the subset of (γ, r)-good experts;
    Add experts $\{e_{r,i}\}_{i=1}^{k_r}$ to ES, where $e_{r,i}$ has loss-estimate given by $1 - e_{r,i}^V(X_{\leq r',1})$ for round r';
    $E_{r+1} = E_r \cup \{e_{r,1}, \ldots, e_{r,k_r}\}$;
    $\mathcal{D}_r = \emptyset$;
  else
    $\mathcal{KB}_{r+1} = \mathcal{KB}_r$;
    $E_{r+1} = E_r$;

If the oracle in **IPS-General** proposes every possible program from strings less than a certain length, then **ALRL** can be thought of as an idealized model of a

lifelong reinforcement learner which can be studied theoretically. If the oracle in **IPS-General** consists of "black boxes" of different practical techniques for proposing useful programs, then **ALRL** can be thought of as a practical framework for building a lifelong reinforcement learner, where the framework comes with high level guarantees which guide practical design choices. We finish this section by expanding on what these high level guarantees are.

What reward will ALRL receive in future rounds? Let $E_r^* \subset E_r$ be the set of (γ, r)-good experts. Let $\gamma(R) = o(R)$ be a non-trivial error bound function. From Proposition 5.1, it follows that for any $R \in \mathbb{N}$

$$\frac{1}{R}\sum_{r=1}^{R} \text{total-reward}_r \geq \frac{1}{R}\sum_{r=1}^{R} \max_{e \in E_r^*} e^V(X_{\leq r,1}) + o(1)$$

Therefore, if one wants to guarantee high average reward across future rounds larger than some round R', it suffices to ensure that at some point during rounds $r < R'$, **ALRL** added experts e to its experts set E_r which (1) are γ-good at predicting their own rewards and (2) predict high rewards on the rounds $\geq R'$. This is a formal way of capturing the intuition that if you come across problems you learned to solve in the past, you'll do well on them in the future.

Will ALRL learn to perform well for a certain class of problems? Suppose one would like **ALRL** to perform well on average if it is given rounds from a certain class of RL problems. Let S be a set of solutions (experts) such that (1) any $e \in S$ has a policy e^π which performs well on (solves) this class of RL problems, and (2) the reward and domain predictors e^V and e^I are sufficiently accurate. By the previous paragraph, it suffices that at some round r, **ALRL** adds some $e \in S$ into the expert set E_r. This happens precisely when **IPS-General** proposes an expert $e \in S$ when given data $\mathcal{D}$ of prior environment interactions. How can we be sure this happens?

Proposition 5.7 suggests[9] that this will happen provided that (1) we expose **ALRL** to environment interactions such that the solution weighting function $\mathbb{P}(p|e)$ prefers solutions in S to other solutions, and (2) the oracle proposes at least one solution $e \in S$ with high probability. From a practical perspective, if **ALRL** is failing to perform well on a class of round problems, there are two things which need to be checked:

1. Are the environment interactions **ALRL** is being exposed to, and the choice of weighting function $\mathbb{P}(p|s)$, putting high enough weight on solutions in S? If not, one may consider choosing a more appropriate weighting function (i.e. loss function), or exposing **ALRL** to rounds which have interaction data $\mathcal{D}$ which will put more weight on solutions in $e \in S$ (i.e. curriculum training).

2. Is there some black box in the oracle which proposes solutions from the set S? If not, one may consider engineering a new black box which covers the class of solutions not covered by other black boxes, and adding this to the oracle. In this way, one can modularly add black boxes to the oracle to incrementally cover different classes of problems as the need arises. This might best be illustrated by a concrete example. Suppose one observes **ALRL** is not learning to perform well during rounds where the task is to complete a puzzle which has multiple stages (e.g. solve a rubik's cube). At a high level, one hypothesizes that a class of solutions S which is good at solving a rubik's cube can be encoded by programs which break the task down into tractable subgoals. One decides to explicitly model this idea using Universal Value Function Approximators (Schaul et al., 2015), a technique by which one learns abstract subgoals and a value function $V(as, g)$ which predicts the probability of achieving goal g when in abstract state as. One builds a model-based RL neural network training

[9]Technically the proposition is for **IPS-Oracle** instead of **IPS-General**, but we expect essentially the same result to hold for **IPS-General** modulo marginalizing over the set of possible problem decompositions $\{pe\}_{pe \in \text{Oracle}}$ of the data $\mathcal{D}$.

procedure around this idea, and adds it as a new black box to the system. If one modeled the problem sufficiently well, they would expect this black box to propose solutions which solve the rubik's cube rounds, as well as other problems which can be solved by subgoal planning.

Will ALRL learn specific knowledge $\mathcal{KB}_r$ which can be used by a proposed solution e? In order to be able to reason about which solutions will be proposed by black boxes, one needs to know what knowledge is available to use in $\mathcal{KB}_r$. It is therefore natural to ask whether knowledge α from a particular set Γ will be added to $\mathcal{KB}_r$. Proposition 5.5 suggests that this will happen provided (1) the shared knowledge $\alpha \in \Gamma$ is useful for many solutions on average in the precise sense of condition (1) of the proposition, and (2) some black box proposes at least one instance $\alpha \in \Gamma$ with high probability. In practice, if one notices **ALRL** does not appear to be incrementally learning a certain class of very useful shared knowledge, one can attempt to explicitly engineer a black box which proposes it. This might best be illustrated with a concrete example. Suppose rounds consist of robotic arm manipulation tasks, and suppose one thinks there is a class of solutions S which perform very well at these tasks by decomposing the world into objects and their interactions. For example, maybe red objects magnetically repel other red objects but attract blue objects. And maybe blue objects are slippery and hard to manipulate. In this case, useful shared knowledge $\alpha(A, B)$ could encode a model of how object A interacts with object B for any pair of objects (A, B). In order to be able to propose solutions in the class S, one needs to ensure that $\mathcal{KB}_r$ accumulates knowledge of new object interactions $\alpha(A', B')$ as objects A' and B' are encountered across rounds, so that solutions can be constructed as a function of this knowledge. Conversely, knowledge $\alpha(A', B')$ of how two objects A' and B' interact with each other is probably shared across many solution programs (and so would likely be given high weight by **IPS-General**). In practice, one can

attempt to model this class of knowledge with an incrementally growing graph neural network embedded in a world model. This idea is explored in Section 5.4.

5.4 Incrementally Learning Object And Relation Knowledge

In the previous sections we considered knowledge and solutions in the abstract. In practice, there are a variety of heuristic techniques for learning solutions and knowledge concretely. For example, Section 2.2 reviews some of the different techniques researchers have used to model knowledge in the context of lifelong learning. In the context of deep learning, one of the simplest techniques is weight sharing. For example, suppose there are k classification tasks (problems). One would learn k neural networks, one for each classification task, where each neural network consists of shared parameter weights $\theta \in \mathbb{R}^d$ as well as a small number of task specific weights unique to each task. The shared knowledge is θ. If one is given a new task, they can re-use prior knowledge by learning a neural network whose parameters consist of (1) the fixed weights θ and (2) a small number of learned layers on top of θ. While weight sharing is simple and works in practice for some problems, it suffers from the drawback that it isn't always clear what the "shared knowledge" θ corresponds to, or how to incrementally combine shared knowledge over time. For example, if θ_1 and θ_2 are two distinct shared neural network parameters obtained in different contexts, how do we "combine" this knowledge when learning new problems?

In the previous section, we needed to learn world models to solve RL tasks. In this section, we briefly propose an idea for incrementally learning world models analogously to incrementally learning graph neural networks (Wei et al., 2022). In particular, we model the world as if it can be decomposed into objects and relations. By learning knowledge in terms of atomic object types and the interactions between them, it can be more intuitive to incrementally combine shared knowledge and create growing models over time. We do this by proposing a fusion of two existing techniques: DreamerV3 (Hafner et al., 2023) and Structured World Models (Kipf et al., 2020). We start by reviewing each of these techniques at a high level, and then

giving our model which combines and extends these ideas. We then discuss other model choices and open questions. In this section, we discuss reinforcement learning as defined in the previous section (Definition 5.7), but for simplicity we suppress mentioning distinct rounds and only focus on timesteps.

DreamerV3: DreamerV3 is a model-based reinforcement learning algorithm which achieves high performance across a wide variety of RL environments. The algorithm works by learning a simulator (world model) of the environment, and then learning an RL policy on top of this simulation. The world model is implemented as a Recurrent State-Space Model. More precisely, DreamerV3 learns an abstract representation $as_t = (h_t, z_t)$ of the environment at time t consisting of a discrete stochastic component z_t and a deterministic component h_t. The model updates the abstract representation via $h_{t+1} = f^{\text{rep}}(h_t, z_t, a_t)$ and outputs a posterior distribution over the discrete state z_{t+1} given by $f^{\text{enc}}(h_{t+1}, \text{obs}_{t+1})$. Intuitively, h_t can be thought of as the component of the environment which is deterministic, while z_t is related to the stochastic outcome of the environment which is partially observed at time t. During inference, the model simulates the environment by drawing z_t from the predictive distribution $f^{\text{pred}}(h_t)$. Given an initial state h_1, one can therefore generate a trajectory of abstract states by sampling z_t given h_t, and then computing h_{t+1} given the agent action a_t. Given as_t, the world model also predicts $\text{reward}_t, \text{end}_t$ and obs_t.

Structured world models: Like DreamerV3, Structured World Models (SWMs) is also an approach for learning a world model from environment interactions. SWMs supposes that the environment consists of distinct objects which interact through pairwise interactions, and proposes a graph neural network architecture (Wu et al., 2021) for learning the world model. More precisely, given an image obs_t of the environment at time t, SWMs learns a convolutional neural network f^{CNN} which outputs k channels, one corresponding to each object. Each channel is flattened and fed into a

multi-layer perceptron f^{MLP} to get an object representation $h_t^i = f^{\text{MLP}}(f^{\text{CNN}}(\text{obs}_t)_i)$, where $f^{\text{CNN}}(\text{obs}_t)_i$ denotes the ith channel of the output, and $i \in [k]$. To predict the next abstract state h_{t+1}^i, one computes

$$\hat{h}_{t+1}^i = f^{\text{node}}\left(h_t^i, \sum_{j \neq i} f^{\text{edge}}\left(h_t^i, h_t^j\right), a_t\right)$$

i.e. for $i \in [k]$, each h_t^i corresponds to a node in a graph neural network with pairwise edge interactions specified by the function f^{edge} and node function f^{node}. The world model differs from DreamcoderV3 in that (1) there are no stochastic states, (2) the abstract state only depends on the current observation, and (3) SWMs decompose the abstract state into factored objects with parwise interactions. Kipf et al. (2020) do not demonstrate that SWMs work for training RL agents, but they do show that this neural network architecture appears to automatically separate out distinct objects in ways intuitive to humans.

Our model proposal: For intuition, imagine viewing a 2D plane with distinct objects which interact with each other. For example, imagine there are red and green balls which attract each other, and squares which move in spirals and collide with all other objects. Imagine new objects might spawn in randomly, or fly off the screen. In this case, the world can be described by a collection of objects of different types (red balls, green balls, squares), which interact with each other pairwise depending on their types. Suppose one learns a model for the behaviour and pairwise interactions between each type of object. Now suppose one is given new data coming from the same environment, except that triangles, which repel both red and green balls, are added to the simulation. Ideally, one would like to retain the knowledge of how red and green balls and squares behave, while incrementally learning how triangles interact with these existing object types. Moreover, one would like to do this given one already knows how the other objects interact with each other (which presumably

makes the inference task much easier).

Similar to SWMs, we suppose that at each time t the world can be described by k hidden objects, where the ith object consists of a deterministic component h_t^i, a discrete stochastic component z_t^i, and an object type $\chi_t^i \in [N]$. Analogous to DreamerV3, we define the corresponding abstract state as $as_t^i = (h_t^i, z_t^i)$. In contrast to SWMs, objects are updated as a function of their type, rather than by a single update function for all objects. This allows us to learn separate models for different groups of objects which can be modularly combined. For each object type $a, b \in [N], a \neq b$, we learn the functions f_a^{node} and $f_{a,b}^{\text{edge}}$ which correspond to update functions in a graph neural network. The function $f_{a,b}^{\text{edge}}$ corresponds to the pairwise interaction between objects of types a and b, while f_a^{node} aggregates these interactions into a state update. We then update the deterministic component of an object, and its type, via

$$h_t^i, \chi_t^i = f_{\chi_{t-1}^i}^{\text{node}} \left(as_{t-1}^i, \sum_{\substack{j \in [k] \\ j \neq i}} f_{\chi_{t-1}^i, \chi_{t-1}^j}^{\text{edge}} (as_{t-1}^i, as_{t-1}^j), a_{t-1} \right)$$

Allowing the type of an object to change can be thought of as either (1) the object in the ith slot being destroyed and being replaced by another object, or equivalently (2) as the object in the ith slot transitioning to an object of a different type. As in DreamerV3, we predict the stochastic component z_t^i from h_t^i. For $Y \in \{\text{reward}, \text{end}, \text{obs}\}$, we imagine a dummy object for Y which depends on the k latent objects and is used to predict Y via the model

$$Y_t \sim f_Y^{\text{node}} \left(\sum_{j \in [k]} f_{Y, \chi_t^j}^{\text{edge}} (as_t^j) \right)$$

We use the same idea of dummy objects for defining the value function and RL policy. This allows different object types to have distinct, learnable effects on these predicted quantities. The shared knowledge of this model therefore consists of (1) for each object type $a \in [N]$, $f_a^{\text{node}}, f_a^{\text{pred}}$, and f_a^{enc}, and (2) for each pair of objects $a, b \in [N], a \neq b$, the pairwise interaction term $f_{a,b}^{\text{edge}}$. In the incremental learning setting, one might re-initialize a model with these learned components and learn new components for new objects as required. When learning a new object type a, one might start out assuming that object type a does not interact with any other prior learned objects (i.e. $f_{a,b}^{\text{edge}} = 0$), and then incrementally learn a sparse set of pairwise interactions as is needed to explain the data. Formally, the full model is

$$h_t^i, \chi_t^i = f_{\chi_{t-1}^i}^{\text{node}} \left(as_{t-1}^i, \sum_{\substack{j \in [k] \\ j \neq i}} f_{\chi_{t-1}^i, \chi_{t-1}^j}^{\text{edge}} (as_{t-1}^i, as_{t-1}^j), a_{t-1} \right) \tag{5.57}$$

$$z_t^i \sim f_{\chi_t^i}^{\text{enc}}(h_t^i, \text{obs}_t) \tag{5.58}$$

$$\hat{z}_t^i \sim f_{\chi_t^i}^{\text{pred}}(h_t^i) \tag{5.59}$$

$$\text{reward}_t \sim f_{\text{R}}^{\text{node}} \left(\sum_{j \in [k]} f_{\text{R}, \chi_t^j}^{\text{edge}} (as_t^j) \right) \tag{5.60}$$

$$\text{end}_t \sim f_{\text{E}}^{\text{node}} \left(\sum_{j \in [k]} f_{\text{E}, \chi_t^j}^{\text{edge}} (as_t^j) \right) \tag{5.61}$$

$$\text{obs}_t \sim f_{\text{O}}^{\text{node}} \left(\sum_{j \in [k]} f_{\text{O}, \chi_t^j}^{\text{edge}} (as_t^j) \right) \tag{5.62}$$

$$\text{a}_t \sim f_{\text{A}}^{\text{node}} \left(\sum_{j \in [k]} f_{\text{A}, \chi_t^j}^{\text{edge}} (as_t^j) \right) \tag{5.63}$$

$$V_t = f_{\text{V}}^{\text{node}} \left(\sum_{j \in [k]} f_{\text{V}, \chi_t^j}^{\text{edge}} (as_t^j) \right) \tag{5.64}$$

Discussion: There are many ways one can formulate the high level idea of learning objects with distinct types and interactions into a concrete world model. We think

exploring different model choices is an interesting direction for future work, both from a theoretical and practical perspective. From a practical perspective, one can study which formulations of these world models perform best in continual learning tasks while being benchmarked against their end-to-end optimized equivalents. From a theoretical perspective, one can study which model formulations have good incremental learning properties. For example, suppose one has a model $m = \{f_1, \ldots, f_n\}$ which consists of n components; in the model of this section, the components would consist of the graph neural network components for each object type and their pairwise interactions. A nice theoretical property for an incremental learning model would be that, if the environment changes in a "reasonable" way (e.g. a new object is added), one only needs to update, add or remove a small number of components from the model in order to accommodate this change. The right definition of "reasonable", and how to construct such object-relation models with this property, we believe are interesting questions for future work.[10]

[10]We considered an alternative object relation model which appears to be closer to the ideal of only requiring a constant number of model component updates when a new object is added, but it is considerably more complicated and therefore omitted.

Conclusion

Until we know how to resolve questions like P vs NP, researchers in bounded rationality will need to continue to find indirect ways of modelling and understanding how programs can behave optimally subject to computational constraints. Towards this goal, this thesis introduced three new models, and highlighted further directions for research in each of these areas.

In Chapter 3, we studied how strategic players with restricted processing power trade off memory resources for strategy performance. More broadly, this setting is about how programs can trade off non-uniform advice for processing speed. In computational complexity theory, this question corresponds to studying the complexity class P/poly (polynomial-time algorithms with polynomial length advice strings). A natural question to ask is whether there are programs which run in t seconds, and perform "almost as well" as any other program which runs within t seconds. As argued in Chapter 3, making progress on this problem requires understanding how to restrict program size so that programs do not trivially precompute all the answers. In strategic adversarial settings such as mini-max games, programs invariably require randomization to avoid being exploited by precomputed strategies.

Chapter 4 studied how algorithms can trade off a resource budget for improved regret bounds. Perhaps unexpectedly, guarantees for this problem automatically translate to resource tradeoff results in other problem areas. A natural direction for further research is to systematically study the consequences of **FPML** for algorithms which make use of a regret minimizing subroutine. A collection of such algorithms is detailed

in Arora et al. (2012). Such a study may uncover new resource performance tradeoffs in other application areas. A natural candidate to start with would be boosting (Freund and Schapire, 1996), which uses a multiplicative weights procedure to combine weak base classifiers into a stronger classifier.

Chapter 5 considered a previously unexplored approach to modelling bounded rationality through the lens of lifelong learning. In contrast to other approaches to optimal decision making, LL focuses on modelling the process of learning rather than the final optimal action or policy. Because LL is still a developing field, there is significant potential for developing theory in this area. The main challenge is giving theoretical models which have useful guarantees, but do not make rigid distributional assumptions and can make use of practical techniques like neural networks. The work in sections 5.2 and 5.3 is a first step towards this goal. A natural continuation would be to incorporate algorithm runtimes and resource tradeoffs into the theory. One way to do this is to meta model how programs are searched for by charging a utility cost for computation, similar to prior work in meta reasoning and bandit runtime algorithms. Another direction for future work is to explore practically building a system based on **IPS-General**. This might involve building and testing the object relation model in Section 5.4, and finding ways to share knowledge between existing black box techniques in the literature in a way which is consistent with the theory of Section 5.2.

Appendix A

Appendix For Chapter 3

Theorem A.1. *(Hoeffding's inequality.) Let $X_1, \ldots, X_n$ be independent random variables where $\forall i \in [n], a_i \leq X_i \leq b_i$ almost surely. Let $S = \sum_{i=1}^{n} X_i$. Then for any $t > 0$,*

$$\mathbb{P}\left[|S - \mathbb{E}[S]| \geq t\right] \leq 2\exp\left(-\frac{2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right)$$

Proof of Lemma 3.4:

Proof. The proof is fairly long, and mostly consists of definition checking. We split it up into steps for easy readability.

1. **Step 1:** Argue that the claim follows if we can show that for any pure strategies $\tilde{\sigma}_1, \tilde{\sigma}_2$ for G, there exists pure deterministic strategies $\tilde{\sigma}'_1, \tilde{\sigma}'_2$ for G' with the same utility $\tilde{u}_1^{(\tilde{\sigma}_1, \tilde{\sigma}_2)} = (u')_1^{(\tilde{\sigma}'_1, \tilde{\sigma}'_2)}$, and vice versa.

2. **Step 2:** Define a bijection between a pure strategy $\tilde{\sigma}_i$ in G and a pure deterministic strategy $\tilde{\sigma}'_i$ in G', and check it is well defined.

3. **Step 3:** Define the weighting functions $z_1, z_2 : H' \to [0, 1]$. Show that $\tilde{u}_1^{(\tilde{\sigma}_1, \tilde{\sigma}_2)} = (u')_1^{(\tilde{\sigma}'_1, \tilde{\sigma}'_2)}$ under the bijective correspondence defined in step 2. We do this by breaking $(u')_1^{(\tilde{\sigma}'_1, \tilde{\sigma}'_2)}$ into two parts.

(a) **Step 3A:** Show $\sum_{h' \in Z'} (\pi')^{\tilde{\sigma}'}(h') u_1(Q(h')) = \sum_{h \in Z} \pi^{\tilde{\sigma}}(h) u_1(h)$.

(b) **Step 3B:** Show $\sum_{h' \in Z'} (\pi')^{\tilde{\sigma}'}(h') z_i(h') = |S_i|$.

Step 1: Suppose there is a bijection f such that for any pure strategies $\tilde{\sigma}_1, \tilde{\sigma}_2$ for G, there are pure deterministic strategies $\tilde{\sigma}'_1 = f(\tilde{\sigma}_1), \tilde{\sigma}'_2 = f(\tilde{\sigma}_2)$ for G' where $\tilde{u}_1^{(\tilde{\sigma}_1, \tilde{\sigma}_2)} = (u')_1^{(\tilde{\sigma}_1, \tilde{\sigma}'_2)}$. Then we know that there is also a bijective relationship between mixed strategies in G and mixed deterministic strategies in G'. By Kuhn's theorem, we know that every mixed deterministic strategy in game G' has an equivalent behavior strategy in game G' (because G' is an extensive form perfect recall game). Conversely, any behaviour strategy in G' has an equivalent mixture of deterministic strategies in G'. To see this, note that a behaviour strategy $\tilde{\sigma}'_i$ can be arbitrarily approximated by a deterministic function of the history h' and a random input r sampled uniformally in $\{0,1\}^d$ for sufficiently large d, i.e. if we randomly sample r, then the joint distribution of the random variables $\{\tilde{\sigma}'_i(h', r)\}_{h' \in H', P(h')=i}$ is arbitrarily close in distribution to $\{\tilde{\sigma}'_i(h')\}_{h' \in H', P(h')=i}$. The mixture of deterministic strategies $\{\tilde{\sigma}'_i(\cdot, r)\}_{r \in \{0,1\}^d}$ will therefore approximate any behaviour strategy $\tilde{\sigma}'_i$ to an arbitrary degree of accuracy. To summarize, assuming the bijection f, there is a correspondence between mixed strategies in game G and mixed deterministic strategies in game G'. There is also a correspondence between mixed deterministic strategies in game G', and behaviour strategies in game G'. The conclusion of Lemma 3.4 then follows.

Step 2: Fix any pure strategies $\tilde{\sigma}_i \in Pre(\sigma_{pre}, \sigma_i)$ for $i \in \{1, 2\}$ in the meta-precomputation game G, where $\tilde{\sigma}_i$ has memorization set S_i. Recall that for every $h' \in H'$ for the game G' where $P(h') \neq c$, there is an associated history $Q(h') = h$ in game G. We map $\tilde{\sigma}_i$ to the pure deterministic strategy $\tilde{\sigma}'_i$ in game G', where player i chooses action 1 (to precompute) at history h' iff $Q(h') \in S_i$. We want to show that this mapping defines a valid policy in game G' (i.e. respects information sets in game G') and is bijective. To show that $\tilde{\sigma}'_i$ is a valid policy, we need to show $\tilde{\sigma}'_i(x) = \tilde{\sigma}'_i(y)$

for any $x, y \in H'$ in the same information set, and that player i never chooses to precompute in game G' if they previously chose to stop precomputing. The latter property follows because S_i is prefix closed. The former property is immediate, since if x, y are in the same information set, the actions of the chance player which lead to x or y must be equal, which means that $Q(x) = Q(y)$, so $\tilde{\sigma}'_i(x) = \tilde{\sigma}'_i(y)$. On the other hand, if $\tilde{\sigma}'_i$ is a pure deterministic strategy for game G', then consider the pure strategy $\tilde{\sigma}_i$ in game G where $h \in S_i$ iff I is an information set where player i has the option of choosing to precompute, and player i chooses to precompute at I where $h' \in I$ and $h = Q(h')$. We claim that this definition is also well defined. To see this, note that if player i has the option to precompute at information set I, then player i must have chosen to precompute at every available action before I. Recall that I is uniquely determined by the sequence of choices of player i and the chance player leading up to I. Since the actions of player i are already determined, any information set I during which player i has the option to precompute in game G' can be uniquely associated with the sequence of actions of the chance player leading up to I, and is therefore uniquely associated with a history $h \in H$. Therefore the reverse mapping is also well defined.

Step 3: Finally, we need to show that

$$\tilde{u}_1^{(\tilde{\sigma}_1, \tilde{\sigma}_2)} = (u')_1^{(\tilde{\sigma}'_1, \tilde{\sigma}'_2)} \tag{A.1}$$

which shows that corresponding pure strategies in each game have the same values. We begin by defining the functions z_i for $i \in \{1, 2\}$. For $h' \in H'$, let $\pi'_c(h')$ be the probability of reaching h' where we only take into account the chance player (i.e. the product of the probabilities of all edges leading to h' where the chance player chooses an action). Now fix any terminal history $h' \in Z'$ for game G'. Without loss

of generality, we assume that $\forall h' \in Z'$, $\pi'_c(h') > 0$ (otherwise we can remove h' from the game without affecting the utility). Let $M_i(h') = \{h'_{i,1}, \ldots, h'_{i,j_i}\}$ be the set of histories $h'_{i,1} < h'_{i,2}, \ldots, h'_{i,j_i} < h'$ where player i chose to memorize (action 1) in order to reach history h'. Then we define

$$z_i(h') := \sum_{h'' \in M_i(h')} \frac{1}{\pi'_c(h'')} \tag{A.2}$$

The intuition behind this choice is that if player i chose to precompute at history h'', then we want each terminal history $h' \in Z'$ which follows from h'' to contain a term $\frac{1}{\pi'_c(h'')}$. The sum of these terms over terminal children h' of h'', weighted by their probability, will then be equal to 1. This ensures that $\sum_{h' \in Z'} (\pi')^{\tilde{\sigma}'}(h') z_i(h') = |S_i|$.

Let $\tilde{\sigma} = (\tilde{\sigma}_1, \tilde{\sigma}_2)$ and $\tilde{\sigma}' = (\tilde{\sigma}'_1, \tilde{\sigma}'_2)$. If Z' is the set of terminal histories for G', we need to compute

$$(u')_1^{\tilde{\sigma}'} = \sum_{h' \in Z'} (\pi')^{\tilde{\sigma}'}(h')[u_1(Q(h')) - \lambda_1 z_1(h') + \lambda_2 z_2(h')]$$

Step 3A: Focusing on the first term, we claim

$$\sum_{h' \in Z'} (\pi')^{\tilde{\sigma}'}(h') u_1(Q(h')) = \sum_{h \in Z} \pi^{\tilde{\sigma}}(h) u_1(h)$$

To see this, let $h = Q(h')$. By construction of $\tilde{\sigma}$ and $\tilde{\sigma}'$, $(\pi')^{\tilde{\sigma}'}(h') = \pi'_c(h') = \pi^{\tilde{\sigma}}(h)$ if and only if history h' corresponds to players 1 and 2 choosing to precompute according to memorization sets S_1, S_2 up until history h', and 0 otherwise. This is because $\tilde{\sigma}'$ consists of deterministic strategies, i.e. for any h' where $(\pi')^{\tilde{\sigma}'}(h') > 0$, $(\pi')^{\tilde{\sigma}'}(h') = \pi'_c(h')$ because the probabilities on the edges of the game tree leading up to h' where players 1 and 2 make moves are all 1, and so only the probability

contribution from the chance player is relevant. Conversely, if h' doesn't correspond to choosing to precompute according to memorization sets S_1, S_2, then some edge will have probability 0 and hence $(\pi')^{\tilde{\sigma}'}(h') = 0$. Therefore, for any $h \in H$, there is exactly one $h' \in Q^{-1}(h)$ where $(\pi')^{\tilde{\sigma}'}(h') > 0$. Write $h' = Q_*^{-1}(h)$ for this h'. Let $h \in Z$. It follows that $\sum_{h' \in Q^{-1}(h)} (\pi')^{\tilde{\sigma}'}(h') = (\pi')^{\tilde{\sigma}'}(Q_*^{-1}(h)) = \pi^{\tilde{\sigma}}(h)$. The equality follows since $\{Q^{-1}(h)\}_{h \in Z}$ forms a partition of Z'.

Step 3B: Moving our attention to the remaining two terms in

$$(u')_1^{\tilde{\sigma}'} = \sum_{h' \in Z'} (\pi')^{\tilde{\sigma}'}(h')[u_1(Q(h')) - \lambda_1 z_1(h') + \lambda_2 z_2(h')]$$

we have

$$\sum_{h' \in Z'} (\pi')^{\tilde{\sigma}'}(h') z_i(h') = \sum_{h' \in Z'} (\pi')^{\tilde{\sigma}'}(h') \sum_{h'' \in M_i(h')} \frac{1}{\pi'_c(h'')} \tag{A.3}$$

$$= \sum_{h' \in Z'} (\pi')^{\tilde{\sigma}'}(h') \sum_{\substack{h'' < h' \\ Q(h'') \in S_i}} \frac{1}{\pi'_c(h'')} \tag{A.4}$$

$$= \sum_{h \in S_i} \sum_{\substack{h' \in Q^{-1}(h) \\ (\pi')^{\tilde{\sigma}'}(h') > 0}} \frac{1}{\pi'_c(h')} \sum_{\substack{h'' \in Z' \\ h'' > h' \\ (\pi')^{\tilde{\sigma}'}(h'') > 0}} \pi'_c(h'') \tag{A.5}$$

$$= \sum_{h \in S_i} \sum_{\substack{h' \in Q^{-1}(h) \\ (\pi')^{\tilde{\sigma}'}(h') > 0}} \sum_{\substack{h'' \in Z' \\ h'' > h' \\ (\pi')^{\tilde{\sigma}'}(h'') > 0}} \pi'_c(h', h'') \tag{A.6}$$

$$= \sum_{h \in S_i} \sum_{\substack{h' \in Q^{-1}(h) \\ (\pi')^{\tilde{\sigma}'}(h') > 0}} \sum_{\substack{h'' \in Z' \\ h'' > h' \\ (\pi')^{\tilde{\sigma}'}(h'') > 0}} (\pi')^{\tilde{\sigma}'}(h', h'') \tag{A.7}$$

$$= \sum_{h \in S_i} \sum_{\substack{h' \in Q^{-1}(h) \\ (\pi')^{\tilde{\sigma}'}(h') > 0}} 1 \tag{A.8}$$

$$= \sum_{h \in S_i} 1 \tag{A.9}$$

$$= |S_i| \tag{A.10}$$

Lines A.3 and A.4 follow from the definitions of z_i and M_i. In line A.5, we reverse the order of the summation. Instead of fixing a terminal history $h' \in Z'$ and summing over all ancestors $h'' < h'$ where player i chose to precompute, we sum over all histories h' where player i chose to precompute (i.e. $h' \in Q^{-1}(h)$ where $h \in S_i$ and h' has positive support on $\tilde{\sigma}'$), and then over the relevant terminal histories. Line A.6 is just a multiplicative identity. Line A.7 follows because the product of the action probabilities of each player are 1 on histories with positive support. Line A.8 follows because the probability that we reach some terminal history when starting from history h' is 1. As discussed before, note that the summation $\sum_{\substack{h' \in Q^{-1}(h) \\ (\pi')^{\tilde{\sigma}'}(h') > 0}}$ is over exactly one element (using the fact that we are only considering deterministic

strategies). Collecting these results, we find that

$$\begin{aligned}(u')_1^{\tilde{\sigma}'} &= \sum_{h \in Z} \pi^{\tilde{\sigma}}(h) u_1(h) - \lambda_1 |S_1| + \lambda_2 |S_2| \\ &= \tilde{u}_1^{\tilde{\sigma}}\end{aligned}$$

showing the required equivalence. Lastly, for a policy profile $\tilde{\sigma}'$ where each player plays deterministically, it is easy to see that

$$\begin{aligned}\sum_{h' \in Z'} \pi^{\tilde{\sigma}'}(h') |u_1(h')| &\leq \sum_{h' \in Z'} \pi^{\tilde{\sigma}'}(h')(1 + \lambda_1 z_1(h') + \lambda_2 z_2(h')) \\ &= 1 + \lambda_1 |S_1| + \lambda_2 |S_2|\end{aligned}$$

where $|S_1|, |S_2|$ are the size of the memorization sets for players 1 and 2 respectively.

□

Proof of Lemma 3.5:

Proof. This proof requires some familiarity with Zinkevich et al. (2007), although the adaption is very minor. Given a policy profile σ, let $\sigma|_{I \to a}$ be the policy profile identical to σ but where player i plays action a when in information set I. Let $\pi^{\sigma}_{-i}(h)$ be the probability of reaching history h where player i always plays deterministically to reach the information set which contains h (i.e. the action probabilities for player i along the path of history h are all 1). In the proof of Theorem 4 in Zinkevich et al. (2007), the value of C in Lemma 3.5 comes from bounding

$$\sum_{h \in I, h' \in Z'} \pi^{\sigma}_{-i}(h) \pi^{\sigma|_{I \to a}}(h, h') u_i(h') - \sum_{h \in I, h' \in Z'} \pi^{\sigma}_{-i}(h) \pi^{\sigma}(h, h') u_i(h') \tag{A.11}$$

which is done using $\Delta_{u,i}$, the maximum difference in utility of terminal histories for player i. Instead, we bound this value by

$$\sum_{h\in I,h'\in Z'} \pi^{\sigma}_{-i}(h)\pi^{\sigma|_{I\to a}}(h,h')u_i(h') - \sum_{h\in I,h'\in Z'} \pi^{\sigma}_{-i}(h)\pi^{\sigma}(h,h')u_i(h') \tag{A.12}$$

$$\leq \sup_{\sigma',\sigma\in X} \sum_{h\in I,h'\in Z'} \pi^{\sigma'}(h)\pi^{\sigma'}(h,h')u_i(h') - \sum_{h\in I,h'\in Z'} \pi^{\sigma}(h)\pi^{\sigma}(h,h')u_i(h') \tag{A.13}$$

$$\leq 2\sup_{\sigma\in X} \left| \sum_{h\in I,h'\in Z'} \pi^{\sigma}(h)\pi^{\sigma}(h,h')u_i(h) \right| \tag{A.14}$$

$$\leq 2\sup_{\sigma\in X} \sum_{h\in I,h'\in Z'} \pi^{\sigma}(h)\pi^{\sigma}(h,h')|u_i(h')| \tag{A.15}$$

$$\leq 2\sup_{\sigma\in X} \sum_{h'\in Z'} \pi^{\sigma}(h')|u_i(h')| \tag{A.16}$$

The remainder of the result is identical to what is shown in Zinkevich et al. (2007). □

Experimental Details: Experiments were run on a single thread of a Intel Xeon E5-2678 v3 2.5GHz CPU for around 3 hours per plot. While it is feasible to run this experiment in full with more hardware, we make the following key simplifications to make the computation manageable on a desktop:

1. Instead of considering all possible actions, we limit policies to play the from the top $k = 2$ choices recommended by the Stockfish engine.

2. We set the maximum game length to $L = 100$ half moves; a draw with utility 0.5 is declared beyond this point.

3. Instead of explicitly estimating the utility of each board state, we use Stockfish to compute the centerpawn (cp) score of each board state. If there is a centerpawn advantage of ≥ 400 for any player, then this is calculated as a win for that player ($u = 1$ or 0). Otherwise the utility value is given as a draw ($u = 0.5$). This choice (as opposed to e.g. estimating the expected value as a

linear function of the cp score) was made to give the qualitative interpretation of the results a more conservative lower bound. In particular, we can interpret the utility of the precomputing player as a qualitative indication of the fraction of the time they are able to reach an overwhelming advantage against their opponent (where we can be reasonably sure they would win if playing normally from that point onwards). In contrast, it isn't clear that e.g. a 50cp score for white (scored by Stockfish(50ms)) would translate to Stockfish(10ms) having a slightly higher probability of winning, because Stockfish(10ms) may not be powerful enough to make use of that advantage.

Appendix B

Appendix For Chapter 4

B.1 Proofs

Lemma B.1. *(Independence lemma for Lemma 4.2) Let $X_1, \ldots, X_K$ be jointly independent continuous random variables. Let $i_1, \ldots, i_k$ and $v_{i_1}, \ldots, v_{i_k}$ be the indices and values of the largest $k < K$ random variables, and let $X := \{X_i | i \notin \{i_1, \ldots, i_k\}$ be the smallest $K - k$ random variables. Then conditional on $(i_j, v_{i_j})_{j\in[k]}$, the distributions of each $X_i \in X$ are jointly independent. Moreover, the marginal distribution $X_i|(i_j, v_{i_j})_{j\in[k]}$ for $i \notin \{i_1, \ldots, i_k\}$ is $X_i|X_i \leq \min_{j\in[k]} v_{i_j}$.*

Proof. Let $M := \min_{j\in[k]} v_{i_j}$. The conditional joint density function is

$$
\begin{aligned}
f(X_1, \ldots, X_K|(i_j, v_{i_j})_{j\in[k]}) &\propto f(X \wedge (i_j, v_{i_j})_{j\in[k]}) && \text{(B.1)}\\
&= \prod_{j\in[K]-\{i_1,\ldots,i_k\}} f(X_j) \prod_{j\in\{i_1,\ldots,i_k\}} f(X_j = v_j)\\
&\times \prod_{j\in[K]-\{i_1,\ldots,i_k\}} \mathbb{1}[X_j \leq M] && \text{(B.2)}\\
&\propto \prod_{j\in[K]-\{i_1,\ldots,i_k\}} f(X_j)\mathbb{1}[X_j \leq M] && \text{(B.3)}
\end{aligned}
$$

i.e. the joint density factorizes for each X_j (which implies joint independence), and marginally the density for $X_j \in X$ is $\propto f(X_j)\mathbb{1}[X_j \leq M]$ which gives the required result. □

Proof of Lemma 4.4

Proof. Fix $t \in [T]$ and let $V \coloneqq \min_{a \in \mathcal{A}-S_t} \hat{C}^*_{t-1}(a)$. So for any a, $a \in S_t$ iff $\hat{C}^*_{t-1}(a) < V$. Define the event $E_a \coloneqq \{\hat{C}^*_{t-1}(a) < V - K\}$; if this holds then a must have been ahead of every action $a' \notin S_t$ by at least K and therefore cannot be overtaken by any such action, since the estimated costs are all upper-bounded by K. So

$$\{a \text{ overtaken by some } a' \notin S_t\} \subset E_a^c.$$

Note that

$$\begin{aligned}
\{a_t^* \notin S_t\} &= \{\exists a' \in \mathcal{A} - S_t : \forall a \in S_t,\ a' \text{ overtakes } a \text{ at round } t\} && \text{(B.4)}\\
&= \bigcup_{a' \in \mathcal{A}-S_t} \bigcap_{a \in S_t} \{a' \text{ overtakes } a \text{ at round } t\} && \text{(B.5)}\\
&\subset \bigcap_{a \in S_t} \bigcup_{a' \in \mathcal{A}-S_t} \{a' \text{ overtakes } a \text{ at round } t\} && \text{(B.6)}\\
&= \bigcap_{a \in S_t} \{a \text{ overtaken by some } a' \notin S_t\} && \text{(B.7)}
\end{aligned}$$

Let $\mathcal{G}_t \coloneqq \sigma(S_t, (\hat{C}^*_{t-1}(a))_{a \notin S_t})$ be the σ-algebra generated by the random set S_t and the current perturbed estimated cumulative costs of the actions not in it. We have

$$\begin{aligned}
\mathbb{P}\left(a_t^* \notin S_t \mid \mathcal{G}_t\right) &\leq \mathbb{P}\left(\bigcap_{a \in S_t} \{a \text{ overtaken by some } a' \notin S_t\} \mid \mathcal{G}_t\right) && \text{(B.8)}\\
&\leq \mathbb{P}\left(\bigcap_{a \in C} E_a^c \mid \mathcal{G}_t\right) && \text{(B.9)}\\
&= \mathbb{P}\left(\bigcap_{a \in S_t} \{\hat{C}^*_{t-1}(a) < V - K\} \mid \mathcal{G}_t\right). && \text{(B.10)}
\end{aligned}$$

But, since $V = \min_{a \in \mathcal{A}-S_t} \hat{C}^*_{t-1}(a)$, applying Lemma B.1 (using the assumption that the perturbation p is independent of $\hat{C}_{t-1}$) gives us that

$$\mathbb{P}\left(\bigcap_{a \in S_t} \{\hat{C}^*_{t-1}(a) < V - K\} \mid \mathcal{G}_t\right) = \prod_{a \in S_t} \mathbb{P}\left(\hat{C}^*_{t-1}(a) < V - K \mid \hat{C}^*_{t-1}(a) \leq V\right) \tag{B.11}$$

By the memoryless property of the exponential distribution, we get the same result as before:

$$1 - \mathbb{P}\left(p(a) \geq \hat{C}_{t-1}(a) - V + K \mid p(a) \geq \hat{C}_{t-1}(a) - V\right) \leq 1 - \mathbb{P}(p(a) \geq K) \tag{B.12}$$

$$= 1 - \mathrm{e}^{-K\varepsilon} \tag{B.13}$$

Thus $\mathbb{P}(a_t^* \notin S_t \mid \mathcal{G}_t) \leq (1 - \mathrm{e}^{-K\varepsilon})^B$. This immediately implies that $\mathbb{P}(a_t^* \notin S_t) \leq (1 - \mathrm{e}^{-K\varepsilon})^B$. The result then follows, since $\mathbb{E}[|\mathcal{I}|] = \sum_{t=1}^{T} \mathbb{P}(a_t^* \notin S_t) \leq T(1 - \mathrm{e}^{-K\varepsilon})^B$. □

Proof of Lemma 4.5

Proof. For each $j \in [B_1]$ write $\Delta_j \coloneqq f(S^*_{B_0}) - f(\bar{G}_j)$. Following Streeter and Golovin (2008), for any $j \in [B_1]$, $b > 0$ and $S \in \mathcal{S}$ with $\ell(S) \leq b$,

$$f(S) \leq f(\bar{G}_j) + b \cdot (s_j + \varepsilon_j),$$

where

$$s_j \coloneqq \max_{a \in \mathcal{A}} f(\bar{G}_j \oplus \langle a \rangle) - f(\bar{G}_j) = f(\bar{G}_j \oplus \bar{g}_j) - f(\bar{G}_j) = f(\bar{G}_{j+1}) - f(\bar{G}_j),$$

so in particular for any $j \in [B_1]$

$$f(S^*_{B_0}) = \max_{S \in \mathcal{S}: \ell(S) = B_0} f(S) \leq f(\bar{G}_j) + B_0 \cdot (s_j + \varepsilon_j) \tag{B.14}$$

$$= f(\bar{G}_j) + B_0 \left(f(\bar{G}_{j+1}) - f(\bar{G}_j) + \varepsilon_j\right) \tag{B.15}$$

$$= f(\bar{G}_j) + B_0 \left(\Delta_j - \Delta_{j+1} + \varepsilon_j\right), \tag{B.16}$$

giving $\Delta_j \leq B_0 (\Delta_j - \Delta_{j+1} + \varepsilon_j)$. Rearranging gives $\Delta_{j+1} \leq \Delta_j \left(1 - \frac{1}{B_0}\right) + \varepsilon_j$ for each j, and unrolling this inequality and using that $1 - \frac{1}{B_0} < 1 \; \forall j \in [B_1]$ as in Streeter and Golovin (2008) gives us

$$\Delta_{B_1+1} \leq \Delta_1 \prod_{j=1}^{B_1} \left(1 - \frac{1}{B_0}\right) + \sum_{j=1}^{B_1} \varepsilon_j \tag{B.17}$$

$$< \Delta_1 \mathrm{e}^{-B_1/B_0} + \sum_{j=1}^{B_1} \varepsilon_j \tag{B.18}$$

$$\tag{B.19}$$

using $1 - x < e^{-x}$ for $x > 0$. Therefore

$$f(S^*_{B_0}) - f(\bar{G}_{L+1}) = \Delta_{B_1+1} \tag{B.20}$$

$$< \Delta_1 \mathrm{e}^{-B_1/B_0} + \sum_{j=1}^{B_1} \varepsilon_j \tag{B.21}$$

$$\leq f(S^*_{B_0}) \mathrm{e}^{-B_1/B_0} + \sum_{j=1}^{L} \varepsilon_j \tag{B.22}$$

giving $f(\bar{G}) > (1 - \mathrm{e}^{-B_1/B_0}) f(S^*_{B_0}) - \sum_{j=1}^{B_1} \varepsilon_j$ as required. □

Proof of Lemma 4.6

Proof. Consider the quantity $\rho_{B,B'} \coloneqq \left(1 - \mathrm{e}^{-B/B'}\right) \sum_{t=1}^{T} f_t(S^*_{B'}) - \sum_{t=1}^{T} f_t(S_t)$. As argued in Streeter and Golovin (2008), we may view the sequence of actions $a^i_1, \ldots, a^i_T$ selected by each experts algorithm $\mathcal{E}_i$ as a single 'meta-action' $\tilde{a}_i \in \mathcal{A}^T$; so the schedules $S_1, \ldots, S_T$ output by **OG** can be viewed as a single 'meta-schedule' $\tilde{S} = \langle \tilde{a}_1, \ldots, \tilde{a}_B \rangle$ over $\mathcal{A}^T$ which is a version of the greedy schedule $\bar{G}_{B+1}$ for the job $f = \frac{1}{T} \sum_{t=1}^{T} f_t$. Let

$$\rho_{B,B'} = T \left[\left(1 - \mathrm{e}^{-B/B'}\right) f(S^*_{B'}) - f(\tilde{S}) \right] \tag{B.23}$$

Applying Lemma 4.5 with $B_0 = B'$, $B_1 = B$ then gives

$$\rho_{B,B'} < T \sum_{j=1}^{B} \varepsilon_j \tag{B.24}$$

Taking expectations,

$$\mathbb{E}[\rho_{B,B'}] \leq T \sum_{j=1}^{B} \mathbb{E}[\varepsilon_j] \tag{B.25}$$

$$= T \sum_{j=1}^{B} \mathbb{E}\left[\frac{R_{T,1}(\mathcal{E}_j)}{T}\right] \tag{B.26}$$

where $R_{T,1}(\mathcal{E}_j)$ is the 1-regret incurred by the j^{th} experts algorithm. So $\mathbb{E}[\rho_{B,B'}] \leq \mathbb{E}\left[\sum_{j=1}^{B} R_{T,1}(\mathcal{E}_j)\right]$. Since $B \geq B' \log T$, $\mathrm{e}^{-B/B'} \leq \mathrm{e}^{-\ln T} = T^{-1}$. Thus

$$\rho_{B,B'} \geq \left(1 - T^{-1}\right) \sum_{t=1}^{T} f_t(S^*_{B'}) - \sum_{t=1}^{T} f_t(S_t) \tag{B.27}$$

$$= R_{B'} - T^{-1} \sum_{t=1}^{T} f_T(S^*_{B'}) \tag{B.28}$$

$$\geq R_{B'} - 1 \tag{B.29}$$

where $R_{B'} \coloneqq \sum_{t=1}^{T} f_t(S^*_{B'}) - \sum_{t=1}^{T} f_t(S_t)$ is the regret of interest. □

B.2 Experiments

Full comparison of $\mathbf{OG_{hybrid}}$

In Table B.1 we give a more detailed comparison of **FPML** and **OG** with various instantiations of $\mathbf{OG_{hybrid}}$ on the hyperparameter-selection task from Section 4.4. Specifically, we include for each B and each possible pair (B_1, B_2) s.t. $B_1 B_2 = B$ a version of $\mathbf{OG_{hybrid}}$ with B_2 internal boxes and arm budget B_1 per box. As can be seen, in all cases decreasing the greediness and adding more arms per box is beneficial in this application.

Table B.1: Sample means and standard deviations of normalized validation scores of **FPML**, $\mathbf{OG_{hybrid}}$ and **OG** over black-box optimizers.

(a) $B = 1$

	Mean	StD
Best in hindsight	0.574	0
FPML	0.426	0.0202
Exp3	0.351	0.0194

(b) $B = 2$

	Mean	StD
Best in hindsight	0.710	0
FPML	0.652	0.0194
$\mathbf{OG_{hybrid}}$ $((B_1, B_2) = (1, 2))$	0.577	0.0187
OG	0.519	0.0179

(c) $B = 3$

	Mean	StD
Best in hindsight	0.779	0
FPML	0.751	0.0151
$\mathbf{OG_{hybrid}}$ $((B_1, B_2) = (1, 3))$	0.657	0.0191
OG	0.617	0.0166

(d) $B = 4$

	Mean	StD
Best in hindsight	0.836	0
FPML	0.813	0.0108
$\mathbf{OG_{hybrid}}$ $((B_1, B_2) = (2, 2))$	0.756	0.0149
$\mathbf{OG_{hybrid}}$ $((B_1, B_2) = (1, 4))$	0.716	0.0178
OG	0.689	0.0151

(e) $B = 5$

	Mean	StD
Best in hindsight	0.874	0
FPML	0.855	0.0094
$\mathbf{OG_{hybrid}}$ $((B_1, B_2) = (1, 5))$	0.756	0.0150
OG	0.734	0.0140

(f) $B = 6$

	Mean	StD
Best in hindsight	0.901	0
FPML	0.888	0.0072
$\mathbf{OG_{hybrid}}$ $((B_1, B_2) = (3, 2))$	0.836	0.0111
$\mathbf{OG_{hybrid}}$ $((B_1, B_2) = (2, 3))$	0.814	0.0143
$\mathbf{OG_{hybrid}}$ $((B_1, B_2) = (1, 6))$	0.785	0.0137
OG	0.767	0.0157

Synthetic tasks

In this section we evaluate our algorithms on three synthetic tasks. In all cases,

- let S^* be the best-in-hindsight set of B arms;
- let S_{greedy} be the greedy choice of B arms in hindsight;
- let S_{top} be the individual top B arms in hindsight.

Task 1: The first environment is one where $S^* = S_{\text{greedy}}$ and this set does better than S_{top}; greediness is better than picking the individual top B arms. There are $|\mathcal{A}| = 15$ available arms and two types of round, A and B, which occur with equal probability; costs are distributed within each round according to Table B.2. So the best fixed arm set of any size up to 10 will be split evenly across arms $\{1, 2, 3, 4, 5\}$ and arms $\{11, 12, 13, 14, 15\}$—and will be the greedy choice—but for $B \leq 5$ the top

B arms will always be in $\{1, 2, 3, 4, 5\}$. We observe that FPML does not outperform OG on this task.

Task 2: The second environment is one where (approximately) $S^* = S_{\text{greedy}} = S_{\text{top}}$; greediness is good but no better than picking the top B arms. There are $|\mathcal{A}| = 10$ available arms and costs are distributed according to Table B.3; because there are no groups of anticorrelated actions, the performance gap between the best set and the top B arms is trivially small. We observe that FPML outperforms OG on this task.

Task 3: The third environment is one where $S^* = S_{\text{top}}$ and this set does better better than S_{greedy}; greediness is worse than just picking the top B arms. Suppose there are $|\mathcal{A}| = 4$ available arms and a budget of $B = 3$. Costs are deterministic and listed in Table B.4 for some parameter δ which we set to 0.01. The top 3 arms are $S_{\text{top}} = \{1, 3, 4\}$ and this is also the best-in-hindsight set S^*, incurring minimum cost 0 at each round. A quick calculation shows that the greedy choice S_{greedy} is either $\{1, 2, 3\}$ or $\{1, 2, 4\}$, though, and either of these sets incur an average minimum cost of $1/8 - \delta/4$, substantially higher. Our empirical results in Table B.5 show this gap in practice.

Table B.2: Cost distributions for round types A and B in the first synthetic environment; Beta distributions are parameterized by mean and variance, not shape.

Arm	A**-rounds**	B**-rounds**	**Resulting mean**
Actions 1 to 5	Beta$(0.4, 0.01)$	Always 1	0.7
Actions 6 to 10	Beta$(0.6, 0.01)$	Always 1	0.8
Actions 11 to 15	Always 1	Beta$(0.8, 0.01)$	0.9

Table B.3: Cost distributions in the second synthetic environment.

Arm	Distribution
1	$\text{Beta}(0.4, 0.01)$
2	$\text{Beta}(0.45, 0.01)$
3	$\text{Beta}(0.5, 0.01)$
4	$\text{Beta}(0.55, 0.01)$
5	$\text{Beta}(0.6, 0.01)$
6	$\text{Beta}(0.65, 0.01)$
7	$\text{Beta}(0.7, 0.01)$
8	$\text{Beta}(0.75, 0.01)$
9	$\text{Beta}(0.8, 0.01)$
10	$\text{Beta}(0.85, 0.01)$

Table B.4: Costs in the third synthetic environment, for some parameter $\delta \in (0, 1/2)$.

Arm	Reward at rounds $i \equiv k \mod 4$ for...				Average cost
	$k = 1$	$k = 2$	$k = 3$	$k = 4$	
1	$1 - \delta$	$1 - \delta$	0	0	$1/2 - \delta/2$
2	$1/2 - \delta$	$1/2 - \delta$	1	1	$3/4 - \delta/2$
3	0	1	0	1	$1/2$
4	1	0	1	0	$1/2$

Table B.5: Means and standard deviations over 50 trials of performances (1-cost) for various combinations of **FPML** and online greedy algorithms in the third synthetic environment, with $\delta = 0.01$.

Algorithm	Mean	StD
Best-in-hindsight	1.000	0
Top-of-leaderboard	1.000	0
FPML	0.964	0.0145
$\mathbf{OG_{hybrid}}$ $((B_1, B_2) = 1, 3))$	0.823	0.0200
OG	0.799	0.0202

Geometric resampling

The *geometric resampling* technique used in the second and third partial feedback versions of **FPML** in the experiments is adapted from Neu and Bartók (2013). At each round cost estimates

$$\hat{c}_t(a) := \begin{cases} \frac{c_t(a)}{\hat{q}_{t,a}} & \text{if } a \text{ was pulled,} \\ 0 & \text{otherwise} \end{cases}$$

are made, where $\hat{q}_{t,a}$ is an estimate of the probability $q_{t,a} := \mathbb{P}(\text{arm } a \text{ pulled at round } t)$. These estimates are made by sampling $\frac{1}{\hat{q}_{t,a}} \sim \text{Geom}(q_{t,a})$, which is done by repeating the algorithm's execution at this round and counting how many trials are needed until a is pulled again. In practice, the number of repetitions must be capped and this introduces some bias to the estimates, but this is not problematic in practice. In fact, there is a bias variance trade-off, because $K = \max_{a \in \mathcal{A}} |\hat{c}_t(a)|$ is bounded by the number of samples we take. Therefore more samples lead to lower bias but higher variance. Using bounds similar to those of Proposition 4.2 as a guide, we picked the number of samples to be $\left(N \left(\frac{TN}{\ln(N)} \right)^{\tilde{B}} \right)^{1/(2\tilde{B}+1)}$, so $K = \left(N \left(\frac{TN}{\ln(N)} \right)^{\tilde{B}} \right)^{1/(2\tilde{B}+1)}$ and $\varepsilon = \left(\frac{\ln(N)}{T} \left(\frac{\ln(N)}{TN} \right)^{\tilde{B}} \right)^{1/(2\tilde{B}+1)}$, where $\tilde{B}$ is the budget of each **FPML-partial** box.

These estimators make complete use of the information received at each round, unlike the simple one-arm uniform sampling, B arms exploiting version of **FPML** with partial feedback mentioned in Section 4.2.1.

Methods

Reward definitions: For the black-box optimization experiments in Section 4.4, the *reward* (1−cost) for each black-box optimizer on each machine learning task (i.e. round) was defined as follows. This approach was inspired heavily by the Bayesmark package used in the 2020 NeurIPS BBO Challenge and which we based our implementation on (Uber, 2020).

Fix a round t and an optimizer a. Let $\texttt{opt}_t$ be an estimate of the global minimum classification/regression loss achievable (at validation, not test) on the task corresponding to round t. Define $\overline{\texttt{rand}}_t$ to be the mean performance of a random hyperparameter search on this task (i.e. the smallest loss achieved using any hyperparameter in the random search, averaged over trials).[1] Finally, let $\overline{\texttt{loss}}_t(a)$ be the actual averaged minimum loss of the optimizer a on this problem.

The reward is then defined as

$$r_i(a) := 1 - \frac{\overline{\texttt{loss}}_t(a) - \texttt{opt}_t}{\overline{\texttt{rand}}_t(a) - \texttt{opt}_t}.$$

Conceptually, the reward is 0 when optimizer a performs as badly as a random search, and 1 when it performs as well as is possible on this task.

As per usual, the reward for a bandit algorithm selecting multiple optimizers at each round is then calculated as the maximum of the rewards of each optimizer (equivalent to the minimum of costs).

Bayesian optimizers used: The nine black-box optimization algorithms we ran the experiments in Section 4.4 over were as follows:

1. Hyperopt (Bergstra et al., 2015)
2. The `AUCBanditMetaTechniqueA` technique from OpenTuner (Ansel et al., 2014)
3. The `PSO_GA_Bandit` technique from OpenTuner (Ansel et al., 2014)
4. The `PSO_GA_DE` technique from OpenTuner (Ansel et al., 2014)
5. PySOT (Eriksson et al., 2019)
6. Scikit-Optimize (Head et al., 2018) using base estimator `GBRT` and acquisition objective `gp_hedge`

[1] In reality this is estimated using a more statistically efficient technique than actually performing the random search, as in the Bayesmark package.

7. Scikit-Optimize (Head et al., 2018) using base estimator `GP` and acquisition objective `gp_hedge`

8. Scikit-Optimize (Head et al., 2018) using base estimator `GP` and acquisition objective `LCB`

9. Random search

The default settings of each package were used.

Appendix C

Appendix For Chapter 5

C.1 Section 5.1

Theorem C.1. *(Azuma's inequality, stronger form.) Let* $\{X_i\}_{i=0}^{\infty}$ *be a super-martingale and* $\forall i \in \mathbb{N}$, $|X_i - X_{i-1}| \leq c_i$ *almost surely. Then for all* $N \in \mathbb{N}$ *and* $\varepsilon > 0$,

$$\mathbb{P}\left(\max_{i\in[N]} X_i - X_0 \geq \varepsilon\right) \leq \exp\left(\frac{-\varepsilon^2}{2\sum_{i=1}^{N} c_i^2}\right)$$

Discussion of bounding the deviation on subsets chosen by an algorithm: We motivated why we expect the deviation between the loss estimates and their expectations to be small on any fixed subset, but we did not explain why we also expect this to hold when the subset is chosen dynamically based on the algorithm's choices. This requires being more formal about the stochastic process. Here we formulate a precise statement of this claim. Here we imagine that c_r is 1 if the algorithm picks the expert in round r, and 0 otherwise. At the end of round $r-1$, we imagine that all experts generate their estimates b_r as a function of the current available information $(\mathcal{F}_{r-1})$. The algorithm also chooses c_r as a function of the current available information, and then the new loss l_r (contained in $\mathcal{F}_r$) is revealed in the new round.

Conjecture C.1. *Let the processes* $\{l_r\}_{r=1}^{\infty}$ *be adapted with respect to the filtration*

$\mathcal{F} = \{\mathcal{F}_0, \mathcal{F}_1, \dots\}$, *and* $\forall r$ *let* $l_r \in [0,1]$ *w.p. 1. Let the process* $\{c_r\}_{r=1}^{\infty}$ *be predictable with respect to* $\mathcal{F}$ *(i.e.* c_r *is* $\mathcal{F}_{r-1}$ *measurable), and let* $b_r = \mathbb{E}[l_r|\mathcal{F}_{r-1}]$. *Then*

$$\mathbb{P}\left[\exists R' \in \{1,\dots,R\} \text{ s.t. } \sum_{r=1}^{R'} c_r(l_r - b_r) > k\sqrt{\sum_{r=1}^{R'} c_r}\right] \leq \exp\left(-k^2/2\right)$$

We believe this conjecture to be true, but there are technical difficulties in the proof. Note that $M_r = \{c_r(l_r - b_r)\}$ is a martingale by construction. One can use a maximal form of the Azuma-Hoeffding inequality (AH) [1] to get a high probability bound which holds for every $R' \leq R$ simultaneously. More precisely, it's straightforward to show that $\{\exists R' \in \{1,\dots,R\} \text{ s.t. } \sum_{r=1}^{R'} c_r(l_r - b_r) > k\sqrt{R}\}$ occurs with probability less than $\exp(-k^2/2)$. Ideally, we would now like to argue that the right hand side of the inequality should really be $k\sqrt{\sum_{r=1}^{R} c_r}$, because rounds where $c_r = 0$ can be ignored. Unfortunately we were unable to find a way to do this formally, however we do believe the conjecture to be true. While idealized results are useful for sanity checking the reasonableness of a choice of γ, in practice the choice of γ may be dominated by empirical considerations, e.g. how well the loss estimates $b_{e,r}$ upper bound $l_{e,r}$ on average, or how quickly loss estimates adapt to distribution shift.

C.2 Section 5.2

Definition C.1.

1. *Let* Σ *be a finite ordered alphabet of symbols, and* $(\Sigma)^*$ *the set of finite strings of* Σ *(including the empty string). If* $x = x_1 \dots x_k$ *is a string in* $(\Sigma)^*$ *with* $x_i \in \Sigma$, *then we associate* x *with the number* 0 *if* x *is the empty string, and otherwise* $\sum_{i=1}^{k} ord(x_i)|\Sigma|^{i-1} + 1$, *where* $ord(x_i) \in \{0,\dots,|\Sigma|-1\}$ *is the order of* x_i *in* Σ. *This association defines a bijection between* $(\Sigma)^*$ *and* $\mathbb{N}_0$.

[1] see e.g. https://people.math.wisc.edu/ roch/grad-prob/gradprob-notes20.pdf

2. *Given $x \in \{0,1\}^*$, we associate x with the number represented as $1x$ in binary, minus one. This defines a bijection between $\{0,1\}^*$ and $\mathbb{N}_0$.*

3. *Given $x \in \{0,1\}^*$, let $l(x)$ be the length of x. We define $\overline{x} = 1^{l(x)}0x$ (i.e. $\overline{x}$ is prefixed by $l(x)$ ones) and $x' = \overline{l(x)}x = 1^{l(l(x))}0l(x)x$. The mapping $x \to x'$ maps binary strings to a prefix free code.*

4. *Given $x \in (\Sigma)^*$, we associate x with an element in $\{0,1\}^*$ by mapping x to its corresponding element in $\mathbb{N}_0$ using (1), mapping then to its corresponding element in $\mathbb{B}^*$ using (2), and finally mapping this to its corresponding element in the prefix free code using (3).*

Bibliography

Karim Ahmed, Mohammad Haris Baig, and Lorenzo Torresani. Network of experts for large-scale image categorization. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VII*, volume 9911 of *Lecture Notes in Computer Science*, pages 516–532. Springer, 2016. doi: 10.1007/978-3-319-46478-7_32. URL https://doi.org/10.1007/978-3-319-46478-7_32. 122

Rahaf Aljundi, Punarjay Chakravarty, and Tinne Tuytelaars. Expert gate: Lifelong learning with a network of experts. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 7120–7129. IEEE Computer Society, 2017. doi: 10.1109/CVPR.2017.753. URL https://doi.org/10.1109/CVPR.2017.753. 34, 122, 123

Marcin Andrychowicz, Misha Denil, Sergio Gomez Colmenarejo, Matthew W. Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 3981–3989, 2016. URL http://papers.nips.cc/paper/6461-learning-to-learn-by-gradient-descent-by-gradient-descent. 38

Jason Ansel, Shoaib Kamil, Kalyan Veeramachaneni, Jonathan Ragan-Kelley, Jeffrey Bosboom, Una-May O'Reilly, and Saman Amarasinghe. Opentuner: An extensible framework for program autotuning. In *Proceedings of the 23rd international conference on Parallel architectures and compilation*, pages 303–316, 2014. 214

Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009. ISBN 978-0-521-42426-4. URL http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521424264. 2

Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory Comput.*, 8(1):121–164, 2012. doi: 10.4086/toc.2012.v008a006. URL https://doi.org/10.4086/toc.2012.v008a006. 9, 83, 85, 110, 111, 112, 195

Jean-Yves Audibert, Sébastien Bubeck, and Gábor Lugosi. Regret in online combinatorial optimization. *Math. Oper. Res.*, 39(1):31–45, 2014. doi: 10.1287/moor.2013.0598. URL https://doi.org/10.1287/moor.2013.0598. 84

Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E Schapire. The nonstochastic multiarmed bandit problem. *SIAM journal on computing*, 32(1):48–77, 2002. 106

Robert J. Aumann. Survey of repeated games. In *Essays in Game Theory and Mathematical Economics in Honor of Oskar Morgenstern*, 1981. 3, 13

Baruch Awerbuch and Robert Kleinberg. Online linear optimization and adaptive routing. *Journal of Computer and System Sciences*, 74(1):97–114, 2008. 80

Samy Badreddine, Artur S. d'Avila Garcez, Luciano Serafini, and Michael Spranger. Logic tensor networks. *Artif. Intell.*, 303:103649, 2022. doi: 10.1016/j.artint.2021.103649. URL https://doi.org/10.1016/j.artint.2021.103649. 45

Maria-Florina Balcan, Vaishnavh Nagarajan, Ellen Vitercik, and Colin White. Learning-theoretic foundations of algorithm configuration for combinatorial partitioning problems. In Satyen Kale and Ohad Shamir, editors, *Proceedings of the 30th Conference on Learning Theory, COLT 2017, Amsterdam, The Netherlands, 7-10 July 2017*, volume 65 of *Proceedings of Machine Learning Research*, pages 213–274. PMLR, 2017. URL http://proceedings.mlr.press/v65/balcan17a.html. 5, 19

Maria-Florina Balcan, Travis Dick, Tuomas Sandholm, and Ellen Vitercik. Learning to branch. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 353–362. PMLR, 2018. URL http://proceedings.mlr.press/v80/balcan18a.html. 5, 19, 50

Maria-Florina Balcan, Tuomas Sandholm, and Ellen Vitercik. Learning to optimize computational resources: Frugal training with generalization guarantees. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 3227–3234. AAAI Press, 2020. URL https://aaai.org/ojs/index.php/AAAI/article/view/5721. 5, 22, 79

D.F. Beal. An analysis of minimax. *Advances in Computer Chess*, 2:103–109, 1980. 4, 18

E. Ben-Porath and B. Peleg. On the folk theorem and finite automata. In *The Hebrew University discussion paper.*, 1987. 3, 4, 14

Eli Ben-Sasson, Adam Tauman Kalai, and Ehud Kalai. An approach to bounded rationality. *NIPS*, pages 145–152, 2006. 4, 8, 18, 50

Abhijit Bendale and Terrance E. Boult. Towards open world recognition. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 1893–1902. IEEE Computer Society, 2015. doi: 10.1109/CVPR.2015.7298799. URL https://doi.org/10.1109/CVPR.2015.7298799. 41, 179

Abhijit Bendale and Terrance E. Boult. Towards open set deep networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 1563–1572. IEEE Computer Society, 2016. doi: 10.1109/CVPR.2016.173. URL https://doi.org/10.1109/CVPR.2016.173. 41

James O. Berger. *Statistical Decision Theory and Bayesian Analysis, Second Edition*. Springer, 1985. ISBN 3-540-96098-8. doi: 10.1007/978-1-4757-4286-2. URL https://doi.org/10.1007/978-1-4757-4286-2. 1

James Bergstra, Brent Komer, Chris Eliasmith, Dan Yamins, and David D Cox. Hyperopt: a python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, 8(1):014008, 2015. 214

Aditya Bhaskara, Sreenivas Gollapudi, Sungjin Im, Kostas Kollias, and Kamesh Munagala. Online learning and bandits with queried hints. In Yael Tauman Kalai, editor, *14th Innovations in Theoretical Computer Science Conference, ITCS 2023, January 10-13, 2023, MIT, Cambridge, Massachusetts, USA*, volume 251 of *LIPIcs*, pages 16:1–16:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi: 10.4230/LIPIcs.ITCS.2023.16. URL https://doi.org/10.4230/LIPIcs.ITCS.2023.16. 85, 93

David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, 2003. URL http://jmlr.org/papers/v3/blei03a.html. 40

Haitham Bou-Ammar, Eric Eaton, Paul Ruvolo, and Matthew E. Taylor. Online multi-task learning for policy gradient methods. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, volume 32 of *JMLR Workshop and Conference Proceedings*, pages 1206–1214. JMLR.org, 2014. URL http://proceedings.mlr.press/v32/ammar14.html. 43

Haitham Bou-Ammar, Eric Eaton, Paul Ruvolo, and Matthew E. Taylor. Unsupervised cross-domain transfer in policy gradient reinforcement learning via manifold alignment. In Blai Bonet and Sven Koenig, editors, *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, pages 2504–2510. AAAI Press, 2015. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9916. 44

Xavier Bouthillier and Gaël Varoquaux. *Survey of machine-learning experimental methods at NeurIPS2019 and ICLR2020*. PhD thesis, Inria Saclay Ile de France, 2020. 107

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Confer-*

ence on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html. 7

Emma Brunskill and Lihong Li. Pac-inspired option discovery in lifelong reinforcement learning. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, volume 32 of *JMLR Workshop and Conference Proceedings*, pages 316–324. JMLR.org, 2014. URL http://proceedings.mlr.press/v32/brunskill14.html. 11, 44, 173

Robert R Bush and Frederick Mosteller. Stochastic models for learning. 1955. 46

Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka Jr., and Tom M. Mitchell. Toward an architecture for never-ending language learning. 2010. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1879. 41

Nicolò Cesa-Bianchi and Gábor Lugosi. *Prediction, learning, and games*. Cambridge University Press, 2006. ISBN 978-0-521-84108-5. doi: 10.1017/CBO9780511546921. URL https://doi.org/10.1017/CBO9780511546921. 9, 21

Nicolò Cesa-Bianchi and Gábor Lugosi. Combinatorial bandits. *J. Comput. Syst. Sci.*, 78(5):1404–1422, 2012. doi: 10.1016/j.jcss.2012.01.001. URL https://doi.org/10.1016/j.jcss.2012.01.001. 84

Wei Chen, Wei Hu, Fu Li, Jian Li, Yu Liu, and Pinyan Lu. Combinatorial multi-armed bandit with general reward functions. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages

1651–1659, 2016a. URL https://proceedings.neurips.cc/paper/2016/hash/aa169b49b583a2b5af89203c2b78c67c-Abstract.html. 80, 83, 84

Wei Chen, Yajun Wang, Yang Yuan, and Qinshi Wang. Combinatorial multi-armed bandit and its extension to probabilistically triggered arms. *J. Mach. Learn. Res.*, 17:50:1–50:33, 2016b. URL http://jmlr.org/papers/v17/14-298.html. 84

Zhiyuan Chen and Bing Liu. Topic modeling using topics from many domains, lifelong learning and big data. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, volume 32 of *JMLR Workshop and Conference Proceedings*, pages 703–711. JMLR.org, 2014. URL http://proceedings.mlr.press/v32/chenf14.html. 40, 41

Zhiyuan Chen and Bing Liu. *Lifelong Machine Learning, Second Edition*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2018. doi: 10.2200/S00832ED1V01Y201802AIM037. URL https://doi.org/10.2200/S00832ED1V01Y201802AIM037. ii, 5, 10, 23, 115, 136

Zhiyuan Chen, Nianzu Ma, and Bing Liu. Lifelong learning for sentiment classification. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 2: Short Papers*, pages 750–756. The Association for Computer Linguistics, 2015. doi: 10.3115/v1/p15-2123. URL https://doi.org/10.3115/v1/p15-2123. 24

Richard Combes, Mohammad Sadegh Talebi, Alexandre Proutière, and Marc Lelarge. Combinatorial bandits revisited. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Pro-*

cessing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada, pages 2116–2124, 2015. URL https://proceedings.neurips.cc/paper/2015/hash/0ce2ffd21fc958d9ef0ee9ba5336e357-Abstract.html. 84

Vincent Conitzer. Metareasoning as a formal computational problem. In Michael T. Cox and Anita Raja, editors, *Metareasoning - Thinking about Thinking*, pages 119–128. MIT Press, 2011. doi: 10.7551/mitpress/9780262014809.003.0008. URL https://doi.org/10.7551/mitpress/9780262014809.003.0008. 20

Vincent Conitzer and Tuomas Sandholm. Definition and complexity of some basic metareasoning problems. In Georg Gottlob and Toby Walsh, editors, *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pages 1099–1106. Morgan Kaufmann, 2003. URL http://ijcai.org/Proceedings/03/Papers/158.pdf. 5, 20

Travis Dick, Wesley Pegden, and Maria-Florina Balcan. Semi-bandit optimization in the dispersed setting. In Ryan P. Adams and Vibhav Gogate, editors, *Proceedings of the Thirty-Sixth Conference on Uncertainty in Artificial Intelligence, UAI 2020, virtual online, August 3-6, 2020*, page 374. AUAI Press, 2020. URL http://www.auai.org/uai2020/proceedings/374_main_paper.pdf. 5, 9, 22, 50

James Dow. Search decisions with limited memory. *The Review of Economic Studies*, 58(1):1, January 1991. doi: 10.2307/2298042. URL https://doi.org/10.2307/2298042. 4, 17, 50

Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl$ˆ2$: Fast reinforcement learning via slow reinforcement learning. *CoRR*, abs/1611.02779, 2016. URL http://arxiv.org/abs/1611.02779. 37

Kevin Ellis, Lucas Morales, Mathias Sablé-Meyer, Armando Solar-Lezama, and Josh Tenenbaum. Learning libraries of subroutines for neurally-guided bayesian pro-

gram induction. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 7816–7826, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/7aa685b3b1dc1d6780bf36f7340078c9-Abstract.html. 25, 116, 138

Kevin Ellis, Catherine Wong, Maxwell I. Nye, Mathias Sablé-Meyer, Luc Cary, Lucas Morales, Luke B. Hewitt, Armando Solar-Lezama, and Joshua B. Tenenbaum. Dreamcoder: Growing generalizable, interpretable knowledge with wake-sleep bayesian program learning. *CoRR*, abs/2006.08381, 2020. URL https://arxiv.org/abs/2006.08381. 10, 25, 116, 138, 139

David Eriksson, David Bindel, and Christine A Shoemaker. pysot and poap: An event-driven asynchronous framework for surrogate optimization. *arXiv preprint arXiv:1908.00420*, 2019. 214

Absalom E. Ezugwu, Abiodun M. Ikotun, Olaide Nathaniel Oyelade, Laith Mohammad Abualigah, Jeffrey O. Agushaka, Christopher I. Eke, and Andronicus Ayobami Akinyelu. A comprehensive survey of clustering algorithms: State-of-the-art machine learning applications, taxonomy, challenges, and future research prospects. *Eng. Appl. Artif. Intell.*, 110:104743, 2022. doi: 10.1016/j.engappai.2022.104743. URL https://doi.org/10.1016/j.engappai.2022.104743. 146

Geli Fei, Shuai Wang, and Bing Liu. Learning cumulatively to become more knowledgeable. In Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi, editors, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1565–1574. ACM, 2016. doi:

10.1145/2939672.2939835. URL https://doi.org/10.1145/2939672.2939835. 41, 42

Fernando Fernández and Manuela M. Veloso. Learning domain structure through probabilistic policy reuse in reinforcement learning. *Prog. Artif. Intell.*, 2(1):13–27, 2013. doi: 10.1007/s13748-012-0026-6. URL https://doi.org/10.1007/s13748-012-0026-6. 44

Diogo R. Ferreira. The impact of search depth on chess playing strength. *ICGA journal*, 32(2):67–80, 2013. doi: 10.3233/ICG-2013-36202. 18

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135. PMLR, 2017. URL http://proceedings.mlr.press/v70/finn17a.html. 38, 149

Lance Fortnow. Program equilibria and discounted computation time. In Aviad Heifetz, editor, *Proceedings of the 12th Conference on Theoretical Aspects of Rationality and Knowledge (TARK-2009), Stanford, CA, USA, July 6-8, 2009*, pages 128–133, 2009. doi: 10.1145/1562814.1562833. URL https://doi.org/10.1145/1562814.1562833. 16

Lance Fortnow and Rahul Santhanam. Bounding rationality by discounting time. *ICS*, pages 143–155, 2010. 4, 16

Lance Fortnow and Duke Whang. Optimality and domination in repeated games with bounded players. In Frank Thomson Leighton and Michael T. Goodrich, editors, *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing,*

23-25 May 1994, Montréal, Québec, Canada, pages 741–749. ACM, 1994. doi: 10.1145/195058.195448. URL https://doi.org/10.1145/195058.195448. 4, 15

Arthur Franz, Oleksandr Antonenko, and Roman Soletskyi. A theory of incremental compression. *Inf. Sci.*, 547:28–48, 2021. doi: 10.1016/j.ins.2020.08.035. URL https://doi.org/10.1016/j.ins.2020.08.035. 139

Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In Lorenza Saitta, editor, *Machine Learning, Proceedings of the Thirteenth International Conference (ICML '96), Bari, Italy, July 3-6, 1996*, pages 148–156. Morgan Kaufmann, 1996. 195

Yoav Freund and Robert E Schapire. Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, 29(1-2):79–103, 1999. 113

Drew Fudenberg and David K Levine. *The theory of learning in games*, volume 2. MIT press, 1998. 45

Artur d'Avila Garcez and Luis C Lamb. Neurosymbolic ai: The 3rd wave. *Artificial Intelligence Review*, pages 1–20, 2023. 45

Manuel Garcia-Piqueras and José Hernández-Orallo. Conditional teaching size. *CoRR*, abs/2107.07038, 2021. URL https://arxiv.org/abs/2107.07038. 139

Sally A. Goldman and Michael J. Kearns. On the complexity of teaching. In Manfred K. Warmuth and Leslie G. Valiant, editors, *Proceedings of the Fourth Annual Workshop on Computational Learning Theory, COLT 1991, Santa Cruz, California, USA, August 5-7, 1991*, pages 303–314. Morgan Kaufmann, 1991. URL http://dl.acm.org/citation.cfm?id=114865. 46

Carla P. Gomes and Bart Selman. Algorithm portfolios. *Artif. Intell.*, 126(1-2):43–

62, 2001. doi: 10.1016/S0004-3702(00)00081-3. URL https://doi.org/10.1016/S0004-3702(00)00081-3. 80

Ian J. Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks, 2015. 123

Aditya Gopalan, Shie Mannor, and Yishay Mansour. Thompson sampling for complex online problems. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, volume 32 of *JMLR Workshop and Conference Proceedings*, pages 100–108. JMLR.org, 2014. URL http://proceedings.mlr.press/v32/gopalan14.html. 84

Peter Grünwald and Teemu Roos. Minimum description length revisited. *CoRR*, abs/1908.08484, 2019. URL http://arxiv.org/abs/1908.08484. 141

Leo Gugerty. Newell and simon's logic theorist: Historical background and impact on cognitive modeling. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 50(9):880–884, October 2006. doi: 10.1177/154193120605000904. URL https://doi.org/10.1177/154193120605000904. 12

Rishi Gupta and Tim Roughgarden. A PAC approach to application-specific algorithm selection. In Madhu Sudan, editor, *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*, pages 123–134. ACM, 2016. doi: 10.1145/2840728.2840766. URL https://doi.org/10.1145/2840728.2840766. 20

David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Process-*

ing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada, pages 2455–2467, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/2de5d16682c3c35007e4e92982f1a2ba-Abstract.html. 179

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy P. Lillicrap. Mastering diverse domains through world models. *CoRR*, abs/2301.04104, 2023. doi: 10.48550/arXiv.2301.04104. URL https://doi.org/10.48550/arXiv.2301.04104. 11, 117, 178, 188

Joseph Y. Halpern and Rafael Pass. Algorithmic rationality: Game theory with costly computation. *Journal of Economic Theory*, 156:246–268, 2015. 4, 16, 17

Joseph Y. Halpern, Rafael Pass, and Lior Seeman. Computational extensive-form games. *EC*, pages 681–698, 2016. 17

Joseph Y. Halpern, Rafael Pass, and Daniel Reichman. On the existence of nash equilibrium in games with resource-bounded players. In Dimitris Fotakis and Evangelos Markakis, editors, *Algorithmic Game Theory - 12th International Symposium, SAGT 2019, Athens, Greece, September 30 - October 3, 2019, Proceedings*, volume 11801 of *Lecture Notes in Computer Science*, pages 139–152. Springer, 2019. doi: 10.1007/978-3-030-30473-7_10. URL https://doi.org/10.1007/978-3-030-30473-7_10. 4, 17

Yanjun Han, Yining Wang, and Xi Chen. Adversarial combinatorial bandits with general non-linear reward functions. In *International Conference on Machine Learning*, pages 4030–4039. PMLR, 2021. 84

Elad Hazan and Tomer Koren. The computational power of optimization in online learning. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 128–141. ACM, 2016. doi:

10.1145/2897518.2897536. URL https://doi.org/10.1145/2897518.2897536. 132

Tim Head, MechCoder, Gilles Louppe, Iaroslav Shcherbatyi, fcharras, Zé Vinícius, cmmalone, Christopher Schröder, nel215, Nuno Campos, Todd Young, Stefano Cereda, Thomas Fan, rene rex, Kejia (KJ) Shi, Justus Schwabedal, carlosdanielcsantos, Hvass-Labs, Mikhail Pak, SoManyUsernamesTaken, Fred Callaway, Loïc Estève, Lilian Besson, Mehdi Cherti, Karlson Pfannschmidt, Fabian Linzberger, Christophe Cauet, Anna Gut, Andreas Mueller, and Alexander Fabisch. scikit-optimize/scikit-optimize: v0.5.2, March 2018. URL https://doi.org/10.5281/zenodo.1207017. 214, 215

Marcus Hutter. *Universal Artificial Intellegence*. Springer Berlin Heidelberg, 2005. doi: 10.1007/b138233. URL https://doi.org/10.1007/b138233. 1, 144, 175

David Isele, Mohammad Rostami, and Eric Eaton. Using task features for zero-shot knowledge transfer in lifelong learning. In Subbarao Kambhampati, editor, *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 1620–1626. IJCAI/AAAI Press, 2016. URL http://www.ijcai.org/Abstract/16/232. 44

Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Comput.*, 3(1):79–87, 1991. doi: 10.1162/neco.1991.3.1.79. URL https://doi.org/10.1162/neco.1991.3.1.79. 122

K. J. Joseph, Salman H. Khan, Fahad Shahbaz Khan, and Vineeth N. Balasubramanian. Towards open world object detection. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pages 5830–5840. Computer Vision Foundation / IEEE, 2021. doi: 10.1109/CVPR46437.

2021.00577. URL https://openaccess.thecvf.com/content/CVPR2021/html/Joseph_Towards_Open_World_Object_Detection_CVPR_2021_paper.html. 117

Daniel Kahneman. Maps of bounded rationality: Psychology for behavioral economics. *American economic review*, 93(5):1449–1475, 2003. 12

Lukasz Kaiser, Ofir Nachum, Aurko Roy, and Samy Bengio. Learning to remember rare events. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=SJTQLdqlg. 37

Adam Tauman Kalai and Santosh S. Vempala. Efficient algorithms for online decision problems. *J. Comput. Syst. Sci.*, 71(3):291–307, 2005. doi: 10.1016/j.jcss.2004.10.016. URL https://doi.org/10.1016/j.jcss.2004.10.016. 9, 83, 87, 88, 89, 91, 99

Ehud Kalai and William Stanford. Finite rationality and interpersonal complexity in repeated games. *Econometrica*, 56(2):397, March 1988. doi: 10.2307/1911078. URL https://doi.org/10.2307/1911078. 4, 14

Varun Kanade, H. Brendan McMahan, and Brent Bryan. Sleeping experts and bandits with stochastic action availability and adversarial rewards. In David A. Van Dyk and Max Welling, editors, *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics, AISTATS 2009, Clearwater Beach, Florida, USA, April 16-18, 2009*, volume 5 of *JMLR Proceedings*, pages 272–279. JMLR.org, 2009. URL http://proceedings.mlr.press/v5/kanade09a.html. 121

Michael J. Kearns and Umesh Vazirani. *An Introduction to Computational Learning Theory*. The MIT Press, 1994. doi: 10.7551/mitpress/3897.001.0001. URL https://doi.org/10.7551/mitpress/3897.001.0001. 1, 20, 28, 115

Thomas N. Kipf, Elise van der Pol, and Max Welling. Contrastive learning of structured world models. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=H1gax6VtDB. 11, 117, 188, 190

James Kirkpatrick, Razvan Pascanu, Neil C. Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *CoRR*, abs/1612.00796, 2016. URL http://arxiv.org/abs/1612.00796. 6, 31

Robert Kleinberg, Kevin Leyton-Brown, and Brendan Lucier. Efficiency through procrastination: Approximately optimal algorithm configuration with runtime guarantees. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 2023–2031. ijcai.org, 2017. doi: 10.24963/ijcai.2017/281. URL https://doi.org/10.24963/ijcai.2017/281. 5, 21, 79

Robert Kleinberg, Kevin Leyton-Brown, Brendan Lucier, and Devon R. Graham. Procrastinating with confidence: Near-optimal, anytime, adaptive algorithm configuration. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 8881–8891, 2019. 21, 79

Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City,*

UT, USA, January 5-8, 2020, pages 1859–1877. SIAM, 2020. doi: 10.1137/1.9781611975994.114. URL https://doi.org/10.1137/1.9781611975994.114. 122

Tor Lattimore and Csaba Szepesvári. *Bandit Algorithms*. Cambridge University Press, July 2020. doi: 10.1017/9781108571401. URL https://doi.org/10.1017/9781108571401. 9, 21, 81, 93, 121

Erwan Lecarpentier, David Abel, Kavosh Asadi, Yuu Jinnai, Emmanuel Rachelson, and Michael L. Littman. Lipschitz lifelong reinforcement learning. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 8270–8278. AAAI Press, 2021. URL https://ojs.aaai.org/index.php/AAAI/article/view/17006. 11, 44, 173

Jian Li and Amol Deshpande. Maximizing expected utility for stochastic combinatorial optimization problems. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 797–806. IEEE Computer Society, 2011. doi: 10.1109/FOCS.2011.33. URL https://doi.org/10.1109/FOCS.2011.33. 84

Ming Li and Paul Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer New York, 2008. doi: 10.1007/978-0-387-49820-1. URL https://doi.org/10.1007/978-0-387-49820-1. 138

Zhizhong Li and Derek Hoiem. Learning without forgetting. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*, volume 9908 of *Lecture Notes in Computer Science*, pages

614–629. Springer, 2016. doi: 10.1007/978-3-319-46493-0_37. URL https://doi.org/10.1007/978-3-319-46493-0_37. 33

Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. *Inf. Comput.*, 108(2):212–261, 1994. doi: 10.1006/inco.1994.1009. URL https://doi.org/10.1006/inco.1994.1009. 9, 83

Jiwei Liu, Bojan Tunguz, and Gilberto Titericz. Gpu accelerated exhaustive search for optimal ensemble of black-box optimization algorithms. *arXiv preprint arXiv:2012.04201*, 2020a. 107

Xinming Liu and Joseph Y. Halpern. Strategic play by resource-bounded agents in security games. In Noa Agmon, Bo An, Alessandro Ricci, and William Yeoh, editors, *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2023, London, United Kingdom, 29 May 2023 - 2 June 2023*, pages 2478–2480. ACM, 2023. doi: 10.5555/3545946.3598973. URL https://dl.acm.org/doi/10.5555/3545946.3598973. 14

Zhengying Liu, Zhen Xu, Shangeth Rajaa, Meysam Madadi, Julio CS Jacques Junior, Sergio Escalera, Adrien Pavao, Sebastien Treguer, Wei-Wei Tu, and Isabelle Guyon. Towards automated deep learning: Analysis of the autodl challenge series 2019. In *NeurIPS 2019 Competition and Demonstration Track*, pages 242–252. PMLR, 2020b. 107

Mitja Luštrek, Matjaž Gamsa, and Ivan Bratkob. Is real-valued minimax pathological? *Artificial Intelligence*, 120(6-7):620–642, 2006. 18

Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. *J. ACM*, 68(4):24:1–24:25, 2021. doi: 10.1145/3447579. URL https://doi.org/10.1145/3447579. 122

Dougal Maclaurin, David Duvenaud, and Ryan P. Adams. Gradient-based hyperparameter optimization through reversible learning. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 2113–2122. JMLR.org, 2015. URL http://proceedings.mlr.press/v37/maclaurin15.html. 38

Horia Mania, Aurelia Guy, and Benjamin Recht. Simple random search provides a competitive approach to reinforcement learning. *CoRR*, abs/1803.07055, 2018. URL http://arxiv.org/abs/1803.07055. 40

Michael Maschler, Eilon Solan, and Shmuel Zamir. *Game Theory*. Cambridge University Press, Cambridge, England, 2 edition, June 2020. 51, 54

Sahisnu Mazumder, Nianzu Ma, and Bing Liu. Towards a continuous knowledge learning engine for chatbots. *CoRR*, abs/1802.06024, 2018. URL http://arxiv.org/abs/1802.06024. 41

Michael McCloskey and Neal J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of Learning and Motivation*, 24:109–165, 1989. doi: 10.1016/s0079-7421(08)60536-8. 6, 31

Nimrod Megiddo and Avi Wigderson. On play by means of computing machines. *TARK*, pages 259–274, 1986. 15

Jorge A. Mendez, Harm van Seijen, and Eric Eaton. Modular lifelong reinforcement learning via neural composition. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=5XmLzdslFNN. 11, 43, 173

Tom M. Mitchell, William W. Cohen, Estevam R. Hruschka Jr., Partha Pratim Talukdar, Justin Betteridge, Andrew Carlson, Bhavana Dalvi Mishra, Matthew Gardner, Bryan Kisiel, Jayant Krishnamurthy, Ni Lao, Kathryn Mazaitis, Thahir Mohamed, Ndapandula Nakashole, Emmanouil A. Platanios, Alan Ritter, Mehdi Samadi, Burr Settles, Richard C. Wang, Derry Wijaya, Abhinav Gupta, Xinlei Chen, Abulhair Saparov, Malcolm Greaves, and Joel Welling. Never-ending learning. pages 2302–2310, 2015. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/10049. 41

Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. *Commun. ACM*, 65(7):33–35, 2022. doi: 10.1145/3528087. URL https://doi.org/10.1145/3528087. 122

Thomas M. Moerland, Joost Broekens, Aske Plaat, and Catholijn M. Jonker. Model-based reinforcement learning: A survey. *Found. Trends Mach. Learn.*, 16(1):1–118, 2023. doi: 10.1561/2200000086. URL https://doi.org/10.1561/2200000086. 146

Tsendsuren Munkhdalai and Hong Yu. Meta networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 2554–2563. PMLR, 2017. URL http://proceedings.mlr.press/v70/munkhdalai17a.html. 36, 37

Pierre-Alexandre Murena, Antoine Cornuéjols, and Jean-Louis Dessalles. Incremental learning with the minimum description length principle. In *2017 International Joint Conference on Neural Networks, IJCNN 2017, Anchorage, AK, USA, May 14-19, 2017*, pages 1908–1915. IEEE, 2017. doi: 10.1109/IJCNN.2017.7966084. URL https://doi.org/10.1109/IJCNN.2017.7966084. 139

D.S. Nau. *Quality of decision versus depth of search on game trees*. PhD thesis, Duke University, 1979. 4, 18

Gergely Neu and Gábor Bartók. An efficient algorithm for learning with semi-bandit feedback. In *International Conference on Algorithmic Learning Theory*, pages 234–248. Springer, 2013. 95, 108, 213

Abraham Neyman. Bounded complexity justifies co-operation in the finitely repeated prisoner's dilemma. *Economics Letters*, 19(3):227–229, 1985. doi: https://doi.org/10.1016/0165-1765(85)90026-6. 3, 4, 13, 14

Caspar Oesterheld. Robust program equilibrium. *Theory and Decision*, 86(1):143–159, November 2018. doi: 10.1007/s11238-018-9679-3. URL https://doi.org/10.1007/s11238-018-9679-3. 4, 16

Randal S Olson, William La Cava, Patryk Orzechowski, Ryan J Urbanowicz, and Jason H Moore. Pmlb: a large benchmark suite for machine learning evaluation and comparison. *BioData mining*, 10(1):1–13, 2017. 107

Thomas Orton. Modeling precomputation in games played under computational constraints. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 2005–2011. ijcai.org, 2021. doi: 10.24963/ijcai.2021/276. URL https://doi.org/10.24963/ijcai.2021/276. v, 7

Thomas Orton and Damon Falck. Trading off resource budgets for improved regret bounds. In *NeurIPS*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/31a57804448363bcab777f818f75f5b4-Abstract-Conference.html. v, 8, 110

Christos H. Papadimitriou and Mihalis Yannakakis. On complexity as bounded rationality. *STOC*, pages 726–733, 1994. 3, 13, 14, 15

Judea Pearl. Asymptotic properties of minimax trees and game-searching procedures. *Artificial Intelligence*, 14(2):113–138, 1980. 4, 18, 50

Anastasia Pentina and Christoph H. Lampert. A pac-bayesian bound for lifelong learning. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, volume 32 of *JMLR Workshop and Conference Proceedings*, pages 991–999. JMLR.org, 2014. URL http://proceedings.mlr.press/v32/pentina14.html. 24, 25

Anastasia Pentina and Ruth Urner. Lifelong learning with weighted majority votes. *NIPS*, pages 3612–3620, 2016. 11, 25, 28

Ameya Prabhu, Philip H. S. Torr, and Puneet K. Dokania. Gdumb: A simple approach that questions our progress in continual learning. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part II*, volume 12347 of *Lecture Notes in Computer Science*, pages 524–540. Springer, 2020. doi: 10.1007/978-3-030-58536-5_31. URL https://doi.org/10.1007/978-3-030-58536-5_31. 136

Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ML predictions. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 9684–9693, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/73a427badebe0e32caa2e1fc7530b7f3-Abstract.html. 122

Rajat Raina, Alexis Battle, Honglak Lee, Benjamin Packer, and Andrew Y. Ng. Self-taught learning: transfer learning from unlabeled data. In Zoubin Ghahramani, editor, *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvallis, Oregon, USA, June 20-24, 2007*, volume 227 of *ACM International Conference Proceeding Series*, pages 759–766. ACM, 2007. doi: 10.1145/1273496.1273592. URL https://doi.org/10.1145/1273496.1273592. 36

Alexander Rakhlin and Karthik Sridharan. Online learning with predictable sequences. In Shai Shalev-Shwartz and Ingo Steinwart, editors, *COLT 2013 - The 26th Annual Conference on Learning Theory, June 12-14, 2013, Princeton University, NJ, USA*, volume 30 of *JMLR Workshop and Conference Proceedings*, pages 993–1019. JMLR.org, 2013. URL http://proceedings.mlr.press/v30/Rakhlin13.html. 122

Esteban Real, Chen Liang, David R. So, and Quoc V. Le. Automl-zero: Evolving machine learning algorithms from scratch. *CoRR*, abs/2003.03384, 2020. URL https://arxiv.org/abs/2003.03384. 39

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H. Lampert. icarl: Incremental classifier and representation learning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 5533–5542. IEEE Computer Society, 2017. doi: 10.1109/CVPR.2017.587. URL https://doi.org/10.1109/CVPR.2017.587. 6, 33

Mark B. Ring. Child: A first step towards continual learning. In Sebastian Thrun and Lorien Y. Pratt, editors, *Learning to Learn*, pages 261–292. Springer, 1998. doi: 10.1007/978-1-4615-5529-2_11. URL https://doi.org/10.1007/978-1-4615-5529-2_11. 10, 43, 173

Jorma Rissanen. Modeling by shortest data description. *Autom.*, 14(5):465–471, 1978. doi: 10.1016/0005-1098(78)90005-5. URL https://doi.org/10.1016/0005-1098(78)90005-5. 141

Julia Robinson. An iterative method of solving a game. *Annals of mathematics*, pages 296–301, 1951. 46

Ariel Rubinstein. Finite automata play the repeated prisoners' dilemma. *Journal of Economic Theory*, 39(1):83–96, 1986. doi: https://doi.org/10.1016/0022-0531(86)90021-9. 2, 14

Ariel Rubinstein. *Modeling Bounded Rationality*, volume 1 of *MIT Press Books*. The MIT Press, December 1997. ISBN ARRAY(0x52527bb0). URL https://ideas.repec.org/b/mtp/titles/0262681005.html. 4, 17

Stuart J. Russell and Devika Subramanian. Provably bounded-optimal agents. *J. Artif. Intell. Res.*, 2:575–609, 1995. doi: 10.1613/jair.133. URL https://doi.org/10.1613/jair.133. 20

Stuart J. Russell and Eric Wefald. *Do the right thing - studies in limited rationality*. MIT Press, 1991. ISBN 978-0-262-18144-0. 2, 5, 19

Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *CoRR*, abs/1606.04671, 2016. URL http://arxiv.org/abs/1606.04671. 31

Paul Ruvolo and Eric Eaton. Active task selection for lifelong machine learning. In Marie desJardins and Michael L. Littman, editors, *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA*. AAAI Press, 2013a. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI13/paper/view/6463. 24

Paul Ruvolo and Eric Eaton. Ella: An efficient lifelong learning algorithm. *ICML*, page 507–515, 2013b. 11, 24, 26, 28

Tim Salimans, Jonathan Ho, Xi Chen, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *CoRR*, abs/1703.03864, 2017. URL http://arxiv.org/abs/1703.03864. 39, 169

Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy P. Lillicrap. Meta-learning with memory-augmented neural networks. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1842–1850. JMLR.org, 2016. URL http://proceedings.mlr.press/v48/santoro16.html. 37

Robert E. Schapire. The strength of weak learnability. *Mach. Learn.*, 5:197–227, 1990. doi: 10.1007/BF00116037. URL https://doi.org/10.1007/BF00116037. 113

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 1312–1320. JMLR.org, 2015. URL http://proceedings.mlr.press/v37/schaul15.html. 185

Walter J. Scheirer, Anderson de Rezende Rocha, Archana Sapkota, and Terrance E. Boult. Toward open set recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35 (7):1757–1772, 2013. doi: 10.1109/TPAMI.2012.256. URL https://doi.org/10.1109/TPAMI.2012.256. 42

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy P. Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *Nat.*, 588(7839):604–609, 2020. doi: 10.1038/s41586-020-03051-4. URL https://doi.org/10.1038/s41586-020-03051-4. 179

John Schulman, Sergey Levine, Pieter Abbeel, Michael I. Jordan, and Philipp Moritz. Trust region policy optimization. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 1889–1897. JMLR.org, 2015. URL http://proceedings.mlr.press/v37/schulman15.html. 40

Burr Settles. Active learning literature survey. 2009. 46

Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 2990–2999, 2017. URL http://papers.nips.cc/paper/6892-continual-learning-with-deep-generative-replay. 6, 34

Lei Shu, Hu Xu, and Bing Liu. Lifelong learning CRF for supervised aspect extraction. In Regina Barzilay and Min-Yen Kan, editors, *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 2: Short Papers*, pages 148–154. Association for Computational Linguistics, 2017a. doi: 10.18653/v1/P17-2023. URL https://doi.org/10.18653/v1/P17-2023. 24, 25

Lei Shu, Hu Xu, and Bing Liu. DOC: deep open classification of text documents. In Martha Palmer, Rebecca Hwa, and Sebastian Riedel, editors, *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pages 2911–2916. Association for Computational Linguistics, 2017b. doi: 10.18653/v1/d17-1314. URL https://doi.org/10.18653/v1/d17-1314. 42

Lei Shu, Hu Xu, and Bing Liu. Unseen class discovery in open-world classification. *CoRR*, abs/1801.05609, 2018. URL http://arxiv.org/abs/1801.05609. 42

Daniel L. Silver, Geoffrey Mason, and Lubna Eljabu. Consolidation using sweep task rehearsal: Overcoming the stability-plasticity problem. In Denilson Barbosa and Evangelos E. Milios, editors, *Advances in Artificial Intelligence - 28th Canadian Conference on Artificial Intelligence, Canadian AI 2015, Halifax, Nova Scotia, Canada, June 2-5, 2015, Proceedings*, volume 9091 of *Lecture Notes in Computer Science*, pages 307–322. Springer, 2015. doi: 10.1007/978-3-319-18356-5_27. URL https://doi.org/10.1007/978-3-319-18356-5_27. 24, 25

Herbert A. Simon and Jonathan Schaeffer. Chapter 1 the game of chess. In *Handbook of Game Theory with Economic Applications*, pages 1–17. Elsevier, 1992. doi: 10.1016/s1574-0005(05)80004-9. URL https://doi.org/10.1016/s1574-0005(05)80004-9. 8, 49, 50

Herbert Alexander Simon. *Models of man: Social and rational: Mathematical essays on rational human behavior in a social setting*. Wiley, 1957. 2, 12

Jake Snell, Kevin Swersky, and Richard S. Zemel. Prototypical networks for few-shot learning. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Confer-*

ence on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA, pages 4077–4087, 2017. URL http://papers.nips.cc/paper/6996-prototypical-networks-for-few-shot-learning. 36

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012. 107

Ray J. Solomonoff. A system for incremental learning based on algorithmic probability. 1989. 138

Ray J. Solomonoff. Progress in incremental machine learning. 2003. 138

Matthew J. Streeter and Daniel Golovin. An online algorithm for maximizing submodular functions. In *Technical Report CMU-CS-07-171*. Carnegie Mellon University, 2007. 84

Matthew J. Streeter and Daniel Golovin. An online algorithm for maximizing submodular functions. In Daphne Koller, Dale Schuurmans, Yoshua Bengio, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8-11, 2008*, pages 1577–1584. Curran Associates, Inc., 2008. URL https://proceedings.neurips.cc/paper/2008/hash/5751ec3e9a4feab575962e78e006250d-Abstract.html. i, 9, 81, 83, 100, 101, 102, 103, 106, 108, 110, 207, 208

Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *CoRR*, abs/1712.06567, 2017. URL http://arxiv.org/abs/1712.06567. 40

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018. ISBN 0262039249. 5, 42, 115

Richard S. Sutton, David A. McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In Sara A. Solla, Todd K. Leen, and Klaus-Robert Müller, editors, *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*, pages 1057–1063. The MIT Press, 1999. 43

F. Tanaka and M. Yamamura. An approach to lifelong reinforcement learning through multiple environments. In *Proceedings of the Sixth European Workshop on Learning Robots*, pages 93–99, 1997. 43

Matthew E. Taylor, Peter Stone, and Yaxin Liu. Transfer learning via inter-task mappings for temporal difference learning. *J. Mach. Learn. Res.*, 8:2125–2167, 2007. doi: 10.5555/1314498.1314569. URL https://dl.acm.org/doi/10.5555/1314498.1314569. 44

Moshe Tennenholtz. Program equilibrium. *Games Econ. Behav.*, 49(2):363–373, 2004. doi: 10.1016/j.geb.2004.02.002. URL https://doi.org/10.1016/j.geb.2004.02.002. 4, 15, 16

Chen Tessler, Shahar Givony, Tom Zahavy, Daniel J. Mankowitz, and Shie Mannor. A deep hierarchical approach to lifelong learning in minecraft. In Satinder Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 1553–1561. AAAI Press, 2017. URL http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14630. 10, 43, 173

Sebastian Thrun. Is learning the n-th thing any easier than learning the first? *NIPS*, page 640–646, 1996. 11, 24, 25, 36

Sebastian Thrun and Tom M. Mitchell. Lifelong robot learning. *Robotics Auton. Syst.*, 15(1-2):25–46, 1995. doi: 10.1016/0921-8890(95)00004-Y. URL https://doi.org/10.1016/0921-8890(95)00004-Y. 5, 23, 43, 117

Sebastian Thrun and Lorien Y. Pratt, editors. *Learning to Learn*. Springer, 1998. ISBN 978-1-4613-7527-2. doi: 10.1007/978-1-4615-5529-2. URL https://doi.org/10.1007/978-1-4615-5529-2. 35

Ryan Turner, David Eriksson, Michael McCourt, Juha Kiili, Eero Laaksonen, Zhen Xu, and Isabelle Guyon. Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020. In *NeurIPS 2020 Competition and Demonstration Track*, pages 3–26. PMLR, 2021. 82, 107

Uber. Bayesmark documentation, 2020. URL https://bayesmark.readthedocs.io/en/latest/index.html. 108, 213

Leslie G. Valiant. A theory of the learnable. In Richard A. DeMillo, editor, *Proceedings of the 16th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1984, Washington, DC, USA*, pages 436–445. ACM, 1984. doi: 10.1145/800057.808710. URL https://doi.org/10.1145/800057.808710. 20

Wiebe van der Hoek, Cees Witteveen, and Michael J. Wooldridge. Program equilibrium - a program reasoning approach. *Int. J. Game Theory*, 42(3):639–671, 2013. doi: 10.1007/s00182-011-0314-6. URL https://doi.org/10.1007/s00182-011-0314-6. 16

Ricardo Vilalta and Youssef Drissi. A perspective view and survey of meta-learning. *Artif. Intell. Rev.*, 18(2):77–95, 2002. doi: 10.1023/A:1019956318069. URL https://doi.org/10.1023/A:1019956318069. 35

Oriol Vinyals, Charles Blundell, Tim Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 3630–3638, 2016. URL http://papers.nips.cc/paper/6385-matching-networks-for-one-shot-learning. 36

Chang Wang and Sridhar Mahadevan. Manifold alignment without correspondence. In Craig Boutilier, editor, *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 1273–1278, 2009. URL http://ijcai.org/Proceedings/09/Papers/214.pdf. 44

Shuai Wang, Zhiyuan Chen, and Bing Liu. Mining aspect-specific opinion using a holistic lifelong topic model. In Jacqueline Bourdeau, Jim Hendler, Roger Nkambou, Ian Horrocks, and Ben Y. Zhao, editors, *Proceedings of the 25th International Conference on World Wide Web, WWW 2016, Montreal, Canada, April 11 - 15, 2016*, pages 167–176. ACM, 2016. doi: 10.1145/2872427.2883086. URL https://doi.org/10.1145/2872427.2883086. 40

Zhi Wang, Chunlin Chen, and Daoyi Dong. Lifelong incremental reinforcement learning with online bayesian inference. *IEEE Trans. Neural Networks Learn. Syst.*, 33(8):4003–4016, 2022. doi: 10.1109/TNNLS.2021.3055499. URL https://doi.org/10.1109/TNNLS.2021.3055499. 11, 44, 173, 174

Di Wei, Yu Gu, Yumeng Song, Zhen Song, Fangfang Li, and Ge Yu. Incregnn: Incremental graph neural network learning by considering node and parameter importance. In Arnab Bhattacharya, Janice Lee, Mong Li, Divyakant Agrawal, P. Krishna Reddy, Mukesh K. Mohania, Anirban Mondal, Vikram Goyal, and

Rage Uday Kiran, editors, *Database Systems for Advanced Applications - 27th International Conference, DASFAA 2022, Virtual Event, April 11-14, 2022, Proceedings, Part I*, volume 13245 of *Lecture Notes in Computer Science*, pages 739–746. Springer, 2022. doi: 10.1007/978-3-031-00123-9_59. URL https://doi.org/10.1007/978-3-031-00123-9_59. 188

Gellért Weisz, András György, and Csaba Szepesvári. LEAPSANDBOUNDS: A method for approximately optimal algorithm configuration. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 5254–5262. PMLR, 2018. URL http://proceedings.mlr.press/v80/weisz18a.html. 21, 79

Gellért Weisz, András György, and Csaba Szepesvári. Capsandruns: An improved method for approximately optimal algorithm configuration. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 6707–6715. PMLR, 2019. URL http://proceedings.mlr.press/v97/weisz19a.html. 5, 22, 79

Aaron Wilson, Alan Fern, Soumya Ray, and Prasad Tadepalli. Multi-task reinforcement learning: a hierarchical bayesian approach. In Zoubin Ghahramani, editor, *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvallis, Oregon, USA, June 20-24, 2007*, volume 227 of *ACM International Conference Proceeding Series*, pages 1015–1022. ACM, 2007. doi: 10.1145/1273496.1273624. URL https://doi.org/10.1145/1273496.1273624. 11, 44, 173

David H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259,

1992. doi: 10.1016/S0893-6080(05)80023-1. URL https://doi.org/10.1016/S0893-6080(05)80023-1. 35

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Trans. Neural Networks Learn. Syst.*, 32(1):4–24, 2021. doi: 10.1109/TNNLS.2020.2978386. URL https://doi.org/10.1109/TNNLS.2020.2978386. 178, 189

Xiaojin Zhu, Adish Singla, Sandra Zilles, and Anna N. Rafferty. An overview of machine teaching. *CoRR*, abs/1801.05927, 2018. URL http://arxiv.org/abs/1801.05927. 46

Shlomo Zilberstein. Metareasoning and bounded rationality. In Michael T. Cox and Anita Raja, editors, *Metareasoning - Thinking about Thinking*, pages 27–40. MIT Press, 2011. doi: 10.7551/mitpress/9780262014809.003.0003. URL https://doi.org/10.7551/mitpress/9780262014809.003.0003. 19

Shlomo Zilberstein and Stuart Russell. Optimal composition of real-time systems. *Artif. Intell.*, 82(1-2):181–213, 1996. doi: 10.1016/0004-3702(94)00074-3. URL https://doi.org/10.1016/0004-3702(94)00074-3. 4, 9, 20

Martin Zinkevich, Michael Johanson, Michael H. Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. *NIPS*, pages 1729–1736, 2007. 66, 71, 202, 203