Tel Aviv University
The Raymond and Beverly Sackler
Faculty of Exact Sciences
School of Computer Science

Algorithms for Reinforcement Learning

Dissertation submitted
in partial fulfillment of the requirements
for the degree of doctor of philosophy

By Eyal Even-Dar

Under the supervision of Prof. Yishay Mansour

Submitted to the senate of Tel Aviv University
June 2005

Abstract

The main focus of this dissertation is the study of the behavior of agents in real world scenarios. In the recent decade, there has been an emerged interest towards the field of Reinforcement learning as a tool for solving many real world problems and optimizing large stochastic problems. In this thesis we focus on the theoretical analysis of Reinforcement learning. Our contribution is in providing learning and planning algorithms for Multi armed bandit, Markov Decision Processes and Partially Observable Markov Decision Processes.

While in applications tackled by Reinforcement learning tools, the agents seeks to learn a policy which achieves the global optimum, there exists many setting where agents are selfish and competing with each other and cannot attend the global optimum and can only achieve local optimum. For such environments we use non cooperative game theory setting. Therein, our main concern is the convergence time to Nash equilibrium, the stable point of the system.

Contents

Abstract **iii**

I **1**

1 Introduction **3**

1.1 Outline of Thesis 4

2 Model and Background **7**

2.1 PAC Algorithms 7

2.2 Multi-Armed bandit 8

2.3 Markov Decision Process 9

2.3.1 Solving MDPs 11

2.4 Learning in MDPs 12

2.4.1 Model Free Learning 13

2.4.2 Model Based Learning 14

2.5 Approximate solutions for MDPs 14

2.6 Partially Observable Markov Decision Processes 15

2.7 Routing, Load Balancing and Game Theory 17

2.7.1 Background 18

2.8 Concentration Bounds 20

3 PAC Bounds for the Multi-Armed Bandit **21**

3.1 Successive Elimination 22

3.2 Median elimination 26

4 Phased Learning **29**

5 Action elimination in MDPs **35**

5.1 Model-Based Learning 35

5.1.1 Finite Horizon 36

5.1.2 Discounted Return - Infinite Horizon 37

5.2 Model-Free Learning 41

5.3 Experiments 44

6 Optimistic Q-learning **51**

7 Incremental Q-learning **57**

7.1 Experiments 61

8 Markov Decision Process Minimization **63**

8.1 Hardness of epsilon Equivalence 65

8.2 L_1 versus L_∞ 66

8.3 Approximation scheme 68

8.3.1 A heuristic Finer Partition 73

8.4 Markov Decision Process Equivalence 74

9 The On-line MDP **77**

9.1 Mixing Time 78

9.2 Best Expert Algorithms 81

9.3 Idealized Settings 83

9.4 The General Setting 87

9.4.1 The Analysis 88

10 Learning in POMDPs **93**

10.1 Homing Strategies 95

10.2 Reinforcement Learning with Homing 97

10.2.1 Policy Search 98

10.2.2 A Model Based Algorithm 101

11 planning in POMDPs **105**

11.1 Grid Based Algorithms for POMDPs 106

11.2 Multiplicity Automata 108

11.3 Multiplicity Automata and Partially Observable MDP 110

11.4 A Minimal Basis 112

11.4.1 POMDP to Minimal Matrix Representation 112

11.4.2 A Modified Belief MDP 114

11.4.3 Barycenteric Spanner - Improving the basis 115

11.5 Planning Algorithm 117

11.5.1 The Algorithm 117

11.5.2 Algorithm Analysis 118

12 Convergence Time To Nash Equilibrium **121**

12.1 Centralized Control 122

12.1.1 Potential Games 122

12.1.2 Unrelated Machines 123

12.1.3 Related Machines 126

12.1.4 Identical Machines 133

12.2 Distributed Settings 140

12.2.1 Two Machines 141

12.2.2 Multiple Machines 148

12.2.3 Nash Rerouting policies for Multiple Machines 152

Bibliography **157**

List of Figures

5.1 Example of a Queue of size 5 with three types of packets with values $1, 20, 150$. The discount factor is set to 0.99. We disregard the full queue state in which every action is optimal. We repeated each experiment 15 times and the error bars represent 1 standard deviation. 46

5.2 Example of a Queue of size 5 with three types of packets with values $1, 20, 150$. The discount factor is set to 0.99. This figure demonstrates the elimination rate. We repeated each experiment 15 times and the error bars represent 1 standard deviation. 47

5.3 Example of a 20 state sparse randomly generated MDPs with 50 actions in each state, where $\gamma = 0.833$ (as in [88].) The precision is the distance of the Q-function from the optimal Q-function. We repeated each experiment 10 times and the error bars represent 1 standard deviation. 48

5.4 Example of a 20 state dense randomly generated MDPs with 50 actions in each state, $\gamma = 0.9$. The error bars represent 1 standard deviation. 48

5.5 Example of Howard's Automobile Replacement Problem, where the discount factor, γ, is 0.833. The norm is the steady state norm. The error bars represent 1 standard deviation. 49

7.1 Example of 50 states MDP, where the discount factor, γ, is 0.9. The leaning rate of both Incremental and epsilon greedy Q-learning is set to 0.8. The dashed line represents the epsilon greedy Q-learning. . . . 61

8.1 An example in which two MDPs are ε-equivalent in L_∞ norm but an optimal policy in M_ε is far from optimal in M. Let $n = \lceil 1/\varepsilon \rceil$. The rewards are $R(1,u) = R(1,a) = 0, R(2,a) = R(3,a) = 1, R(2n+4,a) = R_{max}, R(2n+5,a) = 0$ $\forall k \in \{4, 2n+3\}$ $R(k,a) = 2\varepsilon k$. In states $2,3$ the next state distribution is uniform. We note that there are only two policies in M, one performs a at state 1 and other performs u at state 1. In M_ε the two policies coincide. 67
8.2 The graph built by *CheckConsistent* procedure 75

11.1 A deterministic automaton over $\Sigma = \{a, b, c, d\}$ that accepts a string only if it contains the substring ab, the accepting state is the rightmost state. 109

Part I

Chapter 1

Introduction

Many sequential decision problems - ranging from dynamic resource allocation to robotics can be formulated in terms of stochastic control and solved by methods of Reinforcement learning. Therefore, Reinforcement learning (a.k.a Neuro Dynamic Programming) has become one of the major approaches in tackling real life problems.

Many difficult problems arise while trying to model real world problems in terms of Reinforcement learning, several of them are due to the environment size, which can be reflected in both large state space and large action space. A different inherent problem is the partial observability, where the agent cannot observe the exact state and can only observe a signal of the state.

Traditionally, research in Reinforcement learning and Operation research focused on providing algorithms which have good asymptotic behavior, i.e. convergence in the limit to the optimal policy (See [12, 99]). Recently an emerged interest was directed through PAC style convergence bounds (a.k.a as sample complexity bounds) of RL algorithms (See [60, 58, 54, 35]). Parts of this thesis provides insights on the sample complexity of several RL algorithms, and how can the sample complexity be reduced, we would also try give less conservative stopping conditions for several algorithms (rather than using the classical sample complexity bounds as stopping conditions).

Facing Partial observability is among the hardest problems in Reinforcement learning and the even planning becomes intractable there (See [71] for complexity results). Another difficulty arising while learning in partially observed environment is that of no "resets", i.e. the agent cannot get back to its starting state and thus it is not clear what the agent learns. In most applications far this problem is solved artificially by allowing a "reset button" to the learner. In this thesis we would show to overcome this problem theoretically without the restricted assumption of "reset button".

While in Reinforcement learning the main concern of this thesis is a single agent who faces nature, there exists many real world applications in which many agents are competing in the same environment for the same resources. For such environments we use non cooperative game setting to analyze the agents behavior. In such settings we do not try to compute what is the best action for the agent or what the agents can achieve by cooperating, and we focus in how fast the system will reach a stable point, Nash equilibrium, regardless its quality.

1.1 Outline of Thesis

This Ph.D. dissertation is centered around the behavior of agents in real world scenarios. The first part is devoted to single agent who tries to learn the optimal policy. The second part is devoted to selfish agents who are being constantly greedy, and we are focused there on the convergence time to Nash equilibrium.

In Chapter 3 we discuss the most basic Reinforcement learning, the multi armed bandit settings. We analyze the sample complexity of several algorithms in the MAB and give algorithm which have asymptotic optimal sample complexity.

In Chapter 4 we extend the Multi armed bandit algorithms to Markov decision process under the parallel sampling model [60] and improve the upon the best known results and provide almost optimal bounds on the sample complexity.

In Chapter 5 we further extend the results into standard MDPs settings. Our algorithms for MDPs provide both bounds on the sample complexity and also stopping conditions. Stopping conditions are rarely found in RL, their importance in real world applications is very high.

In Chapter 6, we provide a theoretical analysis of the convergence of the widely used optimistic Q-learning. This algorithms is standard Q-learning where the initial value are initialized above their expected value (optimistically) and then follow greedy policy.

In Chapter 7 we provide novel algorithms which can be viewed as analog of the known Boltzmman machine and ε-greedy Q-learning. Those algorithms are shown to have similar behavior to Boltzmman machine and ε-greedy experimentally, and to converge with probability one without using additional randomization.

While the previous Chapter were concentrated on learning algorithm in MDPs, Chapter 8 provides insights on the MDP structure. We provide their algorithm which compute the minimal equivalent MDP and algorithms which compute an almost equivalent minimal MDP. We also provide hardness results concerning computing the almost equivalent minimal MDP.

In Chapter 9 we consider a different setting and handle cases where the environment is only partially Markovian and partially controlled by an adversary. In this setting we give regret bounds with respect to the optimal stationary policy that goes to zero as time approaches infinity.

Chapters 10, 11 provide both learning and planning algorithms for Partially Observable MDPs. The first chapter is devoted to learning in POMDPs where “resets” are not available. We provide a policy which enables the agent to reset himself to an approximate starting state, and thus enable him to learn. The second part is a planning algorithm which is based on Multiplicity automata representation of the PODMP. After showing the equivalence we derive an algorithm which is only exponential in the Predictive state representation (PSR) [68]. Therefore, for structured problems as evident in [94] the planning problem becomes tractable.

Finally, Chapter 12 is devoted to selfish agents in load balancing scenario [9]. We provide convergence results (upper bound and lower bounds) for different setting, distributed/central control, weighted/indetical users, related/unrelated machines. We show that for distributed control the convergence time for identical machines and users is logarithmic, while for centralized control we show that the convergence crucially depends on the scheduler (the centralized control) and the convergence time can vary from linear to exponential time.

Chapter 2

Model and Background

In this chapter we describe the various models considered in this thesis and provide for each literature overview. We start by providing the PAC learning definition and then describing the various Reinforcement Learning models. At the beginning, Section 2.2, we describe the simplest model, the Multi Armed Bandit (MAB), we then describe the most popular RL model Markov Decision Process (MDP) in Section 2.3, in Section 2.3 we describe the most challenging RL model, the Partially Observable Markov Decision Process (POMDP). Finally we describe the Load Balancing model and few basic game theory notations in Section which is related to the game theory results.

2.1 PAC Algorithms

Traditionally in learning theory (see [102, 61]), a learner receives a training set $\{x, y\}$, where $y = f(x)$ (possibly with additional noise) from a given distribution $D(x)$ and she tries to learn a target function g such that $g(x)$ is "similar" to $f(x)$ with high probability. The major question is how large should be training set to ensure similarity between g and f. The sample complexity of a PAC algorithm is the size of the training set to ensure that with probability at least $1 - \delta$ we have $\Pr_{x \sim D(x)}(g(x) \neq f(x)) \leq \varepsilon$. The major difference between this type of analysis to the one encounters in this thesis is that in Reinforcement learning we "lack" of underlying distribution and our actions influence the next samples and the learning problem becomes more challenging .

In this thesis we say that an algorithm is a PAC algorithm with sample complexity T if it produces an ε-optimal policy with probability at least $1-\delta$ after T samples.

2.2 Multi-Armed bandit

The model is comprised of a set of arms A with $n = |A|$. When sampling arm $a \in A$ a reward which is a random variable $R(a)$ is received. We assume that the reward is binary, i.e., for every arm $a \in A$ the reward $R(a) \in [0, 1]$ Denote the arms by $a_1, \cdots, a_n$ and $p_i = \mathbf{E}[R(a_i)]$. For simplicity of notations we enumerate the arms according to their expected reward $p_1 > p_2 > ... > p_n$.

An arm with the highest expected reward is called the *best arm*, and denoted by a^*, and its expected reward r^* is the *optimal reward*. An arm whose expected reward is strictly less than r^*, the expected reward of the best arm, is called a *non-best arm*. An arm a is called an *ϵ-optimal arm* if its expected reward is at most ϵ from the optimal reward, i.e., $\mathbf{E}[R(a)] \geq r^* - \epsilon$.

An algorithm for the MAB problem, at each time step t, samples an arm a_t and receives a reward r_t (distributed according to $R(a_t)$). When making its selection the algorithm may depend on the history (i.e., the actions and rewards) up to time $t-1$. Eventually the algorithm must commit to a single arm and select it.

Next we define the desired properties of an algorithm formally.

Definition 2.1. An algorithm is a (ϵ, δ)-PAC algorithm for the multi armed bandit with *sample complexity* T, if it outputs an ϵ-optimal arm, a', with probability at least $1-\delta$, when it terminates, and the number of time steps the algorithm performs until it terminates is bounded by T.

The multi-armed bandit problem has been widely studied in a variety of setups. The problem was first considered in the 50's in the seminal work of Robbins [90] that derives strategies that asymptotically attain an average reward that converges in the limit to the reward of the best arm. The multi-armed bandit problem was later studied in discounted, Bayesian, Markovian, expected reward, and adversarial setups. (See [11] for a review of the classical results on the multi-armed bandit problem.)

Two notable concepts are the Gittins index and efficiency of allocation rules. In the seminal work of Gittins and Jones [41] a simple optimal decision rule was derived, this rule is known as the Gittins Index. The model in question is Bayesian, where the a-priory distribution of each arm is uniform and reward is discounted. The Gittins index is a way to quantify how good is an arm expected to be, and the derived policy is to simply choose the arm with the highest index. For the expected regret model, where the regret is defined as the difference between the optimal return and the online algorithm return, the seminal work of Lai and Robbins [65] provides tight bounds as a function of the Kullback-Leibler divergence between the arms reward distribution, and a logarithmic growth with the number of steps. The bounds of [65] were shown to be efficient, in the sense that the convergence rates are optimal. The adversarial multi-armed bandit problem was considered in [5, 6], where it was shown that the expected regret grows proportionally to the square root of the number of steps.

2.3 Markov Decision Process

A Markov Decision Process (MDP) formalizes the following problem, an agent is in an environment, which is composed from different states, in each time step the agent performs an action and as a result observes a signal. The signal is composed from the reward to the agent and the state it reaches in the next time step. More formally the MDP is defined as follows,

Definition 2.2. A Markov Decision process (MDP) M is a 4-tuple (S, A, P, R), where S is a set of the states, A is a set of actions, $P^a_{s,s'}$ is the transition probability from state s to state s' when performing action $a \in A$ in state s, and $R(s, a)$ is the reward received when performing action a in state s.

A strategy for an MDP assigns, at each time t, for each state s a probability for performing action $a \in A$, given a history $F_{t-1} = \{s_1, a_1, r_1, ..., s_{t-1}, a_{t-1}, r_{t-1}\}$ which includes the states, actions and rewards observed until time $t-1$. While executing a strategy π we perform at time t action a_t in state s_t and observe a reward r_t (distributed according to $R(s_t, a_t)$), and a next state s_{t+1} (distributed according to

$P^{a_t}_{s_t,\cdot}$). We combine the sequence of rewards into a single value called the *return*. The agent goal is to maximize the return. There several ways to define the return.

- *finite horizon* The return of policy π , $V^\pi = \sum_{t=0}^{H} r_t$ for a given horizon H.

- *discounted return*, which has a parameter $\gamma \in (0,1)$, and the discounted return of policy π is $V^\pi = \sum_{t=0}^{\infty} \gamma^t r_t$, where r_t is the reward observed at time t.

- *undiscounted return* The return of policy π is $V^\pi = \lim_{t\to\infty} \frac{1}{t+1} \sum_{i=0}^{t} r_t$

While both the finite horizon and the discounted return are always defined, the undiscounted return might not exists. However, we only consider it in this thesis for uni-chain model for which it exists. Furthermore, the value of a policy π is independent of the starting state and we denote it as η_π. Throughout the the thesis, we assume that $R(s,a)$ is non-negative and bounded by R_{max}, i.e., for every $s,a: \ 0 \leq R(s,a) \leq R_{max}$. This implies that the discounted return is bounded by $V_{max} = \frac{R_{max}}{1-\gamma}$; for the finite horizon the return is bounded by HR_{max}; and for the average reward criteria it is bounded by R_{max} . We define a value function for each state s, under policy π, as $V^\pi(s) = E^\pi[\sum_{i=0}^{\infty} r_i \gamma^i]$, where the expectation is over a run of policy π starting at state s, for the discounted return and $V^\pi(s) = E^\pi[\frac{1}{t+1}\sum_{i=0}^{t} r_i]$ for the undiscounted return; and $V^\pi(s) = E^\pi[\sum_{i=0}^{H} r_i$ for the finite horizon return. We further denote the state-action value function as using action a in state s and then following π as:

$$Q^\pi(s,a) = R(s,a) + \gamma \sum_{s'} P^a_{s,s'} V^\pi(s').$$

For the finite horizon

$$Q^\pi_t(s,a) = R(s,a) + \sum_{s'} P^a_{s,s'} V^\pi_{t-1}(s').$$

For the undiscounted return

$$Q^\pi_t(s,a) = R(s,a) - \eta_(\pi) + \sum_{s'} P^a_{s,s'} Q_\pi(s',\pi).$$

Let π^* be an optimal policy which maximizes the return from any start state. There exists such a policy which is deterministic and stationary for both discounted return for any MDP and the undiscounted return for uni-chain MDPs [88]. This implies that for any policy π and any state s we have $V^{\pi^*}(s) \geq V^{\pi}(s)$, and $\pi^*(s) = \text{argmax}_a(Q^{\pi^*}(s,a))$. We use V^* and Q^* for V^{π^*} and Q^{π^*}, respectively. We say that a policy π is ϵ-optimal if $\|V^* - V^{\pi}\|_\infty \leq \epsilon$. We also define the policy $Greedy(Q)$ as the policy that prescribes in each state the action that maximizes the Q-function in the state, i.e., $\pi(s) = \text{argmax}_a Q(s,a)$.

For a given trajectory let: $T^{s,a}$ be the set of times in which we perform action a in state s and $T^{s,a,s'}$ be a subset of $T^{s,a}$ in which we reached state s'. Also, $\#(s,a,t)$ is the number of times action a is performed in state s up to time t, i.e., $|T^{s,a} \cap \{1,2,3,\ldots,t\}|$. We similarly define $\#(s,a,s',t)$ as $|T^{s,a,s'} \cap \{1,2,3,\ldots,t\}|$. Next we define the empirical model at time t. Given that $\#(s,a,t) > 0$ we define the empirical next state distribution at time t as $\hat{P}^a_{s,s'} = \frac{\#(s,a,s',t)}{\#(s,a,t)}$ and $\hat{R}(s,a) = \frac{\sum_{t \in T^{s,a}} r_t}{\#(s,a,t)}$. If $\#(s,a,t) = 0$ the empirical model and the reward can be chosen arbitrarily. We define the expectation of the empirical model as $\hat{E}_{s,s',a}[V(s')] = \sum_{s' \in S} \hat{P}^a_{s,s'} V(s')$. To simplify the notations we omit s,a in the notations $\hat{E}_{s'}$ whenever evident.

2.3.1 Solving MDPs

We now introduce the three standard methods to solve an MDP when the MDP is given. These methods are efficient when the MDP is represented in its tabular form, but are not efficient for compact representations, such as Factored MDPs.

Value Iteration: The value iteration is defined as follows, start with some initial value function and then iterate using the Bellman operator, i.e.

$$\begin{aligned} V_0(s) &= C_s \\ V_{t+1}(s) &= TV_t(s) \end{aligned}$$

This method relies on the fact that the Bellman operator is contacting. Therefore, the distance between the optimal value function and current value function contract by a factor of γ with respect to max norm in each iteration.

Policy Iteration: This algorithm starts with initial policy π_0 and iterates over polices. The algorithm has two phases for each iteration. In the first phase, *Value determination step* a value function for π_t is calculated, by finding the fixed point of $T_{\pi_t} V_{\pi_t} = V_{\pi_t}$. The second phase, *Policy Improvement step*, is taking the next policy, $V_{\pi_{t+1}}$ as a greedy policy with respect to V_{π_t}. Policy iteration converges with less iterations than value iteration. In practice the convergence of PI is very fast. However, the only sub exponential bound for PI is given in [74].
Linear Programming: Formulates and solves an MDP as liner program (LP). The LP variables are $V_1, \ldots, V_n$, where $V_i = V(s_i)$. The definition is:

$$\begin{aligned} \text{Variables:} \quad & V_1, \ldots, V_n \\ \text{Minimize:} \quad & \sum_i V_i \\ \text{Subject to:} \quad & V_i \geq [R(s_i, a) + \gamma \sum_j P_{s_i, s_j}(a) V_j] \quad \forall a \in A, s_i \in S \end{aligned}$$

It is worth noting that for exact solutions linear programming tend to behave inferior to Value Iteration and Policy Iteration. We should also note the the model based algorithms use one of those approaches after constructing the approximate MDP.

2.4 Learning in MDPs

Learning in MDPs is one of the most challenging problems in AI, we provide here a survey of the most popular way for learning the optimal policy for other problems such as evaluating polices we refer the reader to [99]. The methods are divided to two categories, the first is the model free learning and the second is the model based learning. We first describe the model free learning, while putting an emphasis on Q-learning and then describe the model based learning.

2.4.1 Model Free Learning

Model free learning also known as direct methods try to learn the optimal policy directly without evaluating the model. The model free algorithms can also be categorized into two groups, the policy search algorithm and the value function algorithms. The first one try to find the optimal policy by searching in the policy space and we will elaborate on them in the approximate solutions section. The later try to estimate the Q function. We start by describing the widely used Q-learning, which represents the second approach.

Q learning The Q-learning algorithm [103] estimates the state-action value function (for discounted return) as follows:

$$Q_{t+1}(s,a) = (1-\alpha_t(s,a))Q_t(s,a) + \alpha_t(s,a)(r_t(s,a) + \gamma V_t(s'))$$

where s' is the state reached from state s when performing action a at time t, and $V_t(s) = \max_a Q_t(s,a)$. We assume that $\alpha_t(s',a') = 0$ for $t \notin T^{s',a'}$.

A learning rate α_t is *well-behaved* if for every state action pair (s,a): (1) $\sum_{t=1}^{\infty} \alpha_t(s,a) = \infty$ and (2) $\sum_{t=1}^{\infty} \alpha_t^2(s,a) < \infty$. If the learning rate is well-behaved and every state action pair is performed infinitely often then Q-Learning converges to Q^* with probability 1 (see [12, 100, 104, 69, 53]).

The convergence rate of Q-learning has been analyzed rigourously in [35], and showed an interesting relationship between the convergence rate and the learning rate used in Q-learning. For a polynomial learning rate, one which is $1/t^\omega$ at time t where $\omega \in (1/2, 1)$, we showed that the convergence rate is polynomial in $1/(1-\gamma)$, where γ is the discount factor. In contrast we show that for a linear learning rate, one which is $1/t$ at time t, the convergence rate has an exponential dependence on $1/(1-\gamma)$.

Another important algorithm which bridges between Q-learning and monte carlo methods is the $TD(\lambda)$ algorithm. The λ parameter controls how much similar to monte carlo we are (Q-learning is $TD(0)$ and monte carlo algorithms are $TD(1)$). This algorithm is also known to converge with probability 1 (see [12]).

2.4.2 Model Based Learning

The model based algorithms are very simple to describe, they build an empirical model,$\hat{P}^a_{s,s'} = \frac{\#(s,a,s',t)}{\#(s,a,t)}$ and $\hat{R}(s,a) = \frac{\sum_{t \in T^{s,a}} r_t}{\#(s,a,t)}$. Then they use one of the standard methods, Linear programming, policy iteration and value iteration to find the optimal policy in the approximate model. Several works [59, 57, 20] studied the sample complexity of these methods, namely how much samples the algorithms need to find an almost optimal policy with high probability.

2.5 Approximate solutions for MDPs

In many cases the model is to complicated to be learnt or solved explicitly and approximate methods our in order. We survey here the major techniques used.

Policy search The policy search algorithms (a.k.a. policy gradient methods) [105, 26, 76] uses gradient methods to the find a good policy in a class of stochastic policies characterized by a parameter $\Theta \in R^k$. Note that those methods finds a stochastic policy rather than deterministic polices as done in the exact methods and they are ensured to converge to the optimal policy. The gradient is estimated using sequences of states and rewards between consecutive visit to the designated start state. The gradient methods often suffer from a large variance and slow convergence rate.

function Approximation: The term function approximation is due to the fact that it takes examples from a desired function (e.g., a value function) and attempts to construct from them an approximation of the entire function. Function approximation is an instance of supervised learning. In contrast to the tabular representation, this time a parameter vector Θ represents the value function. The challenge will be to learn the optimal parameter in the sense of minimum square error, i.e.

$$min_\Theta \sum_{s \in S} (V^\pi(s) - V(s,\Theta))^2$$

One of the most important function approximations is the linear function approximation,

$$V_t(s, \Theta) = \sum_{i=1}^{T} \phi_s(i)\Theta_t(i),$$

where each state has a set of vector features, ϕ_s. A feature based function approximation was analyzed and demonstrated in [101, 12].

Factored Markov Decision Process Factored Markov Decision Process (FMDP) [17]. In a FMDP the set of states is described via a set of random variables $X = \{X_1, \ldots, X_n\}$, where each X_i takes values in some finite domain $Dom(X_i)$. A state s defines a value $x_i \in Dom(X_i)$ for each variable X_i. We define a state transition model using dynamic bayasien network (DBN) [25]. We denote X_i as the variable in the current time and X'_i as the variable in the next time step. The transition graph of a DBN is a two layer DAG, G_T with nodes $\{X_1, \ldots, X_n, X'_1, \ldots, X'_n\}$. The parents of X'_i are in the graph are denoted by $Parents(X'_i)$. Each node X'_i has a conditional probability distribution, $P_T(X'_i|Parents(X'_i)$. The transition probability is defined as $P(s, s') = \Pi_i P_T(x_i|u_i)$, where u_i is the value of x in the parents of X'_i. Although the representation is efficient, not only that finding an ε-optimal policy is intractable [71] but it cannot be represented succinctly [2]. However, under few assumptions on the FMDP structure many heuristics [47, 63] have both theoretical guarantees and nice practical results.

2.6 Partially Observable Markov Decision Processes

A *Partially Observable Markov Decision Process* (POMDP) consists of:

- A finite set of states $S = \{s_1, \ldots, s_n\}$, $n = |S|$.
- A finite set of actions $A(s_i)$ associated with each state s_i. We assume identical action sets, i.e., for every state s_i, $A(s_i) = A$
- The transition probability matrix, $P(s, a, s')$, which is the probability of moving to state s' after performing action a at state s.

- A finite set of observable signals $O \times R$ where O is the observation set and R is the set of immediate rewards. We assume each immediate reward is bounded in $[0, 1]$. For simplicity, we assume that the possible observation signals are identical for all states.

- An observation probability, given a state s_i and action $a \in A(s_i)$, for an observation $o \in O \times R$, is $OB(o|s_i, a)$.

We define the return of a policy π in a POMDP, R^π as the expected sum of its discounted rewards, where there is a discount factor $\gamma \in (0, 1)$, i.e., $R^\pi(s) = \mathbf{E}[\sum_{t=0}^{\infty} \gamma^t r_t|\pi, s]$.

An equivalent problem to finding the optimal strategy in a POMDP is solving the associated *Belief MDP*. The Belief MDP is a continuous state MDP, in which the states are distributions over the original POMDP states, i.e., a state x is a distribution over S such that $x(i)$ is the probability of being at state s_i. These distributions over states are referred to as *belief states*. The transition probabilities in this MDP are defined according to the MDP transition and observation probability (The update is bayesian). Similarly to finite state MDP we define the value function under policy π from state s as $V^\pi(s) = \mathbf{E}[\sum_{t=0}^{\infty} \gamma^t r_t|\pi, s]$ and the Q-function as $Q^\pi(s, a) = \mathbf{E}[\sum_{t=0}^{\infty} \gamma^t r_t|\pi, s, a_0 = a]$. We denote the optimal policy as π^* and denotes its value and Q function as V^* and Q^*.

In POMDP the problem of finding an optimal or near optimal control become intractable [71]. In case where we have no observations at all the problem become undecidable [72]. Those limitations on the control problem imply that the learning problem under those conditions will be at least as hard.

Several ways to tackle POMDPs comes from the MDPs solutions, for instance the policy gradient methods are also used see [86, 87]. A specific ways to solve POMDPs are using the belief state MDP. The belief state MDP is a continuous MDP, where each state is a distribution over the original states. A class of approximation schemes are grid based algorithms for POMDPs (See [70, 19, 49, 15]. In essence, these grid based algorithms discretize the *belief states* in the POMDP in some manner into a finite state MDP. Another approach for planning is Sondik representation [98] to calculate the optimal value function in the belief MDP, however this representation can be double exponential.

2.7 Routing, Load Balancing and Game Theory

In load balancing (routing on parallel links) scenario there are m parallel machines (routes) and n independent jobs (users). Each job selects exactly one machine.

Machines Model. We consider identical, related and unrelated machines. We denote by S_i the speed of M_i. Let S_{min} and S_{max} denote the minimal and maximal speed, respectively. WLOG, we assume that $S_{min} = 1$. For identical and unrelated machines we have $S_i = 1$ for $1 \leq i \leq m$.

Jobs Model. We consider both restricted and unrestricted assignments of jobs to machines. In the unrestricted assignment case each job can select any machine while in the restricted assignment case each job J can only select a machine from a pre-defined subset of machines denoted by $R(J)$.

For a job J, we denote by $w_i(J)$ the weight of J on machine M_i (where $i \in R(J)$) and by $M(J,t)$ the index of the machine on which J runs at time t. When considering identical machines, each job J has a weight $w(J) = w_i(J)$. We denote by W the maximal total weight of the jobs, that is $W = \sum_{i=1}^{n} \max_{j \in R(J_i)}\{w_j(J_i)\}$ and by $w_{max} = \max_i \max_{j \in R(J_i)}\{w_j(J_i)\}$ the maximum weight of a job.

We consider the following weight settings: *General weight setting* – the weights may be arbitrary real numbers. *Discrete weight setting* – there are K different integer weights $w_1 \leq \ldots \leq w_K = w_{max}$. *Integer weight setting* – the weights are integers.

Load Model. We denote by $B_i(t)$ the set of jobs on machine M_i at time t. The load of a machine M_i at time t is the sum of the weights of the jobs that chose M_i, that is $L_i(t) = \sum_{J \in B_i(t)} w(J)$, and its normalized load is $T_i(t) = L_i(t)/S_i$. We also define $L_{max}(t) = \max_i\{L_i(t)\}$ and $T_{max}(t) = \max_i\{T_i(t)\}$. The cost of job J at time t is the normalized load on the machine $M(J,t)$, i.e., $T_{M(J,t)}(t)$. We define the *marginal* load with respect to a job to be the load in the system when this job is removed.

System Model. The *system state* consists of the current assignment of the jobs to the machines. The system starts in an arbitrary state and each job has a full knowledge of the system state. A job wishes to migrate to another machine, if and only if, after the migration its cost is strictly reduced. Before migrating between machines, a job needs to receive a grant from the centralized controller. The controller has no influence on the selection of the target machine by a migrating job, it just gives the job a permission to migrate. The above is known in the literature as an Elementary Stepwise System (ESWS) (see [16, 84]). Essentially, the controller serves as a critical section control. The execution is modeled as a sequence of steps and in each step one job changes its machine. Notice that if all jobs are allowed to move simultaneously, the system might oscillate and never ever reach a Nash equilibrium.

Let $A(t)$ be the set of jobs that may decrease the experienced load at time t by migrating to another machine. When a migrating job selects a machine which minimizes its cost (after the migration), we call to this *best-reply* policy. Otherwise, we call to this *improvement* policy.

The system is said to reach a *pure* (or deterministic) Nash equilibrium if no job can benefit from unilaterally migrating to another machine. The system is said to reach an ϵ-Nash equilibrium if no job can benefit more than ϵ from unilaterally migrating to another machine. We study the number of time steps it takes to reach a Nash equilibrium (or ϵ-Nash equilibrium) for different strategies of ESWS job scheduling.

2.7.1 Background

Routing is one of the most basic tasks in networking. Traditionally it has been view as a large multi-valued optimization problem, which the network provider has to address. Clearly, the selection of the load on the various routes greatly impacts the overall network performance. As part of the optimization process, the goals of different individual users are contrasted with the desire to achieve a high utilization of the network resources.

A different perspective of the routing problem, is to allow individual users to specify the path on which their traffic would be routed. In such setting each individual user optimizes her own utility function, and naturally tries to route her traffic on the least loaded path. An appropriate solution concept for such system is that of Nash equilibrium [83], where no user can benefit from rerouting her traffic. Although the distributed view of network routing has many interesting features, it carries the difficulty that the selfish users decisions might negatively impact the overall performance of the system.

Because of the highly attractive nature of decentralize distributed routing, networks researchers have looked for over a decade into issues involved in the design and implementation of such mechanism. One important line of research has been to try and "determine" the resulting equilibrium, with the motivation of allowing the network administrator to predict and control the resulting equilibrium. Therefore, cases in which the equilibrium is unique have been highly desired [66, 85].

The main focus in theoretical computer science, in this research direction, has been the study of the quality of the resulting Nash equilibrium. The work of Koutsoupias and Papadimitriou [64] defined the *coordination ratio*, which is the ratio of the cost of an optimal global objective function to the cost of the worst Nash equilibrium. Many following works have studied the coordination ratio in a wide range of routing and job scheduling scenarios [23, 24, 38, 64, 91].

The convergence that is our main focus was also studded in the past. In [85] it was shown that for general topology, under certain conditions, a unique equilibrium exists. They also show that in an Elementary Step System (ESS), where only one user reroutes in each time step, the best response dynamics converges to Nash equilibrium for two parallel links. In [66] it was shown that for parallel links and a specific cost function there exists a unique Nash equilibrium. They also show that in the ESS model the best response dynamics converges. In [75] selfish users can select their priority and pay accordingly, and the result is a unique Nash equilibrium which is also Max-Min Fairness.

The work of [44] studied a randomized model in which each user can select a random delay over continuous time. The continuous time implies that only one user tries to reroute at each specific time, and thus this model is very similar to the ESS model. In that model rerouting succeeds only if the user lowers its load. The work shows a simple randomized algorithm in which the expected number of rerouting attempts, until convergence to a Nash equilibrium, is polynomial in the number of links m and users n.

The work of [81] studied a model in which only one user is allowed to move in each time step, but the main interest was not in the equilibrium point but on the social value after a short best response path and therefore they were interested in the convergence time to an approximate solution and not to exact solution as we are.

2.8 Concentration Bounds

We often use large deviation bounds in this thesis, and therefore, we present the bounds at the beginning.

Lemma 2.3. *[50] Let X be a set, D be a probability distribution on X, and $f_1, ..., f_m$ be real-valued functions defined on X with $f_i : X \to [a_i, b_i]$ for $i = 1, ..., m$, where a_i and b_i are real numbers satisfying $a_i < b_i$. Let $x_1, \ldots, x_m$ be IID samples from D. Then we have the following inequality*

$$\Pr\left[\frac{1}{m}\sum_{i=1}^{m} f_i(x_i) - \left(\frac{1}{m}\sum_{i=1}^{m}\int_{a_i}^{b_i} f_i(x)D(x)\right) \geq \epsilon\right] \leq e^{-\frac{2\epsilon^2 m^2}{\sum_{i=1}^{m}(b_i-a_i)^2}}$$
$$\Pr\left[\frac{1}{m}\sum_{i=1}^{m} f_i(x_i) - \left(\frac{1}{m}\sum_{i=1}^{m}\int_{a_i}^{b_i} f_i(x)D(x)\right) \leq -\epsilon\right] \leq e^{-\frac{2\epsilon^2 m^2}{\sum_{i=1}^{m}(b_i-a_i)^2}}.$$

We also often used the relative Chernoff bound

The convergence proofs use extensively large deviation bounds, specifically the relative Chernoff bound [4], as stated in the following lemma.

Lemma 2.4. *[4] Let $0 < p < 1$, let $Z_1, \ldots, Z_n$ be independent binary random variables and $Z = \sum_{i=1}^{n} Z_i$, where $\sum_{i=1}^{n} \frac{E[Z_i]}{n} = p$, and let $\hat{p} = \frac{\sum Z_k}{n}$. Then,*

$$\Pr\left(p \leq \hat{p} + \sqrt{\frac{2p\ln(1/\delta)}{n}}\right) \geq 1-\delta \quad 0 \leq p \leq 1$$
$$\Pr\left(p \geq \hat{p} - \sqrt{\frac{3p\ln(1/\delta)}{n}}\right) \geq 1-\delta \quad \frac{\ln(1/\delta)}{3n} \leq p \leq 1$$
$$\Pr\left(p \geq \hat{p} - \frac{2\ln(1/\delta)}{n}\right) \geq 1-\delta \quad 0 \leq p \leq \frac{\ln(1/\delta)}{3n}$$

Chapter 3

PAC Bounds for the Multi-Armed Bandit

This chapter is based on [31]. In this chapter we look for an (ϵ, δ)-PAC algorithms for the MAB problem. Such algorithms are required to output with probability $1-\delta$ an ϵ-optimal arm. We start with a naive solution that samples each arm $1/(\epsilon/2)^2 \ln(2n/\delta)$ and picks the arm with the highest empirical reward. The sample complexity of this naive algorithm is $O(n/\epsilon^2 \log(n/\delta))$. The naive algorithm is described in Algorithm 1. In Section 3.1 we consider an algorithm that eliminates one arm after the other. In Section 3.2 we finally describe the Median Elimination algorithm whose sample complexity is optimal in the worst case.

> **Input** : $\varepsilon > 0$, $\delta > 0$
> **Output**: An arm
> **foreach** Arm $a \in A$ **do**
>   Sample it $\ell = \frac{4}{\epsilon^2} \ln(\frac{2n}{\delta})$ times;
>   Let $\hat{p}_a$ be the average reward of arm a;
> **end**
> Output $a' = \arg\max_{a \in A}\{\hat{p}_a\}$;

Algorithm 1: Naive Algorithm

Theorem 3.1. *The algorithm* Naive(ϵ, δ) *is an* (ϵ, δ)*-PAC algorithm with arm sample complexity* $O\left(\frac{n}{\epsilon^2} \log(\frac{n}{\delta})\right)$.

Proof. The sample complexity is immediate from the definition of the algorithm. We now prove it is an (ϵ, δ)-PAC algorithm. Let a' be an arm for which $\mathbf{E}(R(a')) < r^* - \varepsilon$. We want to bound the probability of the event $\hat{p}_{a'} > \hat{p}_{a^*}$.

$$\begin{aligned} P\left(\hat{p}_{a'} > \hat{p}_{a^*}\right) &\leq P\left(\hat{p}_{a'} > \mathbf{E}(R(a')) + \varepsilon/2 \text{ or } \hat{p}_{a^*} < r^* - \varepsilon/2\right) \\ &\leq P\left(\hat{p}_{a'} > \mathbf{E}[R(a')] + \varepsilon/2\right) + P\left(\hat{p}_{a^*} < r^* - \varepsilon/2\right) \\ &\leq 2\exp(-(\varepsilon/2)^2 \ell)\,, \end{aligned}$$

where the last inequality uses the Hoeffding inequality. Choosing $\ell = \frac{4}{\epsilon^2}\ln(\frac{2n}{\delta})$ assures that $P\left(\hat{p}_{a'} > \hat{p}_{a^*}\right) \leq \frac{\delta}{n}$. Summing over all possible a' we have that the failure probability is at most $(n-1)(\delta/n) < \delta$. □

3.1 Successive Elimination

The successive elimination algorithm attempts to sample each arm a minimal number of times and eliminate the arms one after the other. To motivate the successive elimination algorithm, we first assume that the expected rewards of the arms are known, but the matching of the arms to the expected rewards is unknown. Let $\Delta_i = p_1 - p_i > 0$. Our aim is to sample arm a_i for $(1/\Delta_i^2)\ln(n/\delta)$ times, and then eliminate it. This is done in phases. Initially, we sample each arm $(1/\Delta_n^2)\ln(n/\delta)$ times. Then we eliminate the arm which has the lowest empirical reward (and never sample it again). At the i-th phase we sample each of the $n-i$ surviving arms

$$O\left(\left(\frac{1}{\Delta_{n-i}^2} - \frac{1}{\Delta_{n-i+1}^2}\right)\log\left(\frac{n}{\delta}\right)\right)$$

times and then eliminate the empirically worst arm. The algorithm described as Algorithm 2 below. In Theorem 3.2 we prove that the algorithm is $(0, \delta)$-PAC and compute its sample complexity.

Input : $\delta > 0$, bias of arms $p_1, p_2, \ldots, p_n$

Output: An arm

Set $S = A$; $t_i = \lceil 4/\Delta_i^2 \ln(n/\delta) \rceil$; and $t_{n+1} = 0$, for every arm a: $\hat{p}_a = 0$, $i = 0$;

while $i < n - 1$ **do**

  Sample every arm $a \in S$ for $t_{n-i} - t_{n-i+1}$ times;

  Let $\hat{p}_a$ be the average reward of arm a (in all rounds);

  Set $S = S \setminus a_{\min}$, where $a_{\min} = \arg\min_{a \in S}\{\hat{p}_a\}$, $i = i + 1$;

end

Output S;

Algorithm 2: Successive Elimination Known Biases

Theorem 3.2. *Suppose that $\Delta_i > 0$ for $i = 2, 3, \ldots, n$. Then the Successive Elimination with Known Biases algorithm is an $(0, \delta)$-PAC algorithm and its arm sample complexity is*

$$O\left(\log(\frac{n}{\delta}) \sum_{i=2}^{n} \frac{1}{\Delta_i^2}\right). \tag{3.1}$$

Proof. We claim that this algorithm outputs the best arm with probability that is not smaller than $1 - \delta$. This is done by showing that, in each phase, the probability of eliminating the best arm is bounded by $\frac{\delta}{n}$. It is clear that the failure probability at phase i is maximized if all the $i - 1$ worst arms have been eliminated in the first $i - 1$ phases. Since we eliminate a single arm at each phase of the algorithm, the probability of the best arm being eliminated at phase i is bounded by

$$Pr[\hat{p}_1 < \hat{p}_2, \hat{p}_1 < \hat{p}_3, ..., \hat{p}_1 < \hat{p}_{n-i}] \leq Pr[\hat{p}_1 < \hat{p}_{n-i}]$$

The probability that $\hat{p}_1 < \hat{p}_{n-i}$, after sampling each arm for $O(1/\Delta_i^2 \log(n/\delta))$ times, is bounded by $\frac{\delta}{n}$. Therefore, the total probability of failure is bounded by δ. The arm sample complexity is the sum over t_k from for $k = 2$ to $k = n$ which leads to Eq. (3.1). □

Next, we relax the requirement that the expected rewards of the arms are known in advance, and introduce the Successive Elimination algorithm that works with any set of biases. The algorithm we present as Algorithm 3 finds the best arm (rather than ε-best) with high probability. We later explain in Remark 3.5 how to modify it to be an (ε, δ)-PAC algorithm.

Input : $\delta > 0$
Output: An arm
Set $t = 1$ and $S = A$;
Set for every arm a: $\hat{p}_a^1 = 0$;
Sample every arm $a \in S$ once and let $\hat{p}_a^t$ be the average reward of arm a by time t;
repeat
    Let $\hat{p}_{max}^t = \max_{a \in S} \hat{p}_a^t$ and $\alpha_t = \sqrt{\ln(cnt^2/\delta)/t}$, where c is a constant;
    foreach arm $a \in S$ such that $\hat{p}_{max}^t - \hat{p}_a^t \geq 2\alpha_t$ **do**
        set $S = S \setminus a$;
    end
    $t = t + 1$;
until $|S| > 1$;

Algorithm 3: Successive elimination with unknown biases

Theorem 3.3. *Suppose that $\Delta_i > 0$ for $i = 2, 3, \ldots, n$. Then the Successive Elimination algorithm (Algorithm 3) is a $(0, \delta)$-PAC algorithm, and with probability at least $1 - \delta$ the number of samples is bounded by*

$$O\left(\sum_{i=2}^{n} \frac{\ln(\frac{n}{\delta \Delta_i})}{\Delta_i^2}\right).$$

Proof. Our main argument is that, at any time t and for any action a, the observed probability $\hat{p}_a^t$ is within α_t of the true probability p_a. For any time t and action $a \in S_t$ we have that,

$$\Pr[|\hat{p}_a^t - p_a| \geq \alpha_t] \leq 2e^{-\alpha_t^2 t} \leq \frac{2\delta}{cnt^2}.$$

By taking the constant c to be greater than 4 and from the union bound we have that with probability at least $1 - \delta/n$ for any time t and any action $a \in S_t$, $|\hat{p}_a^t - p_a| \leq \alpha_t$. Therefore, with probability $1 - \delta$, the best arm is never eliminated. Furthermore, since α_t goes to zero as t increases, eventually every non-best arm is eliminated. This completes the proof that the algorithm is $(0, \delta)$-PAC.

It remains to compute the arm sample complexity. To eliminate a non-best arm a_i we need to reach a time t_i such that,

$$\hat{\Delta}_{t_i} = \hat{p}^{t_i}_{a_1} - \hat{p}^{t_i}_{a_i} \geq 2\alpha_{t_i}$$

The definition of α_t combined with the assumption that $|\hat{p}^t_a - p_a| \leq \alpha_t$ yields that

$$\Delta_i - 2\alpha_t = (p_1 - \alpha_t) - (p_i + \alpha_t) \geq \hat{p}_1 - \hat{p}_i \geq 2\alpha_t,$$

which holds with probability at least $1 - \frac{\delta}{n}$ for

$$t_i = O\left(\frac{\ln(n/\delta\Delta_i)}{\Delta_i^2}\right).$$

To conclude, with probability of at least $1-\delta$ the number of arm samples is $t_2+\sum_{i=3}^n t_i$, which completes the proof. □

Remark 3.4. We can improve the dependence on the parameter Δ_i if at the t-th phase we sample each action in S_t for 2^t times rather than once and take $\alpha_t = \sqrt{\ln(cn\ln(t)/\delta)/t}$. This will give us a bound on the number of samples with a dependency of

$$O\left(\sum_{i=2}^{n} \frac{\log\left(-\frac{n}{\delta}\log(\Delta_i)\right)}{\Delta_i^2}\right).$$

Remark 3.5. One can easily modify the successive elimination algorithm so that it is (ε, δ)-PAC. Instead of stopping when only one arm survives the elimination, it is possible to settle for stopping when either only one arm remains or when each of the k surviving arms were sampled $O(\frac{1}{\varepsilon^2}\log(\frac{k}{\delta}))$. In the latter case the algorithm returns the best arm so far. In this case it is not hard to show that the algorithm finds an ε-optimal arm with probability at least $1 - \delta$ after

$$O\left(\sum_{i:\Delta_i>\varepsilon} \frac{\log(\frac{n}{\delta\Delta_i})}{\Delta_i^2} + \frac{N(\Delta,\varepsilon)}{\varepsilon^2}\log\left(\frac{N(\Delta,\varepsilon)}{\delta}\right)\right),$$

where $N(\Delta, \varepsilon) = |\{i \mid \Delta_i < \varepsilon\}|$ is the number of arms which are ε-optimal.

3.2 Median elimination

The following algorithm substitutes the term $O(\log(1/\delta))$ for $O(\log(n/\delta))$ of the naive bound. The idea is to eliminate the worst half of the arms at each iteration. We do not expect the best arm to be empirically "the best", we only expect an ε-optimal arm to be above the median.

Input : $\varepsilon > 0, \delta > 0$
Output: An arm
Set $S = A$, $\epsilon_1 = \epsilon/4$, $\delta_1 = \delta/2$, $\ell = 1$. **repeat**
    Sample every arm $a \in S$ for $1/(\epsilon_\ell/2)^2 \log(3/\delta_\ell)$ times, and let $\hat{p}_a^\ell$ denote its empirical value;
    Find the median of $\hat{p}_a^\ell$, denoted by m_ℓ;
    $S_{\ell+1} = S_\ell \setminus \{a : \hat{p}_a^\ell < m_\ell\}$;
    $\epsilon_{\ell+1} = \frac{3}{4}\epsilon_\ell$; $\delta_{\ell+1} = \delta_\ell/2$; $\ell = \ell + 1$;
until $|S| = 1$;

Algorithm 4: Median Elimination

Theorem 3.6. *The Median Elimination(ϵ, δ) algorithm is an (ϵ, δ)-PAC algorithm and its sample complexity is*

$$O\left(\frac{n}{\epsilon^2}\log\left(\frac{1}{\delta}\right)\right).$$

First we show that in the ℓ-th phase the expected reward of the best arm in S_ℓ drops by at most ϵ_ℓ.

Lemma 3.7. *For the* Median Elimination(ϵ, δ) *algorithm we have that that for every phase ℓ:*

$$\Pr[\max_{j \in S_\ell} p_j \le \max_{i \in S_{\ell+1}} p_i + \epsilon_\ell] \ge 1 - \delta_\ell.$$

Proof. Without loss of generality we look at the first round and assume that p_1 is the reward of the best arm. We bound the failure probability by looking at the event $E_1 = \{\hat{p}_1 < p_1 - \epsilon_1/2\}$, which is the case that the empirical estimate of the best arm is pessimistic. By step 3 of the algorithm, we sample sufficiently such that $\Pr[E_1] \leq \delta_1/3$.

In case E_1 does not hold, we calculate the probability that an arm j which is not an ϵ_1-optimal arm is empirically better than the best arm.

$$\Pr\left[\hat{p}_j \geq \hat{p}_1 \mid \hat{p}_1 \geq p_1 - \epsilon_1/2\right] \leq \Pr\left[\hat{p}_j \geq p_j + \epsilon_1/2 \mid \hat{p}_1 \geq p_1 - \epsilon_1/2\right] \leq \delta_1/3$$

Let #bad be the number of arms which are not ϵ_1-optimal but are empirically better than the best arm. We have that $\mathbf{E}[\#\text{bad} \mid \hat{p}_1 \geq p_1 - \epsilon_1/2] \leq n\delta_1/3$. Next we apply Markov inequality to obtain,

$$\Pr[\#\text{bad} \geq n/2 \mid \hat{p}_1 \geq p_1 - \epsilon_1/2] \leq \frac{n\delta_1/3}{n/2} = 2\delta_1/3.$$

Using the union bound gives us that the probability of failure is bounded by δ_1. □

Next we prove that arm sample complexity is bounded by $O((n/\varepsilon^2)\log(1/\delta))$.

Lemma 3.8. *The sample complexity of the* Median Elimination(ϵ, δ) *is* $O(\frac{n\log(\frac{1}{\delta})}{\epsilon^2})$.

Proof. The number of arm samples in the ℓ-th round is $\frac{4n_\ell \log(3/\delta_\ell)}{\epsilon_\ell^2}$. By definition we have that

1. $\delta_1 = \delta/2$; $\delta_\ell = \delta_{\ell-1}/2 = \delta/2^\ell$

2. $n_1 = n$; $n_\ell = n_{\ell-1}/2 = n/2^{\ell-1}$

3. $\epsilon_1 = \epsilon/4$; $\epsilon_\ell = \frac{3}{4}\epsilon_{\ell-1} = \left(\frac{3}{4}\right)^{\ell-1}\epsilon/4$

Therefore we have

$$\begin{aligned}
\sum_{\ell=1}^{\log_2(n)} \frac{n_\ell \log(3/\delta_\ell)}{(\epsilon_\ell/2)^2} &= 4 \sum_{\ell=1}^{\log_2(n)} \frac{n/2^{\ell-1} \log(2^\ell 3/\delta)}{((\frac{3}{4})^{\ell-1}\epsilon/4)^2} \\
&= 64 \sum_{\ell=1}^{\log_2(n)} n(\frac{8}{9})^{\ell-1}(\frac{\log(1/\delta)}{\epsilon^2} + \frac{\log(3)}{\epsilon^2} + \frac{\ell \log(2)}{\epsilon^2}) \\
&\leq 64 \frac{n \log(1/\delta)}{\varepsilon^2} \sum_{\ell=1}^{\infty}(\frac{8}{9})^{\ell-1}(\ell C' + C) = O(\frac{n \log(1/\delta)}{\epsilon^2})
\end{aligned}$$

□

Now we can prove Theorem 3.6.

Proof. From Lemma 3.8 we have that the sample complexity is bounded by $O(\frac{n\log(1/\delta)}{\epsilon^2})$. By Lemma 3.7 we have that the algorithm fails with probability δ_i in each round so that over all rounds the probability of failure is bounded by $\sum_{i=1}^{\log_2(n)} \delta_i \leq \delta$. In each round we reduce the optimal reward of the surviving arms by at most ϵ_i so that the total error is bounded by $\sum_{i=1}^{\log_2(n)} \epsilon_i \leq \epsilon$. □

Chapter 4

Phased Learning

This chapter is based on [31]. We analyze the batched sampling setting of [58] and provide a mechanism that can use any (ε, δ)-MAB algorithm to enhance the performance of the Phased Q-learning introduced in [58]. We also provide a matching lower bound. In this limited setup we can fully exploits the connection between the MAB problem and learning in MDPs. The setup is that of parallel sampling where the decision maker can sample every state and action pair, as opposed to the typical Q-learning setup where a single trajectory is followed. We will focus on the phased Q-learning algorithm [58] which partitions the learning to phases. We will use a MAB black-box to perform the updates for each state and phase of the phased Q-learning algorithm.

In phased Q-learning the value of $V_k(s)$ is fixed during the kth phased and updated only at the end of the phase. This implies that for every state and action (s, a) we can define a random variable $Y_s(a)$ whose value is $R(s, a) + \gamma V_k(s')$, where $R(s, a)$ is the random variable representing the reward and s' is distributed using $P^a_{s,s'}$.

Our aim is to find, at each state, the action that maximizes the expected reward, and estimate its expected reward, where the rewards are $Y_s(a)$. The phased Q-Learning can now be viewed as using the naive algorithm for the MAB problem (Algorithm 1) in order to find the best arm. In the following we show how, using more sophisticated MAB algorithms, one can improve the convergence rate of the Phased Q-Learning.

Our algorithm uses any (ϵ, δ)-PAC Multi-armed bandit algorithm as a black box in the learning process. In order to use the MAB algorithm B as, a black box, we define a simple interface, which requires the following procedures:

- $Init_B(\epsilon, \delta)$ - Initialize the parameters of B.
- $GetArm_B()$ - returns the arm a that B wants to sample next.
- $Update_B(a, r)$ - informs B the latest reward r of arm a.
- $Stop_B(a, v)$ - returns TRUE if B terminates, and in such a case a is the output of B and v is its estimated value. (We assume that on termination, with probability at least $1 - \delta$, the arm a is an ε-optimal arm and $|r^* - v| \leq \varepsilon$.)

The MAB Phased Q-learning algorithm uses, as a black box, an algorithm B for the MAB problem. It receives as input (ε, δ) and returns a policy π which is ε-optimal with probability at least $1 - \delta$.

Input : $\varepsilon, \delta > 0$ and B a multi-armed bandit algorithm
Output: A policy
Let $\hat{\epsilon} = \frac{\epsilon(1-\gamma)}{2}$; $n = \log_\gamma(\hat{\varepsilon}/2V_{max}) = O(\ln(\frac{V_{max}}{(1-\gamma)\varepsilon})/(1-\gamma))$; $\hat{\delta} = \frac{\delta}{|S|n}$;
Initialize for every $s \in S$: $V_0(s) = 0$;
for $i = 1 : n$ **do**
  foreach $s \in S$ **do**
    $Init_B(\hat{\epsilon}, \hat{\delta})$;
    repeat
      $a = GetArm_B()$;
      $(s', r) = sample(s, a)$;
      $r' = r + \gamma V_i(s')$;
      $Update_B(a, r')$;
    until $Stop(a, v) = TRUE$;
    $V_{i+1}(s) = v$; $\pi(s) = a$;
  end
end

Algorithm 5: MAB Phased Q-Learning algorithm

Suppose that we have some (ϵ, δ)-PAC MAB algorithm B, and assume B has arm sample complexity $T_B(\epsilon, \delta)$. Namely, with probability $1 - \delta$, algorithm B terminates after at most $T_B(\epsilon, \delta)$ and outputs a policy π which is ϵ-optimal. The following theorem computes the sample complexity of MAB Phased Q-Learning algorithm as a function of T_B.

Theorem 4.1. *Assume B is an $(\hat{\epsilon}, \hat{\delta})$-PAC multi-armed bandit algorithm. Then the MAB Phased Q-Learning(ϵ, δ) algorithm outputs a policy π which is ϵ-optimal policy with probability at least $1-\delta$, and has sample complexity of*

$$T(\varepsilon, \delta) = |S| T_B(\hat{\epsilon}, \hat{\delta}) \log_{\gamma}(\frac{\hat{\epsilon}}{2V_{max}}) = O\left(\frac{|S|}{1-\gamma} \ln(\frac{V_{max}}{(1-\gamma)\epsilon}) T_B\left(\frac{\epsilon(1-\gamma)}{2}, \frac{\delta(1-\gamma)}{|S|\ln(V_{max}/\epsilon)}\right)\right).$$

First we show that in each phase the norm $\|V^* - V\|_\infty$ decreases.

Lemma 4.2. *Assume B is an $(\hat{\epsilon}, \hat{\delta})$-PAC multi-armed bandit algorithm, and consider the MAB Phased Q-Learning(ϵ, δ) algorithm using B. Then with probability at least $1-\delta$, for all $k \le n$, $\|V^* - V_k\|_\infty$ is bounded by $\frac{\hat{\epsilon}}{1-\gamma} + V_{max}\gamma^k$.*

Proof. First we bound the probability that B outputs an arm which is not ϵ-optimal. We bound the failure probability by using the union bound on all the invocations of B. There are

$$|S|n = |S| \log_{\gamma}(\hat{\epsilon}/V_{max}) = O\left(\frac{|S| \ln(\frac{V_{max}}{(1-\gamma)\epsilon})}{1-\gamma}\right)$$

initializations of algorithm B and for each invocation the failure probability is bounded by $\delta/|S|n$. Thus, the failure probability is at most δ.

Next, we show that the error contracts in every phase. We compare the value vector, V_k, with the standard value iteration value vector $\hat{V}_k$ for the case of a known model (at the end of the k-th step). Formally,

$$\hat{V}_{k+1}(s) = \max_u \{\mathbf{E}[R(s,u)] + \gamma \mathbf{E}_{s'}[\hat{V}_k(s')]\},$$

where s' is distributed according to $P^u_{s,s'}$ and $\hat{V}_0 = 0$.

We show by induction on the number of phases, that $d_k = \|V_k - \hat{V}_k\|_\infty \le \frac{\hat{\epsilon}}{1-\gamma}$. The base of the induction, $t = 0$, for every state s we have $d_0 = |V_0(s) - \hat{V}_0(s)| = 0$. We assume that the induction assumption holds for $t < k$ and prove for k. Let

$m_{s,a}$ denote the number of times the state action pair (s, a) was sampled in the k-th iteration.

$$\begin{aligned}
|V_k(s) - \hat{V}_k(s)| &= \Big| \max_u [\frac{1}{m_{s,u}} \sum_{i=1}^{m_{s,u}} r(s,u) + \gamma V_{k-1}(s'_i)] \\
&\quad - \max_a [\mathbf{E}[R(s,a)] + \gamma \sum_{s'} P^a_{s,s'} \hat{V}_{k-1}(s')] \Big| \\
&\leq \max_{\rho \in \{-\hat{\epsilon}, \hat{\epsilon}\}} \Big| \max_u [\mathbf{E}[R(s,u)] + \gamma \sum_{s'} P^u_{s,s'} V_{k-1}(s')] + \rho \\
&\quad - \max_a [\mathbf{E}[R(s,a)] + \gamma \sum_{s'} P^a_{s,s'} \hat{V}_{k-1}(s')] \Big| \\
&\leq \hat{\epsilon} + \max_a \Big| \gamma \sum_{s'} P^a_{s,s'} (V_{k-1}(s') - \hat{V}_{k-1}(s')) \Big| \\
&\leq \hat{\epsilon} + \gamma d_{k-1} \\
&\leq \hat{\epsilon} + \gamma (\frac{\hat{\epsilon}}{1-\gamma}) = \frac{\hat{\epsilon}}{1-\gamma}.
\end{aligned}$$

To conclude the proof note that for the value iteration we have that $\|\hat{V}_k - V^*\|_\infty \leq \gamma^k V_{max}$, where $\hat{V}_0 = 0$ (see, e.g. [13]). □

Lemma 4.3. *When the MAB Phased Q-Learning algorithm terminates, the policy π it returns is ϵ-optimal with probability at least $1 - \delta$.*

Proof. By Lemma 4.2 we have that with probability at least $1 - \delta$ the difference $\|V_k - V^*\|_\infty \leq \frac{\hat{\epsilon}}{1-\gamma} + V_{max}\gamma^k$. Since $\hat{\epsilon} = \epsilon(1-\gamma)/2$, we have that $\|V_k - V^*\|_\infty \leq \epsilon/2 + V_{max}\gamma^k$. The lemma follows from our choice of $n = \log_\gamma(\hat{\varepsilon}/2V_{max})$. □

We can now complete the proof Theorem 4.1.

Proof. The correctness follows from Lemma 4.3. We bound the sample complexity as follows. By definition, the MAB Phased Q-Learning algorithm samples at each state and action during every phase $T_B(\hat{\epsilon}, \hat{\delta})$. By definition of the algorithm, the number of phases is $n = O\left(\ln(V_{max}/\hat{\epsilon})/(1-\gamma)\right)$, and each phase is composed from $|S|$ MAB instances. This completes the bound on the sample complexity. □

Applying the multi-armed bandit algorithms described in the previous sections we derive the following corollary. We show that by using the *median elimination* algorithm, the arm sample complexity can be reduced by a factor of $\log(|A|)$.

Corollary 4.4. *Let B be the median elimination algorithm. MAB Phased Q-Learning algorithm has sample complexity*

$$T(\varepsilon, \delta) = O\left(\frac{|S|\;|A| V_{max}^2}{(1-\gamma)^3 \epsilon^2} \ln(\frac{V_{max}}{(1-\gamma)\epsilon}) \ln(\frac{|S| \ln(V_{max}/\epsilon)}{\delta(1-\gamma)}) \right).$$

Next we introduce an almost matching lower bound. Let us introduce some more notation before we proceed. Let T denote the time until an RL algorithm stops (this may be in general a random number). For a given RL algorithm L and a given MDP L we denote by $\mathbf{E}^{L,M}$ the expectation with respect to randomization in both the algorithm and the MDP.

Theorem 4.5. *Let L be a learning algorithm for MDPs under the parallel sampling model. There are constants $C_1, C_2, \varepsilon_0, \delta_0, \gamma_0$ such that for every $\varepsilon \in (0, \varepsilon_0)$, $\delta \in (0, \delta_0)$, and $\gamma \in (0, \gamma_0)$ if L returns an ε-optimal policy with probability of at least $1 - \delta$ for every MDP with discount factor γ there exist an MDP M for which:*

$$\mathbf{E}^{L,M}[T] \geq C_1 \frac{|S|\;|A|}{(1-\gamma)^2\epsilon^2} \log\left(\frac{C_2}{\delta}\right) = \Omega\left(\frac{|S|\;|A|}{(1-\gamma)^2\epsilon^2}\log(\frac{1}{\delta})\right).$$

Proof. Consider the following construction which was used in Theorem 1 of [73] for the MAB problem. A MAB problem with $|A|$ arms is given. We enumerate the arms from 0 to $|A|-1$, and fix $\hat{\varepsilon} > 0$. In each of the states one of the following hypotheses is true:

$$H_0 : r(0) = 1/2 + \hat{\varepsilon}\,; \qquad r(i) = 1/2 \quad (i = 1, 2, \ldots, |A-1|),$$

and for $\ell = 1, 2, \ldots, |A| - 1$:

$$H_\ell : r(0) = 1/2 + \hat{\varepsilon}\,; \qquad r(i) = 1/2 \quad (i = 1, 2, \ldots, |A-1|, i \neq \ell)\,; \quad r(\ell) = 1/2 + 2\hat{\varepsilon}.$$

$\mathbf{E}_\ell$ be the expectation given than hypothesis H_ℓ is true, and $A(s)$ the event that the algorithm errs in state s. In Lemma 4 in [73] it was proved that there are constants

c_1 and c_2 such that for $\hat{\varepsilon} < \hat{\varepsilon}_0$ every algorithm that can identify the true hypothesis with probability $1 - \hat{\delta}$ for all the hypotheses must satisfy that:

$$\mathbf{E}_0[T] \geq c_1 \frac{|A| - 1}{\hat{\varepsilon}^2} \log(\frac{c_2}{\hat{\delta}}). \tag{4.1}$$

We create the following set of MDPs. In each possible MDP there are $|S|$ states and $|A|$ actions in each state. All the states are absorbing and have one of the above A hypotheses per state. The reward in each state behaves according to H_0 or one of the H_ℓ. (There are $|A|^{|S|}$ possible MDPs.) We set $\hat{\varepsilon} = 2(1 - \gamma)\varepsilon$. We run algorithm L until termination. When L terminates it returns a policy which is ε-optimal with probability of at least $1 - \delta$. Since every state is absorbing, and by our choice of $\hat{\varepsilon}$, it implies that the right hypothesis was found in all states. Note that even if L returns a randomized policy, we will determine that the action with the highest reward is the best one (this is the reason for the factor of 2 in determining $\hat{\varepsilon}$). By taking the sum of Eq. (4.1) over all states we obtain that

$$\mathbf{E}^{L,M}[T] \geq c_1 \frac{|A| - 1}{\hat{\varepsilon}^2} |S| \log(\frac{c_2}{\delta}).$$

The result follows by an appropriate choice of constants. $\square$

Chapter 5

Action elimination in MDPs

This chapter is based on [32]. In this chapter we consider algorithms for the RL problem, which are based on the MAB algorithms presented in Chapter 3. We start from model-based learning in Section 5.1, where the parameters of the models are learned. We describe algorithms which are based on the successive elimination algorithm and provide both action elimination and stopping conditions for this algorithm. In Section 5.2 we consider model-free learning and suggest a version of the Q-learning algorithm that incorporates action elimination and stopping conditions.

5.1 Model-Based Learning

In this section we focus on model-based learning. In model-based methods, we first learn the model, i.e., estimate the immediate reward and the next state distribution. Then by either value iteration, policy iteration, or linear programming on the learned (empirical) model, we find the exact optimal policy for the empirical model. If enough exploration is done, this policy is an almost optimal policy for the real model. We note that there is an inherent difference between the finite horizon and the infinite discounted return. Technically, the finite horizon return is simpler than the discounted return, as one can apply the concentration inequality directly. We provide model-based algorithms for both cases.

5.1.1 Finite Horizon

Let us first recall the classical optimality equations for finite horizon:

$$\begin{aligned} V^H(s) &= \max_a \{R(s,a) + \mathbf{E}_{s'}[V^{H-1}(s')]\}, \quad H > 0 \\ V^0(s) &= \max_a R(s,a), \end{aligned}$$

where $V^H(s)$ is the optimal value function for horizon H. Given the empirical model by time t we define the upper estimate $\overline{V}_\delta$, which will be shown to satisfy for every horizon k and every state s, $\overline{V}^k_\delta(s) \geq V^k(s)$ with high probability. For horizon H we define:

$$\overline{V}^H_\delta(s) = \max_a \left\{ \hat{R}(s,a) + \hat{\mathbf{E}}_{s'}[\overline{V}^{H-1}_\delta(s')] + HR_{max}\sqrt{\frac{\ln\left(\frac{c|S||A|H^2}{\delta}\right)}{|T^{s,a}|}} \right\}, \quad H > 0 \tag{5.1}$$

$$\overline{V}^0_\delta(s) = \max_a \left\{ \hat{R}(s,a) + R_{max}\sqrt{\frac{\ln\left(\frac{c|S||A|}{\delta}\right)}{|T^{s,a}|}} \right\}, \tag{5.2}$$

for some constant $c \geq 4$. Similarly to the upper bound $\overline{V}^H_\delta$, a lower bound may be defined where the R_{max} is replaces by $-R_{max}$. We call this estimate the lower estimate $\underline{V}^H_\delta$. The following Lemma proves that $\overline{V}^H_\delta$ is an upper estimation for any horizon and that $\underline{V}^H_\delta$ is a lower estimation.

Theorem 5.1. *We have that $\overline{V}^k_\delta(s) \geq V^k(s) \geq \underline{V}^k_\delta(s)$ for all states s and horizons k, w with probability at least $1-\delta$.*

Proof. We prove the claim by induction. For the base of the induction, by a simple use of Hoeffding inequality we have that for every state s $\overline{V}^0_\delta(s) \geq \max_a \hat{R}(s,a)$ with probability $1-\delta/(c|S||A|)$. Next we assume that the claim holds for $i \leq k$ and prove for $k+1$ and for every action a. By definition $\overline{V}^{k+1}_\delta(s)$ satisfies for every a that

$$\begin{aligned} \overline{V}^{k+1}_\delta(s) &\geq \hat{R}(s,a) + \hat{\mathbf{E}}_{s'}[\overline{V}^k_\delta(s')] + (k+1)R_{max}\sqrt{\frac{\ln\left(\frac{c|S||A|(k+1)^2}{\delta}\right)}{|T^{s,a}|}} \\ &\geq \hat{R}(s,a) + \hat{\mathbf{E}}_{s'}[V^k(s')] + (k+1)R_{max}\sqrt{\frac{\ln\left(\frac{c|S||A|(k+1)^2}{\delta}\right)}{|T^{s,a}|}}, \end{aligned}$$

where the second inequality follows from the inductive hypothesis. Note that V^k is not a random variable, so we can bound the last expression using Hoeffding's inequality. We arrive at:

$$\Pr\left\{\hat{R}(s,a)\hat{\mathbf{E}}_{s'}[V^k(s')] + (k+1)R_{max}\sqrt{\frac{\ln\left(\frac{c|S||A|(k+1)^2}{\delta}\right)}{|T^{s,a}|}} < R(s,a) + \mathbf{E}_{s'}[V^k(s')]\right\}$$

$$\leq e^{\frac{-\ln\left(\frac{c|S||A|(k+1)^2}{\delta}\right)|T^{s,a}|\left(\frac{(k+1)R_{max}}{\sqrt{|T^{s,a}|}}\right)^2}{((k+1)R_{max})^2}} = \frac{\delta}{c|S||A|(k+1)^2}.$$

Therefore, we have that with high probability the following holds

$$\begin{aligned}
\overline{V}_{\delta}^{k+1}(s) &\geq R(s,a) + \hat{\mathbf{E}}_{s'}[V^k(s')] + kR_{max}\sqrt{\frac{\ln\left(\frac{c|S||A|k^2}{\delta}\right)}{|T^{s,a}|}} \\
&\geq R(s,a) + \mathbf{E}_{s'}[V^k(s')] \\
&= V^{k+1}(s').
\end{aligned}$$

Using the union bound over all state-action pairs and all finite horizons k, we obtain that the failure probability is bounded by $\delta/2$ for $c \geq 4$. Repeating the same argument for the lower estimate and applying the union bound completes the proof. □

Consequently, a natural early stopping condition is to stop sampling when $\|\overline{V}^H - \underline{V}^H\|_\infty < \varepsilon$. We do not provide here an algorithm, however a detailed algorithm will be given in the following subsection.

5.1.2 Discounted Return - Infinite Horizon

In this subsection, we provide upper and lower estimates of the value function V for the infinite horizon case. The optimal value is the solution of the set of the equations:

$$V^*(s) = \max_a\{R(s,a) + \gamma\mathbf{E}_{s'}[V^*(s')]\}, \quad s \in S.$$

As in Subsection 5.1.1, we provide an upper value function $\overline{V}_\delta$, which satisfies with high probability $\overline{V}_\delta(s) \geq V^*(s)$. We define $\overline{V}_\delta$ as the solution of the set of equations:

$$\overline{V}_\delta(s) = \max_a \left\{ \hat{R}(s,a) + \gamma \hat{\mathbf{E}}_{s'}[\overline{V}_\delta(s')] + V_{max}\sqrt{\frac{\ln(\frac{2|S|\,|A|}{\delta})}{|T^{s,a}|)}} \right\}$$

and $\overline{Q}_\delta$ as:

$$\overline{Q}_\delta(s,a) = \hat{R}(s,a) + \gamma \hat{\mathbf{E}}_{s'}[\overline{V}_\delta(s')] + V_{max}\sqrt{\frac{\ln(\frac{2|S|\,|A|}{\delta})}{|T^{s,a}|}}.$$

Similarly, we define $\underline{V}_\delta$ and $\underline{Q}_\delta$ as:

$$\underline{V}_\delta(s) = \max_a \left\{ \hat{R}(s,a) + \gamma \hat{\mathbf{E}}_{s'}\underline{V}_\delta(s') - V_{max}\sqrt{\frac{\ln(\frac{2|S|\,|A|}{\delta})}{|T^{s,a}|)}} \right\}$$

$$\underline{Q}_\delta(s,a) = \hat{R}(s,a) + \gamma \hat{\mathbf{E}}_{s'}[\underline{V}_\delta(s')] - V_{max}\sqrt{\frac{\ln(\frac{2|S|\,|A|}{\delta})}{|T^{s,a}|}}.$$

The next lemma shows that with high probability the upper and lower estimations are indeed correct.

Lemma 5.2. *With probability at least $1-\delta$ we have that $\overline{Q}_\delta(s,a) \geq Q^*(s,a) \geq \underline{Q}_\delta(s,a)$ for every state s and action a.*

Proof. Suppose we run a value iteration algorithm on the empirical model. Let $\overline{V}_\delta^k$ be the kth iteration of the value function algorithm, and let $\overline{Q}_\delta^k$ be the associated Q-function, that is

$$\overline{Q}_\delta^k(s,a) = \hat{R}(s,a) + \gamma \hat{\mathbf{E}}_{s'}[\overline{V}_\delta^k(s')] + V_{max}\sqrt{\frac{\ln(\frac{2|S||A|}{\delta})}{T^{s,a}}}.$$

Assume that we start with $\overline{V}_\delta^0 = V^*$. (The use of V^* is restricted to the proof and not used in the algorithm.) We need to prove that $\overline{Q}_\delta(s,a) \geq Q^*(s,a)$ for every s and a. Note that since the value iteration converges, $\overline{Q}_\delta^k$ converges to $\overline{Q}_\delta$. We prove by induction on the number of the iterations that by taking $\overline{V}_\delta^0 = V^*$, with high probability for every k we have that $\overline{Q}_\delta^k \geq \overline{Q}_\delta^{k-1}$, i.e., $\Pr[\forall k\ \overline{Q}_\delta^k \geq \overline{Q}_\delta^{k-1}] \geq 1-\delta/2$.

For the basis, since V^* is not a random variable we can apply Hoeffding's inequality and obtain that for every state action pair (s,a)

$$\Pr\left\{\hat{R}(s,a)+\gamma\hat{\mathbf{E}}_{s'}[V^*(s')]+V_{max}\sqrt{\frac{\ln(\frac{2|S||A|}{\delta})}{|T^{s,a}|}}<R(s,a)+\gamma\mathbf{E}_{s'}[V^*(s')]\right\}$$
$$\leq e^{-\ln(\frac{2|S||A|}{\delta})}=\frac{\delta}{2|S||A|}\,.$$

Since $\overline{V}^0_\delta(s)=V^*$ we have that $\overline{Q}^1_\delta(s,a)=\hat{R}(s,a)+\gamma\hat{\mathbf{E}}_{s'}[\overline{V}^0_\delta(s')]+V_{max}\sqrt{\frac{\ln(\frac{2|S||A|}{\delta})}{|T^{s,a}|}}$. Therefore, $\overline{Q}^1_\delta\geq\overline{Q}^0_\delta$ with probability $1-\delta/2$. For the induction step, we assume that the claim holds for $i<k$ and prove for k.

$$\overline{Q}^k_\delta(s,a)-\overline{Q}^{k-1}_\delta(s,a)\;=\;\gamma\hat{\mathbf{E}}_{s'}[\overline{V}^{k-1}_\delta(s')-\overline{V}^{k-2}_\delta(s')].$$

Since $\overline{V}^{k-1}_\delta(s')=\max_a\overline{Q}^{k-1}_\delta(s',a)$ we have by the induction that for every s,

$$V^{k-1}_\delta(s)=\max_a\overline{Q}^{k-1}_\delta(s,a)\geq\max_a\overline{Q}^{k-2}_\delta(s,a)=V^{k-2}_\delta(s).$$

So that $\overline{Q}^k_\delta-\overline{Q}^{k-1}_\delta\geq 0$. We conclude that $\Pr[\overline{Q}_\delta\geq Q^*]\geq 1-\delta/2$. Repeating the same argument for the lower estimate, $\underline{Q}_\delta$, and applying the union bound completes the proof. □

The AE procedure is demonstrated in the following algorithm, which also supplies a stopping condition for sampling the model and eliminates actions when they are sub-optimal with high probability.

Input : MDP M, $\varepsilon > 0$, $\delta > 0$
Output: A policy for M
Choose arbitrarily an initial state s_0, let $t = 0$,
and let $U_0 = \{(s,a)|s \in S, a \in A\}$
repeat
  At state s_t perform any action a s.t. $(s_t, a) \in U_t$
  Receive a reward r_t, and a next state s_{t+1}
  Compute, $\overline{Q}_\delta, \underline{Q}_\delta$ from all the samples
  $t = t + 1$
  $U_t = \{(s,a)|\overline{Q}_\delta(s,a) \geq \underline{V}_\delta(s)\}$
until $\forall (s,a) \in U \quad |\overline{Q}_\delta(s,a) - \underline{Q}_\delta(s,a)| < \frac{\varepsilon(1-\gamma)}{2}$;
return $Greedy(\underline{Q}_\delta)$

Algorithm 6: Model-Based AE Algorithm

A direct corollary from Lemma 5.2, is a stopping time condition to the Model-Based algorithm using the following Corollary.

Corollary 5.3. *[[97]] If $\tilde{Q}$ is a function such that $|\tilde{Q}(s,a) - Q^*(s,a)| \leq \varepsilon$ for all $s \in S$ and $a \in A$. Then for all s*

$$V^*(s) - V^{\tilde{\pi}}(s) \leq \frac{2\varepsilon}{1-\gamma},$$

where $\tilde{\pi} = Greedy(\tilde{Q})$.

Theorem 5.4. *Supposed the Model-Based AE Algorithm terminates. Then the policy, π, the algorithm returns is ε-optimal with probability at least $1 - \delta$.*

Proof. By Lemma 5.2 we know that with probability at least $1 - \delta$ for every s and a we have that $\underline{Q}_\delta(s,a) \leq Q^*(s,a) \leq \overline{Q}_\delta(s,a)$. Therefore, with probability of at least $1 - \delta$ the optimal action has not been eliminated in any state. Furthermore, any action b in state s that has not been eliminated satisfies $Q^*(s,b) - \underline{Q}_\delta(s,b) \leq \overline{Q}_\delta(s,b) - \underline{Q}_\delta(s,b) \leq \varepsilon(1-\gamma)/2$. The result follows by Corollary 5.3. $\square$

5.2 Model-Free Learning

In this section we describe a model-free algorithm. We use two functions $\underline{Q}^t$ and $\overline{Q}^t$, which provide lower and upper estimations on Q^*, respectively. We use these functions to derive an asynchronous algorithm, which eliminates actions and supplies stopping condition. This algorithm requires space which is proportional to the space used by Q-learning and converges under the same conditions. Let us first recall the Q-learning algorithm [103]. The Q-learning algorithm estimates the state-action value function (for discounted return) as follows:

$$\begin{aligned} Q^0(s,a) &= 0, \\ Q^{t+1}(s,a) &= (1-\alpha_t(s,a))Q^t(s,a) + \alpha_t(s,a)(r_t(s,a) + \gamma V^t(s')), \end{aligned}$$

where s' is the state reached from state s when performing action a at time t, and $V^t(s) = \max_a Q^t(s,a)$. Set $\alpha_t(s,a) = 1/\#(s,a,t)$ for $t \in T^{s',a'}$ and 0 otherwise. We define the upper estimation process as:

$$\begin{aligned} \overline{Q}_\delta^0(s,a) &= V_{max} \ln(\frac{c|S||A|}{\delta}), \\ \overline{Q}_\delta^{t+1}(s,a) &= (1-\alpha_t(s,a))\overline{Q}_\delta^t(s,a) + \alpha_t(s,a)\Big(R(s,a) + \gamma \overline{V}_\delta^t(s') + \beta(\#(s,a,t))\Big), \end{aligned}$$

where $c > 4$ and s' is the state reached from state s when performing action a at time t, $\overline{V}_\delta^t(s) = \max_a \overline{Q}_\delta^t(s,a)$ and the function β, which maintains the upper estimate interval is defined as:

$$\beta(k) = \sqrt{\frac{\ln(ck^2|S||A|/\delta)}{k}}.$$

Analogously, we define the lower estimate $\underline{Q}_\delta$ as :

$$\begin{aligned} \underline{Q}_\delta^0(s,a) &= -V_{max} \ln(\frac{c|S||A|}{\delta}), \\ \underline{Q}_\delta^{t+1}(s,a) &= (1-\alpha_t(s,a))\underline{Q}_\delta^t(s,a) + \alpha_t(s,a)\Big(R(s,a) + \gamma \underline{V}_\delta^t(s') - \beta(\#(s,a,t))\Big), \end{aligned}$$

where $\underline{V}_\delta(s) = \max_a \underline{Q}_\delta(s,a)$. We claim that these processes converge almost surely to Q^*.

Proposition 5.5. *If every state-action pair is performed infinitely often then the upper (lower) estimation process, $\overline{Q}_\delta^t$ ($\underline{Q}_\delta^t$), converges to Q^* with probability one.*

Proof. In order to show the almost sure convergence of the upper and lower estimations, we follow the proof of [12]. We consider a general type of *iterative stochastic algorithms*, which is performed as follows:

$$X_{t+1}(i) = (1-\alpha_t(i))X_t(i) + \alpha_t(i)((HX_t)(i) + w_t(i) + u_t(i)),$$

where w_t is a bounded random variable with zero expectation and each H is a pseudo contraction mapping (See for detail s [12]).

Definition 5.6. An iterative stochastic algorithm is well behaved if:

1. The step size $\alpha_t(i)$ satisfies (1) $\sum_{t=0}^\infty \alpha_t(i) = \infty$, (2) $\sum_{t=0}^\infty \alpha_t^2(i) < \infty$ and (3) $\alpha_t(i) \in (0,1)$.
2. There exists a constant A that bounds $w_t(i)$ for any history F_t, i.e., $\forall t, i$: $|w_t(i)| \le A$.
3. There exists $\gamma \in [0,1)$ and a vector X^* such that for any X we have $||HX - X^*|| \le \gamma ||X - X^*||$, where $||\cdot||$ is any norm.
4. There exists a nonnegative random sequence θ_t, that converges to zero with probability 1, and is such that
$$\forall i, t \quad |u_t(i)| \le \theta_t(||X_t|| + 1)$$

We first note that the Q-learning algorithm satisfies the first three criteria and the fourth criteria holds trivially since $u_t = 0$, thus its convergence follows if all state-action pairs are tried infinitely often [see Proposition 5.6 in [12]]. The upper estimate has an additional noise term, u_t. If we show that it satisfies the fourth requirement, then the convergence will follow.

Lemma 5.7. *The upper estimation algorithm is well behaved.*

Proof. In the convergence proof of Q-learning, it was shown that requirements 1–3 are satisfied, this implies that the upper estimates satisfies them as well. Now we let $u_t = \theta_t = c\sqrt{\frac{\ln(\#(s,a,t))}{\#(s,a,t)}}V_{max}$. θ_t clearly converges to zero, thus

$$|u_t(i)| = \theta_t \leq \theta_t(||X_t|| + 1).$$

$\square$

Now the proof is completed. $\square$

The following Proposition claims that $\overline{Q}_\delta^t$ upper bounds Q^* and $\underline{Q}_\delta^t$ lower bounds Q^* with high probability.

Proposition 5.8. *With probability at least $1-\delta$ we have that for every state action pair (s,a) and time t:*

$$\overline{Q}_\delta^t(s,a) \geq Q^*(s,a) \geq \underline{Q}_\delta^t(s,a).$$

Proof. We use induction on the time t to show that for every state-action pair the following holds

$$\Pr\left\{\forall t' \leq t \;\; \overline{Q}_\delta^{t'}(s,a) < Q^*(s,a)\right\} \leq \sum_{i=1}^{\#(s,a,t)} \frac{\delta}{c|S||A|i^2}.$$

For the induction basis we have that for every state-action pair: $\overline{Q}_\delta^1(s,a) \geq Q^*(s,a)$. We assume that the claim holds for $k < t$ and prove for t. Suppose that action a is performed at state s at time t, thus for every other state-action pair the claim holds. Let t_i be the time were action a was performed at state s for the ith time. Let s_i be the next state at time $t_i + 1$ and r_i be the reward observed at that time. For (s,a)

we have that,

$$
\begin{aligned}
\overline{Q}_{\delta}^{t}(s,a) &= \frac{1}{\#(s,a,t)} \sum_{i=1}^{\#(s,a,t)} (r_i + \gamma \overline{V}_{\delta}^{t_i}(s_i) + \beta(i)) \\
&\geq \frac{1}{\#(s,a,t)} \sum_{i=1}^{\#(s,a,t)} (R(s,a) + \gamma V^*(s_i)) + \sqrt{\frac{\ln(c\#(s,a,t)^2|S||A|/\delta)V_{max}}{\#(s,a,t)}},
\end{aligned}
$$

where the last expression can be bounded with high probability using Hoeffding's inequality

$$
\Pr\left\{\frac{1}{\#(s,a,t)} \sum_{i=1}^{\#(s,a,t)} (r_i + \gamma V^*(s_i)) + \sqrt{\frac{\ln(c\#(s,a,t)^2|S||A|/\delta)V_{max}}{\#(s,a,t)}} < Q^*(s,a)\right\}
\leq \frac{\delta}{c|S||A|\#(s,a,t)^2},
$$

which completes the induction by using the union bound. Repeating the same argument for the lower estimate completes the proof. □

We combine the upper and lower estimates to an algorithm, which eliminates sub optimal actions whenever possible. Furthermore, the algorithm supplies a stopping condition that assures a near optimal policy. The model free AE algorithm is described in Algorithm 7.

A direct corollary from Proposition 5.8 is a stopping condition to the model free AE algorithm. The following corollary follows from Corollary 5.3 and its proof is similar to the proof of Theorem 5.4.

Corollary 5.9. *Suppose the Model-Free AE Algorithm terminates. Then the policy, π, it returns is ϵ-optimal with probability at least $1-\delta$.*

5.3 Experiments

In this section we show four types of MDPs in which the number of samples used by AE procedures is significantly smaller than the number of samples used by standard

Input : MDP M, $\varepsilon > 0$, $\delta > 0$
Output: A policy for M
For every state action (s, a):
$\overline{Q}(s,a) = \quad V_{max} \ln\left(\frac{c|S||A|}{\delta}\right)$
$\underline{Q}(s,a) = -V_{max} \ln\left(\frac{c|S||A|}{\delta}\right)$
#(s,a) = 1
Choose an arbitrary initial state s
repeat
  Let $U(s) = \{a | \overline{Q}(s,a) \geq \underline{V}(s)\}$
  choose arbitrarily action $a \in U(s)$, perform it and observe the next state s'
  $\overline{Q}(s,a) := (1 - \frac{1}{\#(s,a)})\overline{Q}(s,a) + \frac{1}{\#(s,a)}\Big(R(s,a) + \gamma \overline{V}(s') + \beta(\#(s,a))\Big)$
  $\underline{Q}(s,a) := (1 - \frac{1}{\#(s,a)})\underline{Q}(s,a) + \frac{1}{\#(s,a)}\Big(R(s,a) + \gamma \underline{V}(s') - \beta(\#(s,a))\Big)$
  $\#(s,a) := \#(s,a) + 1$; $s = s'$
until $\forall s \in S \ \ \forall a \in U(s) \quad |\overline{Q}(s,a) - \underline{Q}(s,a)| < \frac{\varepsilon(1-\gamma)}{2}$;
return $Greedy(\underline{Q})$

Algorithm 7: Model-Free AE Algorithm

Q-learning and ε-greedy Q-learning. Both model free AE algorithm and standard Q-learning choose the action in each state uniformly at random. In our experiments we focused on the steady state norm (L_1 weighted by steady state probabilities) rather than the L_∞ norm to emphasize the average behavior. We note that we use the steady state rather than the discounted steady state. We run AE Q-learning algorithm from Section 5.2 with the same input (for actions that were not eliminated) as a standard Q-learning algorithm. The following experiments were conducted:

1. **A queueing system.** The MDP represents a queueing problem that appears in Differentiated Services [1, 62]. The basic settings are that the arriving packets have different values and they are buffered in a FIFO queue to be sent. The major constraints are that we reject or accept a packet upon its arrival (no preemption) and that the buffer has limited capacity. We have looked on a queue of size five and three different packets values, $1, 20, 150$. In each time unit we either receive a packet or send a packet according to some distribution. We modeled the queueing problem via a discounted MDP with discount factor $\gamma =$

0.99. The AE model-free algorithm[1] was compared with ε-greedy Q-learning with epsilon varying from 0.05 to 0.2. In Figure 5.1 we present the results for ε which was empirically best, $\varepsilon = 0.1$. In this experiment we used a fixed step size. We focused here on the fraction of times in which optimal actions were performed and on the value function criterion. The results are demonstrated in Figure 5.1, in which we see that not only AE has better results but the variance in the results is much smaller in both the fraction of times that almost optimal actions were performed and in the value function. Figure 5.2 demonstrates the elimination rate of the AE procedure.

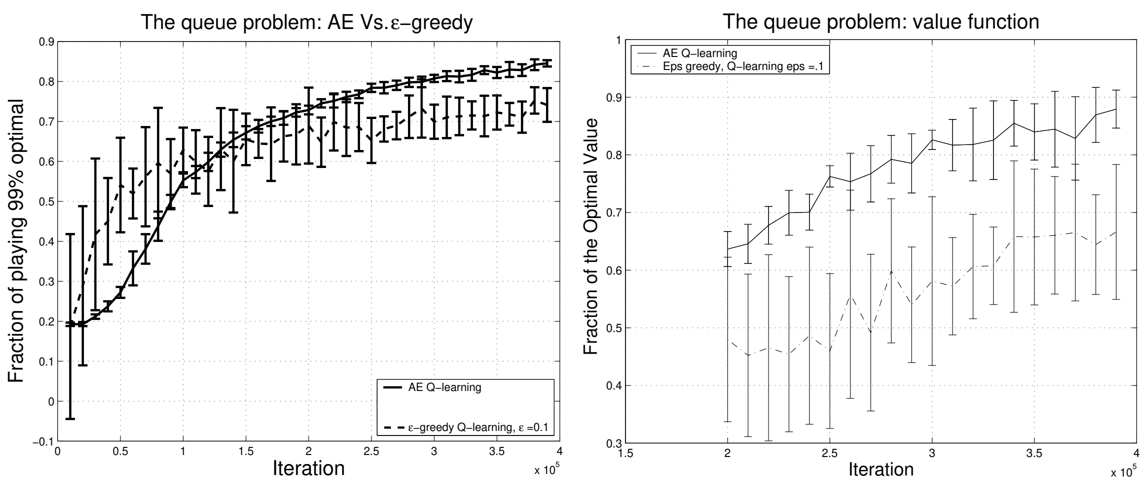


Figure 5.1: Example of a Queue of size 5 with three types of packets with values $1, 20, 150$. The discount factor is set to 0.99. We disregard the full queue state in which every action is optimal. We repeated each experiment 15 times and the error bars represent 1 standard deviation.

2. **Random MDPs.** The random MDPs are randomly generated MDPs with 20 states each and 50 actions in each state. The first one is due to [88] and is a sparse MDP, such that each action can reach only three states. The second random MDP is a dense MDP, such that the next state distribution is randomly

[1] Since we were interested in the short term results rather than the long term, we initialized the upper and lower values to similar values and allowed elimination only after an exploration period, we still used the β function for both the upper and lower estimates as stated in the theoretical part up to constants.

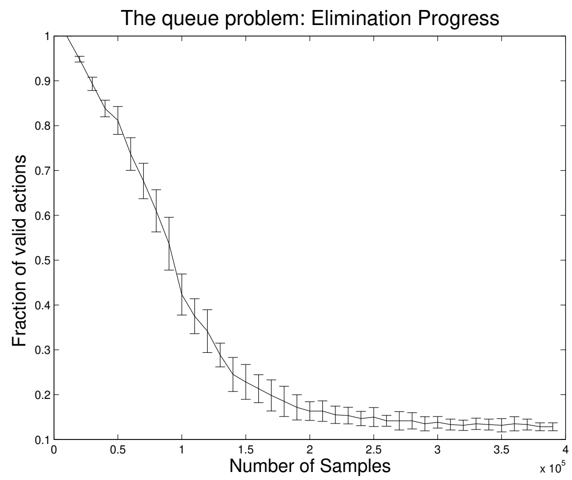


Figure 5.2: Example of a Queue of size 5 with three types of packets with values $1, 20, 150$. The discount factor is set to 0.99. This figure demonstrates the elimination rate. We repeated each experiment 15 times and the error bars represent 1 standard deviation.

chosen for each state-action pair and might include all states. For both MDPs the immediate reward expectation is randomly chosen in the interval $[0, 10]$. Results of ten runs are presented by Figure 5.3 for the sparse MDP, in this experiment the model free AE algorithm needs only about half the samples used by the Q-learning to achieve the same precision. The precision is measured as the distance of the Q-function from the optimal function in steady state norm. In Figure 5.4 for dense MDP, the results are similar. The AE algorithm required about 40% fewer samples.

3. **Howard's automobile replacement problem.** This MDP represents another realistic problem—Howard's automobile replacement problem [52]. This problem contains 40 states, in each state there are 41 actions. See [52] for a detailed description. This problem was considered as a benchmark by several authors in the optimization community. We run the model free AE algorithm for this problem with discount factor $\gamma = 0.833$ against standard Q-learning and the results appear in Figure 5.5. A significant improvement is evident.

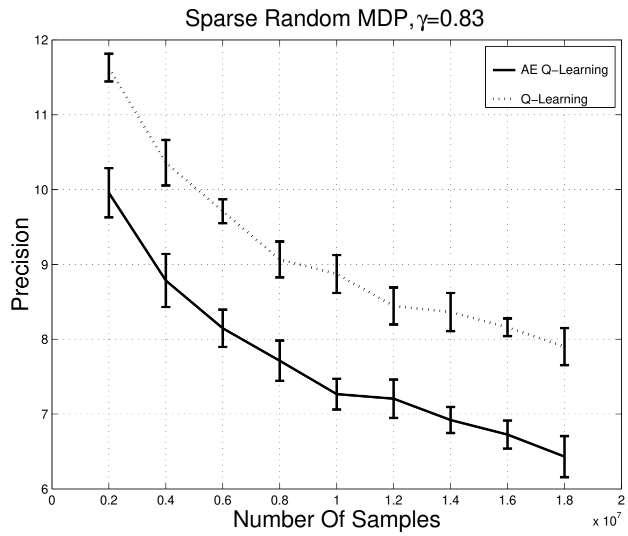


Figure 5.3: Example of a 20 state sparse randomly generated MDPs with 50 actions in each state, where $\gamma = 0.833$ (as in [88].) The precision is the distance of the Q-function from the optimal Q-function. We repeated each experiment 10 times and the error bars represent 1 standard deviation.

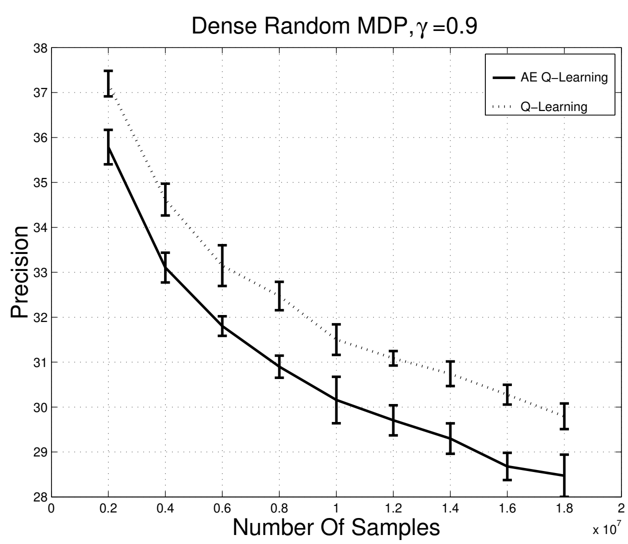


Figure 5.4: Example of a 20 state dense randomly generated MDPs with 50 actions in each state, $\gamma = 0.9$. The error bars represent 1 standard deviation.

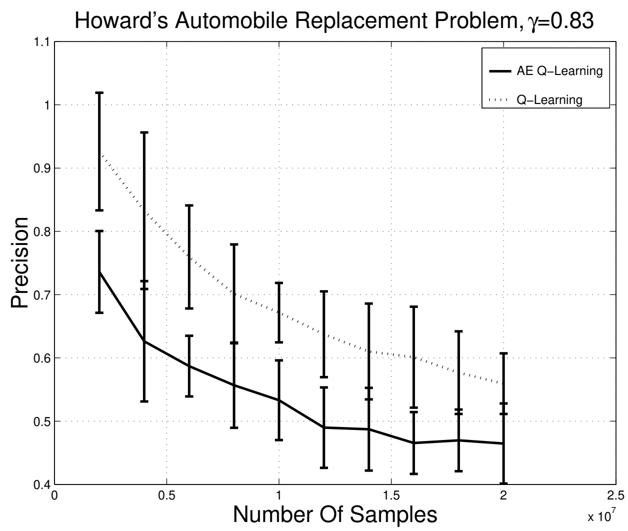


Figure 5.5: Example of Howard's Automobile Replacement Problem, where the discount factor, γ, is 0.833. The norm is the steady state norm. The error bars represent 1 standard deviation.

Chapter 6

Optimistic Q-learning

This Chapter is based on [33].

Optimistic Q-learning is a simple greedy algorithm with respect to the Q-values, where the initial Q-values are set to large values, larger than their optimal values. We show that optimistic Q-learning converges to an ϵ-optimal policy if the initial Q-values are set sufficiently large.

Let $\beta_\tau = \prod_{i=1}^{\tau}(1-\alpha_i)$. We set the initial conditions of the Q-values as follows:

$$\forall s, a: \quad Q_0(s,a) = \frac{1}{\beta_T} V_{max},$$

where $T = T(\epsilon, \delta, S, A, \vec{\alpha})$ will be specified later. Let $\eta_{i,\tau} = \alpha_i \prod_{j=i+1}^{\tau}(1-\alpha_j) = \alpha_i \beta_\tau / \beta_i$. Note that $\sum_{i=1}^{\tau} \eta_{i,\tau} = 1$ and that

$$\begin{aligned} Q_{t+1}(s,a) &= (1-\alpha_t(s,a))Q_t(s,a) + \alpha_t(s,a)(r_t + \gamma V_t(s')) \\ &= \beta_\tau Q_0(s,a) + \sum_{i=1}^{\tau} \eta_{i,\tau} r_{t_i} + \gamma \sum_{i=1}^{\tau} \eta_{i,\tau} V_{t_i}(s_i), \end{aligned}$$

where $\tau = \#(s,a,t)$, s_i is the next state arrived at time t_i when action a is performed for the ith time in state s.

First we show that as long as $\tau = \#(s,a,t) \le T$, we have $Q_t(s,a) \ge Q^*(s,a)$. Latter we will use this to show that every action is performed at least T times in every state.

Lemma 6.1. *In optimistic Q-learning for any state s, action a and time t, such that $\tau = \#(s,a,t) \le T$ we have $Q_t(s,a) \ge V_{max} \ge Q^*(s,a)$.*

Proof. Since all the rewards are non-negative and the initial Q-values are positive, then $Q_t(s,a)$ is non-negative for every time t, state s and action a. Using the following observation, we obtain the lemma

$$Q_t(s,a) = \beta_\tau Q_0(s,a) + \sum_{i=1}^{\tau} \eta_i r_i(s,a) + \gamma \sum_{i=1}^{\tau} \eta_i V_{t_i}(s_i) \ge \frac{\beta_\tau}{\beta_T} V_{max} \ge V^*(s).$$

□

Now we bound T as a function of the algorithm parameters (i.e. $\epsilon, \delta, |S|, |A|$ and the learning rate). We need to set T large enough to guarantee that with probability $1-\delta$, for any $t > T$ updates, using the given learning rate, the deviation from the true value is at most ϵ. Formally, given a sequence X_t of i.i.d. random variables with zero mean which are bounded by V_{max}, and a learning rate $\alpha_t(s,a) = (1/\#(s,a,t))^\omega$ let $Z_{t+1} = (1-\alpha_t)Z_t + \alpha_t X_t$. A time $T(\epsilon,\delta)$ is an *initialization time* if $Pr[\forall t \ge T \;\; : |Z_t| \le \epsilon] \ge 1-\delta$. The following Lemma bounds the initialization time as a function of the parameter ω of the learning rate, and it is a direct and simple corollary of Hoeffding's inequality which appears in Chapter 2

Lemma 6.2. *The initialization time for $\vec{X}$ and $\vec{\alpha}$ is at most $T(\epsilon,\delta) = c\left(\left(\frac{V_{max}^2}{\epsilon^2}(ln(1/\delta) + ln(V_{max}/\epsilon))\right)^{\frac{1}{\omega}}\right)$, for some constant c.*

Next we define a modified process, in which we update using the optimal value function, rather than our current estimate. The modified process is only used in the analysis. For $t \ge 1$ we have,

$$\hat{Q}_{t+1}(s,a) = (1-\alpha_t(s,a))\hat{Q}_t(s,a) + \alpha_t(s,a)(r_t(s,a) + \gamma V^*(s')),$$

where s' is the next state. The following lemma bounds the difference between Q^* and $\hat{Q}_t$, when $\hat{Q}$ is initialized optimistically.

Lemma 6.3. *Consider optimistic Q-learning and let $T = T(\epsilon, \delta)$ be the initialization time. Then with probability $1-\delta$, for any state action, we have $Q^*(s,a) - \hat{Q}_t(s,a) \leq \epsilon$.*

Proof. Let $\tau = \#(s,a,t)$. If $\tau \leq T$ then the $\hat{Q}^*(s,a) \geq Q^*(s,a)$ by similar arguments to those of Lemma 6.1. Next we consider the more subtle case where $\tau > T$. By definition we have

$$\hat{Q}_t(s,a) = \beta_\tau Q_0(s,a) + \sum_{i=1}^{\tau} \eta_{i,\tau} r_i + \gamma \sum_{i=1}^{\tau} \eta_{i,\tau} V^*(s_i).$$

This implies that,

$$Q^*(s,a) - \hat{Q}(s,a) = -\beta_\tau Q_0(s,a) + error\_r[s,a,t] + error\_v[s,a,t]$$

where $error\_r[s,a,t] = E[R(s,a)] - \sum_{i=1}^{\tau} \eta_{i,\tau} r_i$, and $error\_v[s,a,t] = E[V^*(s')|s,a] - \sum_{i=1}^{\tau} \eta_{i,\tau} V^*(s_i)$. We bound both $error\_r[s,a,t]$ and $error\_v[s,a,t]$ using Lemma 6.2. Therefore, with probability $1-\delta$, we have $Q^*(s,a) - \hat{Q}(s,a) \leq \epsilon$, for any $\tau \geq T$. $\square$

Next we bound the difference between our estimate $V_t(s)$ and $V^*(s)$.

Lemma 6.4. *Consider optimistic Q-learning and let $T = T((1-\gamma)\epsilon, \delta/|S||A|)$ be the initialization time. With probability at least $1-\delta$ for any state s and time t, we have $V^*(s) - V_t(s) \leq \epsilon$.*

Proof. We show by induction on t that $V^*(s) - V_t(s) \leq \epsilon$, for every state s. For $t = 0$ we have $V_0(s) > V_{max}$ and hence the claim holds. For the inductive step assume the hypothesis holds up to time t and show that it holds for time $t+1$. Let (s,a) be the state action pair executed in time $t+1$. If $\#(s,a,t+1) \leq T$ then by Lemma 6.1, $V_t(s) \geq V_{max} \geq V^*(s)$, and the induction hypothesis holds. Otherwise, let a^* be the optimal action at state s, then,

$$\begin{aligned} V^*(s) - V_{t+1}(s) &\leq Q^*(s,a^*) - Q_{t+1}(s,a^*) \\ &= Q^*(s,a^*) - \hat{Q}_{t+1}(s,a^*) + \hat{Q}_{t+1}(s,a^*) - Q_{t+1}(s,a^*) \\ &\leq (1-\gamma)\epsilon + \gamma \sum_{i=1}^{\tau} \eta_{i,\tau}(V^*(s_i) - V_{t_i}(s_i)), \end{aligned}$$

where $\tau = \#(s, a, t)$, t_i is the time when the i-th time the action a is performed in state s, and state s_i is the next state. The second inequality holds with probability at least $1 - \delta$ due to Lemma 6.3. Since $t_i \leq t$, by the inductive hypothesis we have that $V^*(s_i) - V_{t_i}(s_i) \leq \epsilon$, and therefore,

$$V^*(s) - V_{t+1}(s) \leq (1 - \gamma)\epsilon + \gamma\epsilon = \epsilon.$$

□

Lemma 6.5. *Consider optimistic Q-learning and let $T = T((1-\gamma)\epsilon, \delta/|S||A|)$ be the initialization time. With probability at least $1 - \delta$ any state action pair (s, a) that is executed infinitely often is ϵ-optimal, i.e., $V^*(s) - Q^*(s, a) \leq \epsilon$.*

Proof. Given a trajectory let U' be the set of state action pairs that are executed infinitely often, and let M' be the original MDP M restricted to U'. For M' we can use the classical convergence proofs (see Chapter 2 for references) converges to $V^*_{M'}(s)$ and $Q_t(s, a)$, for $(s, a) \in U'$, converges to $Q^*_{M'}(s, a)$, both with probability 1. Since $(s, a) \in U'$ is performed infinitely often it implies that $Q_t(s, a)$ converges to $V_t(s) = V^*_{M'}(s)$ and therefore $Q^*_{M'}(s, a) = V^*_{M'}(s)$. By Lemma 6.4 with probability $1 - \delta$ we have that $V^*_M(s) - V_t(s) \leq \epsilon$, therefore $V^*_M(s) - Q^*_M(s, a) \leq V^*_M(s) - Q^*_{M'}(s, a) \leq \epsilon$. □

A simple corollary is that if we set ϵ small enough, e.g., $\epsilon < \min_{(s,a)}\{V^*(s) - Q^*(s, a) | V^*(s) \neq Q^*(s, a)\}$, then optimistic Q-learning converges to the optimal policy. Before deriving our main theorem we provide a simple technical lemma.

Lemma 6.6. *Let π be a policy such that $\pi(s) = a$ only if $V^*(s) - Q^*(s, a) \leq \varepsilon$, then π is an $\frac{\varepsilon}{1-\gamma}$ optimal.*

Proof. The proof is done by using value iteration to calculate V^π. We let $V^\pi_0(s) = V^*(s)$ and prove by induction that for every state s, that $V^\pi_t(s) - V^*(s) \leq \varepsilon \sum_{i=1}^{t} \gamma^{t-1}$. For $t = 0$ the induction holds, since $V^\pi_0(s) = V^*(s)$. Assume the induction hypothesis holds for $t-1$ and prove it for t. Let a be π's action in state s and observe,

$$\begin{aligned}
V^*(s) - V^\pi_t(s) &= V^*(s) - Q^*(s,a) + Q^*(s,a) - V^\pi_t(s) \\
&\leq \varepsilon + r(s,a) + \gamma \sum_{s'} P^a_{s,s'} V^*(s') - \big(r(s,a) + \gamma \sum_{s'} P^a_{s,s'} V^\pi_{t-1}(s')\big) \\
&= \varepsilon + \gamma \sum_{i=1}^{t-1} \varepsilon \gamma^{i-1} \\
&= \varepsilon \sum_{i=1}^{t} \gamma^{i-1}
\end{aligned}$$

Therefore, we conclude that $V^*(s) - V^\pi(s) \leq \frac{\varepsilon}{1-\gamma}$. □

The next Theorem shows that the greedy policy with respect to the Q function is an almost optimal policy with high probability.

Theorem 6.7. *Consider optimistic Q-learning and let $T = T((1-\gamma)\epsilon, \delta/2|S||A|)$ be the initialization time. For any constant ξ, with probability at least $1-\delta$ there is a time $T_\xi > T$ such that at any time $t > T_\xi$ the strategy defined by the optimistic Q-learning is $(\epsilon+\xi)/(1-\gamma)$-optimal.*

Proof. By Lemma 6.5, we have that every action that is performed infinitely often $V^*(s) - Q^*(s,a) \leq \varepsilon$, therefore there exists a time T_ξ such that every action that is performed afterwards satisfy $V^*(s) - Q^*(s,a) \leq \varepsilon + \xi$. Using Lemma 6.6 we obtain the theorem. □

Chapter 7

Incremental Q-learning

In this section we describe a novel algorithm that we call *incremental Q-learning*. The main idea of the algorithm is to achieve a deterministic tradeoff between exploration and exploitation. We present here two variants of the algorithm, one is an analog of the Boltzman machine, and the second is an analog of the ε-greedy algorithm.

Incremental Q-learning is a greedy policy with respect to the estimated Q-values plus a promotion term. The promotion term of a state-action pair (s, a) is promoted each time the action a is *not* executed in state s, and zeroed each time action a is executed. We show that in incremental Q-learning every state-action pair is taken infinitely often, which implies standard convergence of the estimates. We show that the fraction of time in which sub-optimal actions are executed vanishes in the limit. This implies that the strategy defined by incremental Q-learning converges, in the limit, to the optimal policy. Incremental Q-learning estimates the Q-function as in Q-learning:

$$Q_{t+1}(s, a) = (1 - \alpha_t(s, a))Q_t(s, a) + \alpha_t(s, a)(r_t(s, a) + \gamma V_t(s'))$$

where s' is the next state reached when performing action a in state s at time t. The promotion term A_t is define as follows:

$$A_{t+1}(s,a) = 0: \quad t \in T^{s,a}$$
$$A_{t+1}(s,a) = A_t(s,a) + \psi(\#(s,a,t)): \quad t \in T^{s,not(a)}$$
$$A_{t+1}(s,a) = A_t(s,a): \quad t \in T^{not(s)},$$

where $\psi(i)$ is a *promotion function* which in our case depends only on the number of times we performed (s,a'), $a' \neq a$, since the last time we performed (s,a). We say that a promotion function ψ is well-behaved if: (1) The function ψ converges to zero, i.e., $\lim_{i\to\infty} \psi(i) = 0$, and (2) $\psi(1) = 1$ and $\psi(k) > \psi(k+1) > 0$. For example $\psi(i) = \frac{1}{i}$ is well behaved promotion function.

Incremental Q-learning is a greedy policy with respect to $S_t(s,a) = Q_t(s,a) + A_t(s,a)$. First we show that Q_t, in incremental Q-learning, converges to Q^*.

Lemma 7.1. *Consider incremental Q-learning using a well-behaved learning rate and a well-behaved promotion function. Then for uni-chain MDP, Q_t converges to Q^* with probability* 1.

Proof. Since the learning rate is well-behaved, we need only to show that each state action pair is performed infinitely often. We show that each state that is visited infinitely often, all of its actions are performed infinitely often. Since the MDP is uni-chain this will imply that with probability 1 we reach all states infinitely often, which completes the proof.

Assume that state s is visited infinitely often. Since s is visited infinitely often, there has to be a non-empty subset of the actions A' which are performed infinitely often in s. The proof is by contradiction, namely assume that $A' \neq A$. Let t_1 be the last time that an action not in A' is performed in state s. Since ψ is well behaved we have that $\psi(t_1)$ is constant for a fixed t_1 , it implies that $A_t(s,a)$ diverges for $a \notin A'$. Therefore, eventually we reach a time $t_2 > t_1$ such that $A_{t_2}(s,a) > V_{max}$, for every $a \notin A'$. Since the actions in A' are performed infinitely often there is a time $t_3 > t_2$ such that each action $a' \in A'$ is performed at least once in $[t_2, t_3]$. This implies that $A_{t_3}(s,a) > V_{max} + A_{t_3}(s,a')$ for any $a' \in A'$ and $a \notin A'$. Therefore, some action in $a \in A \setminus A'$ will be performed after t_1, contradicting our assumption. $\square$

The following lemma shows that the frequency of sub-optimal actions vanishes.

Lemma 7.2. *Consider incremental Q-learning using a well behaved learning rate and a well behaved promotion function. Let* $f_t(s,a) = |T^{s,a}|/|T^s|$ *and* (s,a) *be any sub-optimal state-action pair. Then* $\lim_{t\to\infty} f_t(s,a) = 0$*, with probability* 1.

Proof. Let a^* be an optimal action in state s and a be a sub-optimal action. By Lemma 7.1, with probability 1 both $Q_t(s,a^*)$ converges to $Q^*(s,a^*) = V^*(s)$ and $Q_t(s,a)$ converges to $Q^*(s,a)$. Let $h = V^*(s) - Q^*(s,a) > 0$, then there exists a time T_ε such that $V_t(s) - Q_t(s,a) > h_\varepsilon > 0$. Now $A_t(s,a)$ has to be at least h_ε for (s,a) to be executed. Since the promotion function is well behaved, the number of time steps required until $A_t(s,a)$ changes from 0 to h_ε increases after each time we perform (s,a). Since the inter-time between executions of (s,a) diverges, the frequency $f_t(s,a)$ vanishes. $\square$

The following corollary gives a quantitative bound for a specific promotion function.

Corollary 7.3. *Consider incremental Q-learning with learning rate* $\alpha_t(s,a) = 1/\#(s,a,t)$ *and* $\psi(k) = 1/e^k$*. Let* (s,a) *be a sub-optimal state-action pair. The number of times* (s,a) *is performed in the first* n *visits to state* s *is* $\Theta(\ln(\frac{n}{V^*(s)-Q^*(s,a)}))$*, for sufficiently large* n.

Furthermore, the return obtained by incremental Q-learning converges to the optimal return.

Corollary 7.4. *Consider incremental Q-learning using a well behaved learning rate and a well behaved promotion function. For every* ϵ *there exists a time* T_ϵ *such that for any* $t > T_\epsilon$ *we have that the strategy* π *defined by incremental Q-learning is* ϵ*-optimal with probability* 1.

So far we only described an algorithm which is an analog of the Boltzman machine, next we present a less intuitive algorithm which is an analog of the ε-greedy algorithm

$$A_{t+1}(s,a) = 0: \ \ t \in T^{s,a}$$
$$A_{t+1}(s,a) = A_t(s,a) + d_t(s,a)\psi(\#(s,a,t)): \ \ t \in T^{s,not(a)}$$
$$A_{t+1}(s,a) = A_t(s,a): \ \ t \in T^{not(s)},$$

where $d_t(s,a)$ is defined as follows,

$$d_{t+1}(s,a) = \max_b(Q_t(s,b) + A_t(s,b)) - Q_t(s,a): \ \ t \in T^{s,a}$$
$$d_{t+1}(s,a) = d_t(s,a): \ \ t \in T^{s,not(a)}$$
$$d_{t+1}(s,a) = d_t(s,a): \ \ t \in T^{not(s)},$$

Our requirements from ψ this time are different and we only require that $\lim_{k\to\infty}\psi(k) = 0$ and we allow ψ to me non monotonic. Next we show that the the modified incremental algorithm converges with probability 1.

Lemma 7.5. *Consider modified incremental Q-learning using a well-behaved learning rate and a modified well-behaved promotion function. Then for uni-chain MDP, Q_t converges to Q^* with probability* 1.

Proof. Since the learning rate is well-behaved, we need only to show that each state action pair is performed infinitely often. We show that each state that is visited infinitely often, all of its actions are performed infinitely often. Since the MDP is uni-chain this will imply that with probability 1 we reach all states infinitely often, which completes the proof.

Assume that state s is visited infinitely often. Since s is visited infinitely often, there has to be a non-empty subset of the actions A' which are performed infinitely often in s. The proof is by contradiction, namely assume that $A' \neq A$. Let t_1 be the last time that an action not in A' is performed in state s. We have that $\psi(t_1)$ is positive constant for a fixed t_1, it implies that $A_t(s,a)$ diverges for $a \notin A'$. Therefore, eventually we reach a time $t_2 > t_1$ such that $A_{t_2}(s,a) > V_{max}$, for every $a \notin A'$. Since the actions in A' are performed infinitely often there is a time $t_3 > t_2$ such

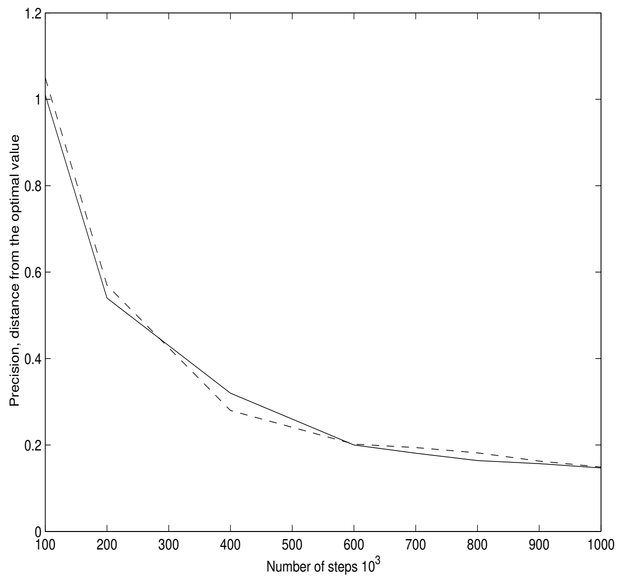


Figure 7.1: Example of 50 states MDP, where the discount factor, γ, is 0.9. The leaning rate of both Incremental and epsilon greedy Q-learning is set to 0.8. The dashed line represents the epsilon greedy Q-learning.

that each action $a' \in A'$ is performed at least once in $[t_2, t_3]$. This implies that $A_{t_3}(s,a) > V_{max} + A_{t_3}(s,a')$ for any $a' \in A'$ and $a \notin A'$. Therefore, some action in $a \in A \setminus A'$ will be performed after t_1, contradicting our assumption. $\square$

7.1 Experiments

In this section we show some experimental results, comparing Incremental Q-Learning and epsilon-greedy Q-Learning. One can consider incremental Q-learning as a derandomization of ϵ_t-greedy Q-Learning, where the promotion function satisfies $\psi_t = \epsilon_t$. The experiment was made on MDP, which includes 50 states and two actions per state. Each state action pair immediate reward is randomly chosen in the interval $[0, 10]$. For each state and action (s,a) the next state transition is random, i.e., for every state s' we have a random variable $X_{s'} \in [0,1]$ and $P^a_{s,s'} = \frac{X^{s,a}_{s'}}{\sum_{\hat{s}} X^{s,a}_{\hat{s}}}$. For the ϵ_t-greedy Q-learning, we have $\epsilon_t = 10000/t$ at time t, while for the incremental we have $\psi_t = 10000/t$. Each result in the experiment is an average of ten different runs.

In Figure 7.1, we observe similar behavior of the two algorithms. This experiment demonstrates the strong experimental connection between these methods. We plan to further investigate the theoretical connection between ϵ-greedy, Boltzman machine and incremental Q-Learning.

Chapter 8

Markov Decision Process Minimization

This Chapter is based on [34]. Finding an equivalent (or almost equivalent) MDP whose size is smaller, is an important problem in Reinforcement learning, since a good policy in small MDP will translate to a good policy in the large MDP. While [42] showed an efficient way to calculate the *exact* minimal MDP, we show that finding an *almost* equivalent minimal MDP is NP-hard. Later we show that using L_∞ norm as was done in [42] is problematic, while L_1 is the appropriate norm. Finally we present an efficient approximation algorithm. We first define an equivalence relation between MDPs

Definition 8.1. An MDP $M_1 = (S_1, A, R_1, P)$ and MDP $M_2 = (S_2, A, R_2, P)$ are ε-equivalent with respect to L_k norm if there exist mappings $\phi_1 : S_1 \to S$ and $\phi_2 : S_2 \to S$ to MDP $M = (S, A, R, P)$, such that ϕ_1 and ϕ_2 are surjective and for every $s_1 \in S_1, s_2 \in S_2$ such that $\phi_1(s_1) = \phi_2(s_2) = s$ the following holds:

1. $\forall a \in A \quad \left(\sum_{s' \in S} \left(\sum_{\hat{s};\phi_i(\hat{s})=s'} P^a_{s_i,\hat{s}}(M_i) - P^a_{s,s'}(M)\right)^k\right)^{1/k} \leq \varepsilon \quad i = 1,2$

2. $\forall a \in A \quad |R_i(s_i, a) - R(s, a)| \leq \varepsilon \qquad i = 1,2$

Definition 8.2. An MDP $M_1 = (S_1, A, R_1, P)$ is an ε-homogenous partition of MDP $M = (S, A, R, P)$ with respect to L_k norm if there exists mapping $\phi : S \to S_1$, such that ϕ is surjective and for every s the following holds:

1. $\forall a \in A \quad \left(\sum_{s' \in S} \left(\sum_{\hat{s};\phi(\hat{s})=s'} P^a_{s,\hat{s}}(M) - P^a_{\phi(s),s'}(M_1)\right)^k\right)^{1/k} \leq \varepsilon$

2. $\forall a \in A \quad \max_{\tilde{s}:\phi(\tilde{s})=s} |R(\tilde{s}, a) - R_1(s, a)| \leq \varepsilon$

We start by describing briefly the minimization process for MDPs of Givan *et al.* [42]. The minimization process starts with an MDP, M, and build a reduced MDP which is equivalent to the original MDP and whose size is the minimal among the equivalent MDPs

An equivalence relation is defined as a transitive binary relation, E. The equivalence relation groups similar states in the original MDP and in the reduced MDP and each equivalence class translates into one state. We often denote by a block a group of states in a specific equivalence class. Given an MDP, M and an equivalence relation $E \subseteq S \times S$, an operator I is defined as

$$I(E) = \{(i, j)|R(i) = R(j) \wedge E(i, j) \wedge P^a_{i,i'}(E) = P^a_{j,i'}(E)\},$$

where $P^a_{i,j'}(E) = \sum_{j:E(j,j')=1} P^a_{i,j}$. The minimal model is the fixed point of the process $E_{n+1} = I(E_n)$, where $I(E) \subseteq E$.

We define the reward partition as the equivalence relation, $E(i, j) = \{i, j| \|R(i, \cdot) - R(j, \cdot)\|_\infty = 0\}$. We say that a block $B \subseteq S$ is stable with respect to block C if every state in B has the same transition probability with respect to C, i.e., for every $i, i' \in B$ we have $P^a_{i,C} = P^a_{i',C}$, where $P^a_{i',C} = \sum_{j \in C} P^a_{i',j}$. Let $B = (B_1, \ldots, B_m)$, where if $i \in B_k$ then $j \in B_k \iff E(i, j) = 1$. We define the procedure $SPLIT(B_i, B_j, P)$ that replaces the block B_i by the uniquely determined sub-blocks of B_i, $B_{i1}, \ldots, B_{ik}$ such that each sub-block is the maximal stable sub-block with respect to B_j. In other words, all the states $s \in B_{il}$ have the same probability P^a_{s,B_j} for every action a and no two sub-blocks B_{il} and B_{ik} have the same set of probabilities.

The algorithm of Givan *et al.* [42] works as follows. It begins by making E_0 the reward partition. It then iterates and checks for every pair of blocks if they are stable; if not, it performs the procedure $SPLIT$. The algorithm terminates when all blocks are stable with respect to every other block. The algorithm clearly terminates, since each SPLIT increases the number of blocks and the number of blocks is bounded by $|S|$.

Theorem 8.3. *[42] The minimization problem for MDP, given in its tabular form, is in P.*

8.1 Hardness of epsilon Equivalence

In [42] it was shown that finding the minimal model is in P. On the other hand they have shown that finding the minimal (or minimal ϵ-equivalent) model is NP-Hard for factored MDPs. We show that finding the minimal ϵ-equivalent model is NP-Hard even for MDPs that are represented in a tabular form. We use here a variant of the k-center problem known as **metric k cluster**.

Lemma 8.4. *[45] Let $G = (V, E)$ be a complete undirected graph with edge costs satisfying the triangle inequality, and let k be a positive integer. Find a partition of V into sets $V_1, ..., V_k$ such that the maximum cost of an edge between two vertices in the same set is minimized.*

We will use different notations and a more restricted problem to simplify the reduction.

Corollary 8.5. *The following ε-cover problem is NP-Complete. Given n points, $X = (x_1, ..., x_n)$, where x_i is in $\mathbb{R}^m$, a positive number K and a norm distance, L_q. Partition the points into K sets, $X_1, ...X_K$ such that $\max_{1\leq i\leq K\ p,p'\in X_i} \|p - p'\|_q \leq \varepsilon$.*

We obtain our theorem by a reduction from the ε-cover problem.

Theorem 8.6. *Given an MDP, M, and a positive number K the problem of determining whether there exists an ϵ-homogenous partition, in L_q, of size no more than K is NP-Complete.*

Proof. The problem is clearly in NP. Next we show a reduction from metric k cluster to this problem. We reduce the n points, $x_1, ..., x_n$ to MDP $M = (s_1, ..., s_n)$ and keep K and ε identical in both problems. In M each state has m actions, the reward function is $R(s_i, a_j) = x_i^j$, where x_i^j is the jth coordinate of x_i, and the next state distribution is uniform, i.e. $\forall i, j, k \quad P^{a_k}_{s_i,s_j} = \frac{1}{n}$. We show that there exists an ϵ - homogenous partition of size no more than K if and only if there exists a partition of X into sets $X_1, ..., X_K$ such that $\max_{1\leq i\leq K, p,p'\in X_i} \|p - p'\|_q \leq \epsilon$. We first prove that if there exists an ϵ - homogenous partition of size K then there exists a K cluster. Since the next state distribution is uniform for every state-action pair, then an ϵ - homogenous partition of the MDP has to be according to the reward. Let $B_1, \ldots, B_K$ be the partition for the MDP, then we have that $\max_{1\leq i\leq K, s,s'\in B_i} \|R(s, \cdot) - R(s', \cdot)\|_q \leq \epsilon$ and let $X_1, \ldots, X_K$ be the matching partition in X. Now we show that the partition that X_i induces on X is an ϵ cover. We observe that

$$\begin{aligned}
\max_{1\leq i\leq K, u,v\in X_i} \|u - v\|_q &= \max_{1\leq i\leq K, u,v\in X_i} \left(\sum_{j=1}^{m} |u_j - v_j|^q\right)^{1/q} \\
&= \max_{1\leq i\leq K, s,s'\in B_i} \left(\sum_{j=1}^{m} |R(s, a_j) - R(s', a_j)|^q\right)^{1/q} \leq \varepsilon.
\end{aligned}$$

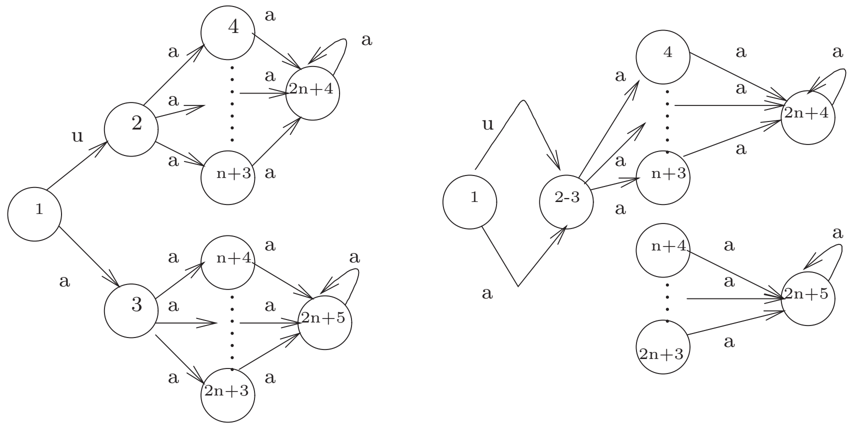
Using the fact that every block is stable with respect to every block, the proof of the other direction is identical. □ □

8.2 L_1 versus L_∞

In this section we present an example, which demonstrates the drawback of using L_∞ as measure for an ϵ-homogenous partition of the MDP. On the other hand we derive performance guarantees for the L_1 measure.

Lemma 8.7. *For every $\varepsilon > 0$, there exists an MDP M with an ε-homogenous partition, M_ε, with respect to L_∞, such that an optimal policy in M_ε induces a policy in M that is arbitrarily far from from the optimal policy.*

Proof. Consider the MDPs in Figure 8.1, where M and M_ε are ε-equivalent for $\varepsilon =$



M (a)

M_ε (b)

Figure 8.1: An example in which two MDPs are ε-equivalent in L_∞ norm but an optimal policy in M_ε is far from optimal in M. Let $n = \lceil 1/\varepsilon \rceil$. The rewards are $R(1,u) = R(1,a) = 0, R(2,a) = R(3,a) = 1, R(2n+4,a) = R_{max}, R(2n+5,a) = 0$ $\forall k \in \{4, 2n+3\}$ $R(k,a) = 2\varepsilon k$. In states 2, 3 the next state distribution is uniform. We note that there are only two policies in M, one performs a at state 1 and other performs u at state 1. In M_ε the two policies coincide.

$1/n$. There are only two policies in M, π_a, which performs action a in state 1 and π_u, which performs action u in state 1. Note that in M_ε both policies are optimal. We consider in M_ε the policy, π_a, which performs action a in state 1. The induced policy, π_a in M, satisfies $V_M^{\pi_a}(1) \le 6$ for any γ. The optimal policy, π_u, satisfies $V_M^*(1) \ge \frac{R_{max}\gamma^3}{1-\gamma} = \gamma^3 V_{max}$. For constant γ the difference can be arbitrarily large, independent of ε. $\square$

In the next lemma we show that for every policy, π^ε in an ε-homogenous partition of an MDP, with respect to L_1 norm, its induced policy, π is at most $\varepsilon\frac{V_{max}}{1-\gamma}$ from the value of π^ε in M_ε for every state. The proof is done by relying on the following fact:

Fact 8.8. Let V_t be the values generated by Value iteration. If $\|V_{t+1} - V_t\|_\infty \le \varepsilon$ then $\|V_t - V^*\|_\infty \le \frac{\varepsilon}{1-\gamma}$

Lemma 8.9. *Let M_ε be an ε-homogenous partition of MDP M, with respect to L_1 norm, and π any policy in M_ε, then the induced policy π^M on M, that is $\pi^M(s) = \pi(\phi(s))$, satisfies*

$$\forall s \in S \quad |V_M^{\pi^M}(s) - V_{M_\varepsilon}^{\pi}(\phi(s))| \leq \frac{\varepsilon V_{max}}{1-\gamma}$$

Proof. The proof is done by estimating the value of π^M using Value Iteration with initial value $V_0^{\pi^M}(s) = V_{M_\varepsilon}^{\pi}(\phi(s))$. Since we use L_1 norm, each state can deviate only in ε from its representative state in M_ε and the difference $\|V_0^{\pi^M} - V_1^{\pi^M}\|_\infty$ is bounded by εV_{max}. Using the fact that $\|V_0^{\pi^M} - V_1^{\pi^M}\|_\infty \leq \varepsilon V_{max}$ and Fact 8.8 we can conclude that

$$\max_{s \in S} |V_M^{\pi^M}(s) - V_{M_\varepsilon}^{\pi}(\phi(s))| = \|V_0^{\pi^M} - V_M^{\pi^M}\|_\infty = \|V_0^{\pi^M} - V_\infty^{\pi^M}\|_\infty \leq \frac{\varepsilon V_{max}}{1-\gamma},$$

where $V_\infty^{\pi^M} = V_M^{\pi^M}$ is the value of π^M in M. □

The next lemma shows that the optimal policy in M_ε induces a nearly optimal policy in M.

Lemma 8.10. *Let M_ε be an ε-homogenous partition of MDP M, with respect to L_1 norm, then the optimal policy in M_ε induces an $\frac{2\varepsilon V_{max}}{1-\gamma}$-optimal policy in M.*

Proof. Let σ be the optimal policy in M_ε and σ^M be the induced policy in M. By Lemma 8.9 we have that $\max_s |V_M^{\sigma^M}(s) - V_{M_\varepsilon}^{\sigma}(\phi(s))| \leq \frac{\varepsilon V_{max}}{1-\gamma}$. Next we show that if we use a Value Iteration on M with initial value $V_0^M(s) = V_{M_\varepsilon}^{\sigma}(\phi(s))$ then we have that

$$\|V_0^M - V_\infty^M\|_\infty = \|V_0^M - V_M^*\|_\infty \leq \frac{\varepsilon V_{max}}{1-\gamma}.$$

Since we use L_1 norm, once again we have that $\|V_0^M - V_1^M\|_\infty$ is bounded by εV_{max}. Combining the two parts, we obtain that σ^M is an $\frac{2\varepsilon V_{max}}{1-\gamma}$-optimal policy in M. □

8.3 Approximation scheme

Since finding the minimal ε-equivalent MDP is NP-Hard, we would like to find an approximate solution to the problem. The natural approximation is to give an MDP, which is ε-equivalent and is K times larger than the minimal ε-equivalent MDP. Unfortunately, we do not present such approximation. We present a bicriteria approximation algorithm for finding an ε-equivalent model for an MDP. Usually in approximation algorithms we are given one objective function, which we try to minimize. In bicriteria approximations two objective functions that we try to minimize are given. Thus, a bicriteria is an approximate solution in two parameters of the problem. Our criteria are both the MDP size and the parameter ε in the ε-equivalence. We prove that our algorithm is $(2\sqrt{|S|}, \sqrt{|S|})$ approximation with respect to (size, ε), that is our MDP might be $\sqrt{|S|}\varepsilon$-equivalent and $2\sqrt{|S|}$ times larger than the minimal ε-equivalent MDP. In fact, for our algorithm either (1) produces an equivalent MDP, which is $2\sqrt{|S|}$ larger than the minimal size ε-equivalent MDP, or (2) produces an MDP which is $\sqrt{|S|}\varepsilon$-equivalent, whose size is at most the size of the minimal ε-equivalent MDP.

We first give a high level description of our approximation algorithm. We initially build a graph $G = (S, E)$, where the vertices are the MDP states and there is an edge between states if their rewards are ε-close. After the initialization, we run in phases. In each phase we check for each edge if it is consistent with some legal partition. A partition is legal with respect to G if every vertex in it is connected to every other vertex. If we observe that the edge is not consistent with any legal partition, then we delete that edge. To check whether an edge (s, t) is consistent, we compute the connected components of G and see if s and t have similar next state transition with respect to them. (The detailed procedure $CheckConsistent$ is formally described in Alg. 9.) We terminate when we cannot delete any edges. Clearly the algorithm terminates after at most $|S|^2$ phases. (The Algorithm is formally described in Alg. 8)

Since our algorithm returns an MDP partition, we show how to construct an MDP from an MDP partition. We note that for an MDP M a partition of S by itself does not define a unique MDP, but a family of MDPs. This family can be viewed also as a bounded MDP [43]; we take a different approach and choose arbitrarily representative MDP for each partition.

Definition 8.11. Let $M = (S, A, P, R)$ and ϕ be a mapping from S to S_1. Then the

Input : MDP M, $\varepsilon > 0$ precision
Output: A partition of M to Equivalence Relations
Build a graph $G = (S, E)$, where $(s, t) \in E$ if $\forall a \in A \; |R(s, a) - R(t, a)| \leq \varepsilon$;
repeat
  Deleted = *FALSE*;
  foreach $(s, t) \in E$ **do**
    foreach $a \in A$ **do**
      if *CheckConsistent(s,t,a,E,S,P)* = *FALSE* **then**
        Deleted = *TRUE* ;
        $E = E - (s, t)$;
      end
    end
  end
until *Deleted* = *FALSE*;
Compute $C_1, \ldots, C_l$ the connected components of G= (S,E);
if $\max_i \mathit{Diameter}(C_i) \geq \sqrt{S}$ **then**
  Return S
else
  Return $C = \cup_i C_i$
end

Algorithm 8: An algorithm which calculates an ε-equivalent MDP

MDP, $M_1 = (S_1, A, P_1, R_1)$ induced by (M, ϕ) is defined as follows. For each $s \in S_1$ we choose arbitrarily s' such that $\phi(s') = s$. The reward in $s \in S_1$ is $R_1(s, a) = R(s', a)$ and the next state distribution is $P^a_{s,t}(M_1) = \sum_{\phi(t')=t} P^a_{s',t'}(M)$.

In the following lemmas we provide correctness proof of Algorithm 8. We first prove that if s and t are in the same equivalence class in some ε-homogenous partition, then the edge (s, t) is never removed from the graph during Algorithm. 8.

Lemma 8.12. *Let s and t be two states such that there exists an ε-homogenous partition MDP in which s and t are in the same equivalence class, then the edge (s, t) is never removed from the graph in Algorithm 8.*

Proof. Let $C^k_1, \ldots, C^k_l$ be the connected components of G at the kth stage and $EC_1, \ldots, EC_m$ be the equivalence classes of an ε-homogenous partition MDP in which s and t are in the same equivalence class. We prove by induction on the algorithm steps (number of calls to $CheckConsistent$) that (1) the edge (s, t) is never removed

Input : state s , state t, action a, Edges between states E, MDP states S, Next state distribution P
Output: Boolean answer whether s and t are consistent
Compute the connected components of $G = (S, E)$, $C_1, \ldots, C_l$;
if $\sum_{i=1}^{l} |P^a_{s,C_i} - P^a_{t,C_i}| \leq \varepsilon$ **then**
    Return TRUE;
else
    Return FALSE;
end

Algorithm 9: CheckConsistent(s,t,E,S,P) Procedure

(2) $C_i^k = \cup_{EC_j \subseteq C_i^k} EC_j$ for every i, i.e., each connected component is a union of equivalence classes from an ε-homogenous partition MDP. The basis is due to the fact that the first stage is the reward partition. We assume that the induction hypothesis holds for the first k stages and prove for the $k+1$ stage. By the induction assumption we have that $C_i^k = \cup_{j:EC_j \subseteq C_i^k} EC_j$ for every i. Therefore, for every action a we obtain the following

$$\begin{aligned}
\sum_{i=1}^{l} |P^a_{s,C_i^k} - P^a_{t,C_i^k}| &= \sum_{i=1}^{l} \left| \sum_{j:EC_j \subseteq C_i^k} P^a_{s,EC_j} - P^a_{t,EC_j} \right| \\
&\leq \sum_{i=1}^{l} \sum_{j:EC_j \subseteq C_i^k} \left| P^a_{s,EC_j} - P^a_{t,EC_j} \right| \\
&= \sum_{j=1}^{m} \left| P^a_{s,EC_j} - P^a_{t,EC_j} \right| \leq \varepsilon.
\end{aligned}$$

We conclude that (s,t) is never removed from the graph if (s,t) are in the same equivalence class in some ε-homogenous partition MDP. Therefore, all states in an equivalence class form a clique and each equivalence class is contained in a single connected component. □ □

Next we prove that at the end of the algorithm we have a $(2\sqrt{S}, \sqrt{S})$-approximation. We first show that the size of the resulting MDP is no larger than $2\sqrt{S}$ the minimal ε-equivalent MDP.

Lemma 8.13. *If* $\max_i Diameter(C_i) \geq \sqrt{|S|}$ *then there are at least* $\frac{\sqrt{|S|}}{2}$ *states in the minimal* ε*-equivalent MDP.*

Proof. Consider the diameter's path, $s_1, ..., s_{\sqrt{|S|}}$, of a connected component C. By Lemma 8.12 we have that no three states in the path can be in the same equivalence class. (Otherwise the diameter would have been smaller.) Thus there are at least $\left\lfloor \sqrt{|S|}/2 \right\rfloor$ states in the minimal ε-equivalent MDP. □ □

Next we claim that if we take each connected component as an equivalence class, then they are εD-stable, where $D = \max_i Diameter(C_i)$.

Lemma 8.14. *Consider an MDP,* M_1*, which is induced by* (M, ϕ)*, such that if* $s, s' \in C_i$ *in Algorithm 8 then* $\phi(s) = \phi(s')$*. Then we have that* M_1 *is* εD*-equivalent to* M*, where* $D = \max_i Diameter(C_i)$*.*

Proof. Since ϕ maps all the states in a connected component C_i to a single state in M, we let the states of M_1 be the connected components. This implies that if $s \in C_i$ then $\phi(s) = C_i$. First we show that for every connected component

$$\forall a \forall s, t \in C_i \quad |R(s,a) - R(t,a)| \leq \varepsilon D$$

We prove it by induction on the length of the path between the states in C_i. For the basis we have that if the path length between s and t is one, then there is an edge between them. This implies that $max_a|R(s,a) - R(t,a)| \leq \varepsilon$. Assume the induction assumption holds for $k-1$ and prove for k and let $s = s_1, \ldots, s_k = t$ be the path between s and t. From the induction assumption we have that $max_a|R(s_1,a) - R(s_{k-1},a)| \leq (k-1)\varepsilon$ and since there is an edge between s_{k-1} to s_k we have that $max_a|R(s_{k-1},a) - R(s_k,a)| \leq \varepsilon$. Using the triangle inequality we complete the induction. Next we prove that for any s, t in a connected component the following holds:

$$\max_{a \in A} \sum_{C_i} \left| \sum_{\hat{s};\phi(\hat{s})=C_i} P^a_{s,\hat{s}}(M) - \sum_{\hat{s};\phi(\hat{s})=C_i} P^a_{t,\hat{s}}(M) \right| \leq \varepsilon D$$

Fix an action a. We prove by induction on the path length that $\sum_{i=1}^{m} |P^a_{s_1,C_i} - P^a_{s_j,C_i}| \le j\varepsilon$, where m is the number of connected component. The basis of the induction is due to the fact that when the algorithm terminates, it cannot delete more edges, and therefore for every edge we have that $\sum_{i=1}^{m} |P^a_{s_j,C_i} - P^a_{s_{j+1},C_i}| \le \varepsilon$. We assume that the induction assumption holds for $k-1$ and prove for k. For a path $s = s_1, \dots, s_k = t$ in G, we show that $\sum_{i=1}^{m} |P^a_{s_1,C_i} - P^a_{s_k,C_i}| \le k\varepsilon$. Since $\sum_{i=1}^{m} |P^a_{s_{k-1},C_i} - P^a_{s_k,C_i}| \le \varepsilon$ and by the induction assumption we have that $\sum_{i=1}^{m} |P(s_1, C_i, a) - P(s_{k-1}, C_i, a)| \le (k-1)\varepsilon$, we can conclude that $\sum_{i=1}^{m} |P^a_{s_1,C_i} - P^a_{s_k,C_i}| \le k\varepsilon$. □

Combining Lemma 8.13 and Lemma 8.14 we obtain our main theorem.

Theorem 8.15. *Given an MDP M, Algorithm 8 terminates in polynomial time and returns an MDP partition, ϕ with induced MDP, $M_\varepsilon = (M, \phi)$, that is at most $\sqrt{|S|}\varepsilon$-equivalent and with size no larger than $2\sqrt{|S|}$ time the size of the minimal ε-equivalent MDP.*

8.3.1 A heuristic Finer Partition

In this section, we suggest an improved *CheckConsistent* procedure. This procedure always performs at least as good as the original *CheckConsistent* procedure. However, this procedure is more complicated and we could not provide a better performance grantees using it. The procedure *CheckConsistent*1 checks whether an edge (s, t) is consistent by building the graph (in Fig. 2) for each action and check if the max flow is close to 1, more precisely if it is at least $1 - \varepsilon/2$. (The detailed procedure *CheckConsistent*1 is formally described in Alg. 10.) We also note that the difference between the procedures is that in *CheckConsistent* procedure we connect states between the second and the third layer if they are in the same connected component, while in *CheckConsistent*1 procedure we connect them only if they have an edge in G.

Input : state s , state t, action a, Edges between states E, MDP states S, Next state distribution P
Output: Answer whether s and t are consistent
Build a graph with four layres;
The first layer contains s ;
The second layer contains S ;
The third layer contains S ;
The fourth layer contains t;
Connect s to every vertex s' in the second layer with capacity $c(s, s') = P^a_{s,s'}$;
If $(s', s'') \in E$ then connect s' in the second layer to a vertex s'' in the third layer with capacity $c(s', s'') = 1$;
Connect every vertex in the third layer to t with capacity $c(s', t) = P^a_{t,s'}$;
Calculate the max flow from s to t;
if *max flow* $< 1 - \varepsilon/2$ **then**
    Return FALSE;
else
    Return TRUE;
end

Algorithm 10: CheckConsistent1(s,t,E,S,P) Procedure

8.4 Markov Decision Process Equivalence

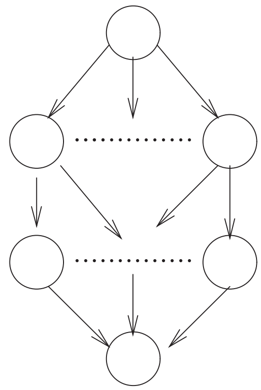
We present a polynomial method to check whether two MDPs are equivalent. Our algorithm is very simple and uses any procedure to compute the minimal MDP, which we call minimal equivalent MDP. For instance such a procedure is given in [42].

Assuming the algorithm for finding the minimal equivalent model is polynomial in the MDP size as in [42], then our algorithm is polynomial as well.

Lemma 8.16. *Algorithm 11 runs in polynomial time in the MDP size.*

Next we show the correctness of the algorithm. We first show that any two states from the same minimal MDP D_1 or D_2 have to be in the same equivalence relation of M.

Lemma 8.17. *No equivalence relation in the minimal partition of M contains two states from either D_1 or D_2.*



(a)

Figure 8.2: The graph built by *CheckConsistent* procedure

Proof. We let $D_1 = (s_1^1, ..., s_1^m)$ and let $D_2 = (s_2^1, ..., s_2^m)$. The proof is done by contradiction. Assume there are two states from D_1, s_1^i and s_1^j, in the same equivalence class in M, then we can construct to M_1 an equivalent model, which consists of at most $size(D_1) - 1$ states. Since s_1^i and s_1^j are stable with respect to any other block in D, then they are stable with respect to the projection of D to D_1 and can be in the same equivalence class. This contradicts the minimality of D_1. □ □

Lemma 8.18. *MDPs M_1 and M_2 are equivalent if and only if all the equivalence relations of D consists only pairs (s_i, s_j) for which $s_i \in D_1$ and $s_j \in D_2$, i.e. $size(D) = size(D_1) = size(D_2)$.*

Proof. (a) $\Leftarrow$

Trivial and we have a mapping from M_1 and M_2 to D.

(b)$\Rightarrow$

We assume that M_1 and M_2 are equivalent; thus their minimal models, $D_1 = (s_1^1, ..., s_1^m)$ and $D_2 = (s_2^1, ..., s_2^m)$, satisfy $\forall i \;\; s_1^i \equiv s_2^i$. (we do not assume that we know which state in D_1 corresponds to which state in D_2).

Now we follow the construction of the minimal model as appears in [42]. We prove

Input : MDPs M_1, M_2
Output: Answer whether the MDPs are equivalent
Calculate D_1, the minimal Equivalent MDP of M_1;
Calculate D_2, the minimal Equivalent MDP of M_2;
if *(size(D_1) $\neq$ size(D_2))* **then**
    Return FALSE;
end
Calculate D, the minimal Equivalent MDP of
$M = D_1 \cup D_2 = (S_1 \cup S_2, R_1 \cup R_2, P_1 \cup P_2, A)$;
if *(size(D) = size(D_1))* **then**
    Return TRUE;
else
    Return FALSE;
end

Algorithm 11: An algorithm which finds whether two MDPs are equivalent

by induction on the construction steps that for each block B the following holds

$$\forall i \quad : \quad s_1^i \in B \iff s_2^i \in B$$

For the induction basis. Since for any i states s_1^i and s_2^i must have the same reward, then they must be in the same block in the initial reward partition. Next we assume that the claim holds for any $j < k$ and prove for k. We assume that in the kth step we check the stability of block B which consists s_1^i and s_2^i with respect to block C. By the induction assumption for every action a we have $P(s_1^i, C, a) = P(s_2^i, C, a)$ and since $SPLIT(B, C, P)$ replaces the block B by the uniquely determined sub-blocks B_i, we have that s_1^i and s_2^i will be in the same sub block after the kth step. Together with Lemma 8.17 we conclude the lemma. □

□

By Lemma 8.18 we have that Algorithm 11 checks whether two MDPs are equivalent and by Lemma 8.16 we have that it is polynomial. Therefore, we derive the following theorem.

Theorem 8.19. *The equivalence problem for MDPs, given in their tabular form, is in P.*

Chapter 9

The On-line MDP

This Chapter is based on [27]. The motivation of this work is to understand how to *efficiently* incorporate the benefits of existing experts algorithms into a more adversarial reinforcement learning setting, where certain aspects of the environment could change over time. A naive way to implement an experts algorithm is to simply associate an expert with each fixed policy. The running time of such algorithms is polynomial in the number of experts and the regret (the difference from the optimal reward) is logarithmic in the number of experts. For our setting the number of policies is huge, namely $\#\text{actions}^{\#\text{states}}$, which renders the naive experts approach computationally infeasible.

Furthermore, straightforward applications of standard regret algorithms produce regret bounds which are logarithmic in the number of policies, so they have linear dependence on the number of states. We might hope for a more effective regret bound which has *no dependence* on the size of state space (which is typically large).

The setting we consider is one in which the dynamics of the environment are known to the learner, but the reward function can change over time. We assume that after each time step the learner has complete knowledge of the previous reward functions (over the entire environment), but does not know the future reward functions.

McMahan etal. [79] also considered a similar setting — they also assume that the reward function is chosen by an adversary and that the dynamics are fixed. However, they assume that the cost functions come from a finite set (but are not observable) and the goal is to find a min-max solution for the related stochastic game.

The on-line MDP similarly to standard MDP consists of state space S; initial state distribution d_1 over S; and a sequence of reward functions $r_1, r_2, \dots r_T$, where r_t is the (bounded) reward function at time step t mapping $S \times A$ into $[0, 1]$.

The goal is to maximize the sum of *undiscounted* rewards over a T step horizon. We assume the agent has complete knowledge of the transition model P, but at time t, the agent only knows the past reward functions r_1, r_2, $\dots r_{t-1}$. Hence, an algorithm $\mathcal{A}$ is a mapping from S and the previous reward functions $r_1, \dots r_{t-1}$ to a probability distribution over actions, so $\mathcal{A}(a|s, r_1, \dots r_{t-1})$ is the probability of taking action a at time t.

We define the return of an algorithm $\mathcal{A}$ as:

$$V_{r_1, r_2, \dots r_T}(\mathcal{A}) = \frac{1}{T} \mathbf{E} \left[\sum_{t=1}^{T} r_t(s_t, a_t) \middle| d_1, \mathcal{A} \right]$$

where $a_t \sim \mathcal{A}(a|s_t, r_1, \dots r_{t-1})$ and s_t is the random variable which represents the state at time t, starting from initial state $s_1 \sim d_1$ and following actions $a_1, a_2, \dots a_{t-1}$. Note that we keep track of the expectation and not of a specific trajectory (and our algorithm specifies a distribution over actions at *every* state and at *every* time step t).

Ideally, we would like to find an algorithm $\mathcal{A}$ which achieves a large reward $V_{r_1, \dots r_T}(\mathcal{A})$ *regardless* of how the adversary chooses the reward functions. In general, this of course is not possible, and, as in the standard experts setting, we desire that our algorithm competes favorably against the best fixed stationary policy $\pi(a|s)$ in hindsight.

9.1 Mixing Time

Before we provide our results a few definitions are in order. For every stationary policy $\pi(a|s)$, we define P^π to be the transition matrix induced by π, where the component $[P^\pi]_{s,s'}$ is the transition probability from s to s' under π, i.e. $[P^\pi]_{s,s'} = \sum_{a \in A} \pi(a|s) P^a_{s,s'}$. Also, define $d_{\pi,t}$ to be the state distribution at time t when following

π, *ie*

$$d_{\pi,t} = d_1 (P^\pi)^t$$

where we are treating d_1 as a row vector here.

We assume throughout this section that the MDP is a unichain Model.

Assumption 9.1. *(Mixing) We assume the transition model over states, as determined by π, has a well defined stationary distribution, which we call d_π. More formally, for every initial state s, $d_{\pi,t}$ converges to d_π as t tends to infinity and $d_\pi P^\pi = d_\pi$. Furthermore, this implies there exists some τ such that for* all *policies π, and distributions d and d',*

$$\|dP^\pi - d'P^\pi\|_1 \leq e^{-1/\tau}\|d - d'\|_1$$

where $\|x\|_1$ denotes the l_1 norm of a vector x. We refer to τ as the mixing time *and assume that $\tau > 1$.*

The parameter τ provides a bound on the planning horizon timescale, since it implies that *every* policy achieves close to its average reward in $O(\tau)$ steps [1]. This parameter also governs how long it effectively takes to switch from one policy to another (after time $O(\tau)$ steps there is little information in the state distribution about the previous policy).

This assumption allows us to define the average reward of policy π in an MDP with reward function r as:

$$\eta_r(\pi) = \mathbf{E}_{s \sim d_\pi, a \sim \pi(a|s)}[r(s,a)]$$

and the value, $Q_{\pi,r}(s,a)$, is defined as

$$Q_{\pi,r}(s,a) \equiv \mathbf{E}\left[\sum_{t=1}^{\infty}\left(r(s_t,a_t) - \eta_r(\pi)\right)\middle| s_1 = s, a_1 = a, \pi\right]$$

[1] If this timescale is unreasonably large for some specific MDP, then one could artificially impose some horizon time and attempt to compete with those policies which mix in this horizon time, as done [59].

where and s_t and a_t are the state and actions at time t, after starting from state $s_1 = s$ then deviating with an immediate action of $a_1 = a$ and following π onwards. We slightly abuse notation by writing $Q_{\pi,r}(s,\pi') = \mathbf{E}_{a\sim\pi'(a|s)}[Q_{\pi,r}(s,a)]$. These values satisfy the well known recurrence equation:

$$Q_{\pi,r}(s,a) = r(s,a) - \eta_r(\pi) + \mathbf{E}_{s'\sim P_{sa}}[Q_\pi(s',\pi)] \tag{9.1}$$

where $Q_\pi(s',\pi)$ is the next state value (without deviation).

If π^* is an optimal policy (with respect to η_r), then, as usual, we define $Q_r^*(s,a)$ to be the value of the optimal policy, *ie* $Q_r^*(s,a) = Q_{\pi^*,r}(s,a)$.

We now provide two useful lemmas. It is straightforward to see that the previous assumption implies a rate of convergence to the stationary distribution that is $O(\tau)$, for all policies. The following Lemma states this more precisely.

Lemma 9.2. *For all policies π,*

$$\|d_{\pi,t} - d_\pi\|_1 \leq 2e^{-t/\tau}\,.$$

Proof. Since π is stationary, we have $d_\pi P^\pi = d_\pi$, and so

$$\|d_{\pi,t} - d_\pi\|_1 = \|d_{\pi,t-1}P^\pi - d_\pi P^\pi\|_1 \leq \|d_{\pi,t-1} - d_\pi\|_1 e^{-1/\tau}$$

which implies $\|d_{\pi,t} - d_\pi\|_1 \leq \|d_1 - d_\pi\|_1 e^{-t/\tau}$. The claim now follows since, for all distributions d and d', $\|d - d'\|_1 \leq 2$. □

The following Lemma derives a bound on the Q values as a function of the mixing time.

Lemma 9.3. *For all bounded reward functions r and policies π, $Q_{\pi,r}(s,a) \leq 3\tau$.*

Proof. First, let us bound $Q_{\pi,r}(s,\pi)$, where π is used on the first step. For all t, including $t = 1$, let $d_{\pi,s,t}$ be the state distribution at time t starting from state s and following π. Hence, we have

$$
\begin{aligned}
Q_{\pi,r}(s,\pi) &= \sum_{t=1}^{\infty} \left(E_{s'\sim d_{\pi,s,t}, a\sim\pi}[r(s',a)] - \eta_r(\pi) \right) \\
&\leq \sum_{t=1}^{\infty} \left(E_{s'\sim d_{\pi}, a\sim\pi}[r(s',a)] - \eta_r(\pi) + 2e^{-t/\tau} \right) \\
&= \sum_{t=1}^{\infty} 2e^{-t/\tau} \leq \int_0^{\infty} 2e^{-t/\tau} = 2\tau
\end{aligned}
$$

Using the recurrence relation for the values, we know $Q_{\pi,r}(s,a)$ could be at most 1 more than the above. The result follows since $1 + 2\tau \leq 3\tau$ □

9.2 Best Expert Algorithms

Best expert algorithms have played an important rule in Computer science in the last decade solving problems varying from classification to on-line portfolios [67, 14]. Since in this chapter we only use them as a black box, we first provide our assumption on their performance and later for completeness provide the weighted majority algorithm [67, 39].

Assumption 9.4. *(Black Box Experts) An optimized best expert algorithm is an algorithm which guarantees that for any sequence of loss functions $c_1, c_2, \ldots c_T$ over actions A, the algorithm selects a distribution q_t over A (using only the previous loss functions $c_1, c_2, \ldots c_{t-1}$) such that*

$$\sum_{t=1}^{T} \mathbf{E}_{a\sim q_t}[c_t(a)] \leq \sum_{t=1}^{T} c_t(a) + \sqrt{TM \log |A|},$$

where $\|c_t(a)\| \leq M$. Furthermore, we also assume that decision distributions do not change quickly:

$$\|q_t - q_{t+1}\|_1 \leq \sqrt{\frac{\log |A|}{t}}$$

Next we provide the weighted majority algorithm , which stratify this assumption the algorithm is originally due to [67, 39], the WM algorithm that we present here require knowing the horizon time T in advance, however there exists other variants such as doubling [39] or changing that learning rate [7] that achieve the desired regret bounds without knowing T in advance.

Weighted Majority (T)
Choose an initial distribution P_1;
for $t = 1$ *to* T **do**
  Update $P_{t+1}(a) = P_t(a)\beta^{c_t(a)}/Z_t$, where $Z_t = \sum_a P_t(a)\beta^{c_t(a)}$;
end

Proposition 9.5. *For the weighted majority algorithm with appropriate β, we have*

$$E[\textit{Cost Wighted Majority at time } T] \leq \textit{Best expert at time } T + O(\sqrt{M \log |A|}T).$$

A different flavor of an best expert algorithm, which does not satisfy Assumption 9.4 but can be much more efficient, was introduced by [55, 48]. The Follow the perturbed leader (FPL) algorithm works for problems that satisfy (1) An oracle (efficient method) which finds the optimum of the static problem, (2) The cost function is additive. The setting is defined as follows, a decision set such that for any two decisions we have $\|d - d'\|_1 \leq D$ (in the former setting each decision was associated with an expert), state set S, cost function $c(d, s) \leq R$; the cost of each state is revealed at the end of each timestep.

FPL
for $t = 1$ *to* ∞ **do**
  Choose an Choose p_t uniform at random in $[0, |S|\sqrt{t}]^{|S|}$;
  Use an oracle M to find the optimal decision for cost function $\sum_{i=1}^{t} c_t + p_t$;
  Use the optimal decision at time $t + 1$;
end

The following Proposition bounds the regret of FPL algorithm.

Proposition 9.6. *The FPL algorithm satisfies*

$$E[\textit{Cost of FPL at time } T] \leq c(d, s_1 + ... + s_T) + 2\sqrt{DR|S|T},$$

for every fixed decision d.

The major advantage of the FPL algorithms is that it can handle an exponential number of experts as long as the static problem can solved efficiently as is evident in the shortest path problem. In the shortest path problem an agent is given a graph with two special nodes s, t in each time step the agent chooses a path between s and t and weight edges are revealed. The agent wants to minimize the average distance and the regret is measured with respect to optimal static path, which can be computed easily using Dijksta algorithm, although the number of paths might be exponential and applying WM is unrealistic.

9.3 Idealized Settings

Every algorithm in the on-line MDP faces two major obstacles. The first is the computational issue since there is an exponential number of polices; the second is "state problem", the current state (distribution over the states) depends on the algorithm past and therefore the current reward depends on both the current action and the past, while the common assumption in *every* best expert algorithm is that the current reward is independent of the past. In this section we concentrate only on the first problem and define an idealized setting in which the second problem is irrelevant. In the idealized setting in *every* time step the algorithm chooses a policy and observes its average reward, $\eta_r(\pi)$. Therefore, the algorithm does not have a "state" anymore and the current reward is not effected by the previous decisions.

We start by describing and analyzing the naive approach. The naive algorithm uses a best expert algorithm satisfying Assumption 9.4, where each deterministic policy is an expert and the loss function for a policy π at time t is $\eta_{r_t}(\pi)$.

Since there is no state in the idealized setting we can use directly the standard best experts algorithms, with $|A|^{|S|}$ experts where the reward $\eta_r(\pi)$ is bounded by one. Substituting these numbers in the best expert performance guarantees we obtain the following Proposition.

FPL in MDPs
for $t = 1$ *to* ∞ **do**
  Choose p_t uniform at random in $[0, |S||A|\sqrt{t}]^{|S||A|}$;
  Calculate the optimal policy, π_t for the MDP with reward function $r = \sum_{i=1}^{t} r_t + p_t$;
  Use π_t at time $t+1$;
end

Proposition 9.7. *Let A be the naive algorithm in the idealized settings. Then for all reward functions $r_1, r_2, \ldots r_T$ and for all stationary policies π,*

$$\frac{1}{T}\sum_{t=1}^{T} \eta_{r_t}(\pi_t) \geq \frac{1}{T}\sum_{t=1}^{T} \eta_{r_t}(\pi) - \sqrt{|S| \log |A|/T},$$

where π_t is the policy produced by A at time t. Also the naive algorithm has space complexity $O(|A|^{|S|})$ and in each time step it makes $O(|A|^{|S|})$ to calculate π_t.

The naive approach is clearly infeasible because of the space and time complexity. However, the MDP has a structure and therefore one can exploit them to give efficient algorithms. The next algorithm is the FPL algorithm adapted to MDPs.

The analysis of this algorithm in MDPs was also done independently by [78].

Proposition 9.8. *For FPL in MDPs in the idealized settings, for all reward functions $r_1, r_2, \ldots r_T$ and for all stationary policies π,*

$$\frac{1}{T}\sum_{t=1}^{T} \eta_{r_t}(\pi_t) \geq \frac{1}{T}\sum_{t=1}^{T} \eta_{r_t}(\pi) - \sqrt{|S||A|/T},$$

where π_t is the policy produced by FPL at time t. Also the space complexity $O(|A||S|)$ and in each time step it runs in time polynomial in $|A|$ and $|S|$ to calculate π_t.

Proof. The FPL algorithm requires the cost function to be additive and it its complexity as the complexity of calculating the optimal static problem. We first show that the cost function is additive, i.e., $\eta_{\sum_{i=1}^t r_t}(\pi) = \sum_{i=1}^t \eta_{r_t}(\pi)$.

$$\begin{aligned}
\sum_{i=1}^t \eta_{r_t}(\pi) &= \sum_{i=1}^t \sum_{s\in S} d_\pi(s)\pi(a|s)r(s,a) \\
&= \sum_{s\in S} \sum_{i=1}^t d_\pi(s)\pi(a|s)r(s,a) \\
&= \sum_{s\in S} d_\pi(s)\pi(a|s) \sum_{i=1}^t r(s,a) \\
&= \eta_{\sum_{i=1}^t r_t}(\pi).
\end{aligned}$$

Next we would like to calculate the algorithm parameters, we know that the cost function is bounded by 1, we also have that for any two policies π, π', $\|d_\pi - d_{\pi'}\| \leq 2$, and that the number of the algorithm states is $|S||A|$. Substituting in Proposition 9.6 gives the desired bound. The space complexity is $|S||A|$ since we only need to store the cumulative reward for every state action pair. The time complexity of solving the static problem is that of solving an MDP and it can be done in time polynomial in $|S|$ and $|A|$. □

Next we present our algorithm which is as efficient as the FPL algorithm and its regret is slightly better. In contrary to FPL this algorithm is not general and it exploits directly the MDP structure. Later, we show that this algorithm translates to a good algorithm in the general setting as well. The algorithm is intuitive and simple to describe. The algorithm uses best expert algorithm which satisfy Assumption 9.4, but instead of using them in the policy level as in the naive approach it associates each state with a best expert algorithm where each expert is an action. The policy is defined by the product of all best expert algorithms distributions. The immediate question is what loss function should we feed to each expert. It turns out Q_{π_t,r_t} is appropriate. Note that although the best expert algorithm are "local" they contain "global" information by using Q_{π_t,r_t}.

Theorem 9.9. *For all sequences $r_1, r_2, \ldots r_T$, the MDP experts algorithm have the*

MDP-E
Put in every state a best expert algorithm B_s;
for $t = 1$ *to* ∞ **do**
  Let $a_t(s)$ be the distribution over action of B_s at time t;
  Let π_t be $\pi_t(s) = a_t(s)$;
  Use policy π_t and obtain r_t from the enviornement;
  Feed B_s with gain function $Q_{\pi_t,r_t}(s,\cdot)$
end

Algorithm 12: MDP Expert algorithm for Idealized settings

following performance bound. For all π*,*

$$\sum_{t=1}^{T} \eta_{r_t}(\pi_t) \geq \sum_{t=1}^{T} \eta_{r_t}(\pi) - \sqrt{3\tau T \log |A|}$$

where $\pi_1, \pi_2, \ldots \pi_T$ *is the sequence of policies generated by* $\mathcal{A}$ *in response to* $r_1, r_2, \ldots r_T$*.*

Before proving the Theorem, we provide a technical lemma, which is a variant of a result in [54]

Lemma 9.10. *For all policies* π *and* π'*,*

$$\eta_r(\pi') - \eta_r(\pi) = \mathbf{E}_{s\sim d_{\pi'}}[Q_{\pi,r}(s,\pi') - Q_{\pi,r}(s,\pi)]$$

Proof. Note that by definition of stationarity, if the state distribution is at $d_{\pi'}$, then the next state distribution is also $d_{\pi'}$ if π' is followed. More formally, if $s \sim d_{\pi'}$, $a \sim \pi'(a|s)$, and $s' \sim P_{sa}$, then $s' \sim d_{\pi'}$. Using this and equation 9.1, we have:

$$\begin{aligned}
\mathbf{E}_{s\sim d_{\pi'}}[Q_{\pi,r}(s,\pi')] &= \mathbf{E}_{s\sim d_{\pi'},a\sim\pi'}[Q_{\pi,r}(s,a)] \\
&= \mathbf{E}_{s\sim d_{\pi'},a\sim\pi'}[r(s,a) - \eta_r(\pi) + \mathbf{E}_{s'\sim P_{sa}}[Q_\pi(s',\pi)] \\
&= \mathbf{E}_{s\sim d_{\pi'},a\sim\pi'}[r(s,a) - \eta_r(\pi)] + \mathbf{E}_{s\sim d_{\pi'}}[Q_\pi(s,\pi)] \\
&= \eta_r(\pi') - \eta_r(\pi) + \mathbf{E}_{s\sim d_{\pi'}}[Q_\pi(s,\pi)]
\end{aligned}$$

Rearranging terms leads to the result. □

MDP-E
Put in every state a best expert algorithm B_s;
for $t = 1$ *to* ∞ **do**
  Let $a_t(s)$ be the distribution over action of B_s at time t;
  Let π_t be $\pi_t(s) = a_t(s)$;
  Use policy π_t and obtain r_t from the enviornement;
  Feed B_s with gain function $Q_{\pi_t,r_t}(s,\cdot)$
end

Algorithm 13: MDP Expert algorithm for General settings

The Lemma shows why our choice to feed each experts algorithm Q_{π_t,r_t} was appropriate. Now we complete the proof of the above theorem.

Proof of Theorem 9.9. Using the assumed regret in assumption 9.4,

$$\begin{aligned}
\sum_{t=1}^{T}\eta_{r_t}(\pi) - \sum_{t=1}^{T}\eta_{r_t}(\pi_t) &= \sum_{t=1}^{T}\mathbf{E}_{s\sim d_\pi}[Q_{\pi_t,r_t}(s,\pi) - Q_{\pi_t,r_t}(s,\pi_t)] \\
&= \mathbf{E}_{s\sim d_\pi}[\sum_{t=1}^{T}Q_{\pi_t,r_t}(s,\pi) - Q_{\pi_t,r_t}(s,\pi_t)] \\
&\leq \mathbf{E}_{s\sim d_\pi}[\sqrt{3\tau T\log A}] \\
&= \sqrt{3\tau T\log A}
\end{aligned}$$

where we used the fact that d_π does not depend on the time in the second step. $\square$

9.4 The General Setting

Now we provide our main result showing how to use any generic experts algorithm in the general setting, where the current reward is influenced by the previous policies.

While in the previous section we use only part of Assumption 9.4, as we shall see, it is important that this 'slow change' condition be satisfied in the general setting. Intuitively, our experts algorithms will be using a similar policy for significantly long periods of time.

Note that although we moved form the idealized settings to the general settings and our target function had changed our algorithm remaind unchanged. We now state our main theorem.

Theorem 9.11. *Let $\mathcal{A}$ be the MDP experts algorithm. Then for all reward functions $r_1, r_2, \ldots r_T$ and for all stationary policies π,*

$$V_{r_1,r_2,\ldots r_T}(\mathcal{A}) \geq V_{r_1,r_2,\ldots r_T}(\pi) - 8\tau^2\sqrt{\frac{\log|A|}{T}} - \sqrt{\frac{3\tau\log|A|}{T}} - \frac{4\tau}{T}$$

As expected, the regret goes to 0 at the rate $O(1/\sqrt{T})$, as is the case with experts algorithms. Importantly, note that the bound does *not depend* on the size of the state space.

9.4.1 The Analysis

The analysis is naturally divided into two parts. First, we use the performance bounds of the algorithm in the idealized setting. Then we take into account the slow change of the policies to show that the actual performance is similar to the instantaneous performance.

Taking Mixing Into Account: This subsection relates the values V to the sums of average reward used in the idealized setting.

Theorem 9.12. *For all sequences $r_1, r_2, \ldots r_T$ and for all $\mathcal{A}$*

$$|V_{r_1,r_2,\ldots r_T}(\mathcal{A}) - \frac{1}{T}\sum_{t=1}^{T}\eta_{r_t}(\pi_t)| \leq 4\tau^2\sqrt{\frac{\log|A|}{T}} + \frac{2\tau}{T}$$

where $\pi_1, \pi_2, \ldots \pi_T$ is the sequence of policies generated by $\mathcal{A}$ in response to $r_1, r_2, \ldots r_T$.

Since the above holds for all $\mathcal{A}$ (including those $\mathcal{A}$ which are the constant policy π), then combining this with Theorem 9.9 (once with $\mathcal{A}$ and once with π) completes the proof of Theorem 9.11. We now prove the above.

The following simple Lemma is useful. It shows how close are the next state distributions when following π_t rather than π_{t+1}.

Lemma 9.13. *Let π and π' be such that $\|\pi(\cdot|s) - \pi'(\cdot|s)\|_1 \leq \epsilon$. Then for any state distribution d, we have $\|dP^{\pi} - dP^{\pi'}\|_1 \leq \varepsilon$.*

Proof. Consider the case when d is a delta function on s. The difference in the next state distributions, $\|dP^{\pi} - dP^{\pi'}\|_1$, is:

$$\begin{aligned}
\sum_{s'} |[P^{\pi}]_{s,s'} - [P^{\pi'}]_{s,s'}| &= \sum_{s'} \sum_{a} |P_{s,a}(s')(\pi(a|s) - \pi'(a|s))| \\
&\leq \sum_{s',a} P_{s,a}(s')|\pi(a|s) - \pi'(a|s)| \\
&= \sum_{a} |\pi(a|s) - \pi'(a|s)| \leq \varepsilon
\end{aligned}$$

where we have used the Assumption 9.4 in the last equation. Linearity of expectation leads to the result for arbitrary d. □

Analogous to the definition of $d_{\pi,t}$, we define $d_{\mathcal{A},t}$

$$d_{\mathcal{A},t} = \Pr[s_t = s | d_1, \mathcal{A}]$$

which is the probability that the state at time t is s given that $\mathcal{A}$ has been followed.

Lemma 9.14. *Let $\pi_1, \pi_2, \ldots \pi_T$ be the sequence of policies generated by $\mathcal{A}$ in response to $r_1, r_2, \ldots r_T$. We have*

$$\|d_{\mathcal{A},t} - d_{\pi_t}\|_1 \leq 2\tau^2 \sqrt{\frac{\log |A|}{t}} + 2e^{-t/\tau}$$

Proof. Let $k \leq t$. Using our experts assumption, it is straightforward to see that that the change in the policy over k steps is $|\pi_k(\cdot|s) - \pi_t(\cdot|s)|_1 \leq (t-k)\sqrt{\log |A|/t}$. Using this with $d_{\mathcal{A},k} = d_{\mathcal{A},k-1} P(\pi_k)$ and $d_{\pi_t} P^{\pi_t} = d_{\pi_t}$, we have

$$\begin{aligned}
\|d_{\mathcal{A},k} - d_{\pi_t}\|_1 &= \|d_{\mathcal{A},k-1} P^{\pi_k} - d_{\pi_t}\|_1 \\
&\leq \|d_{\mathcal{A},k-1} P^{\pi_t} - d_{\pi_t}\|_1 + \|d_{\mathcal{A},k-1} P^{\pi_k} - d_{\mathcal{A},k-1} P^{\pi_t}\|_1 \\
&\leq \|d_{\mathcal{A},k-1} P^{\pi_t} - d_{\pi_t} P^{\pi_t}\|_1 + 2(t-k)\sqrt{\log |A|/t} \\
&\leq e^{-1/\tau} \|d_{\mathcal{A},k-1} - d_{\pi_t}\|_1 + 2(t-k)\sqrt{\log |A|/t}
\end{aligned}$$

where we have used the last lemma in the third step and our contraction assumption 9.1 in the second to last step. Recursing on the above equation leads to:

$$\begin{aligned}
\|d_{\mathcal{A},t} - d_{\pi_t}\| &\leq 2\sqrt{\log |A|/t} \sum_{k=t}^{2} (t-k) e^{-(t-k)/\tau} + e^{-t/\tau} \|d_1 - d_{\pi_t}\| \\
&\leq 2\sqrt{\log |A|/t} \sum_{k=1}^{\infty} k e^{-k/\tau} + 2e^{-t/\tau}
\end{aligned}$$

The sum is bounded by an integral from 0 to ∞, which evaluates to τ^2. □

We are now ready to complete the proof of Theorem 9.12.

Proof. By definition of V,

$$\begin{aligned}
V_{r_1, r_2, \ldots r_T}(\mathcal{A}) &= \frac{1}{T} \sum_{t=1}^{T} E_{s \sim d_{\mathcal{A},t}, a \sim \pi_t}[r_t(s,a)] \\
&\leq \frac{1}{T} \sum_{t=1}^{T} E_{s \sim d_{\pi_t}, a \sim \pi_t}[r_t(s,a)] + \frac{1}{T} \sum_{t=1}^{T} \|d_{\mathcal{A},t} - d_{\pi_t}\|_1 \\
&\leq \frac{1}{T} \sum_{t=1}^{T} \eta_{r_t}(\pi_t) + \frac{1}{T} \sum_{t=1}^{T} \left(2\tau^2 \sqrt{\frac{\log |A|}{t}} + 2e^{-t/\tau} \right) \\
&\leq \frac{1}{T} \sum_{t=1}^{T} \eta_{r_t}(\pi_t) + 4\tau^2 \sqrt{\frac{\log |A|}{T}} + \frac{2\tau}{T}
\end{aligned}$$

where we have bounded the sums by integration in the second to last step. A symmetric argument leads to the result. □

Chapter 10

Learning in POMDPs

This Chapter is based on [29].

In this chapter, we address the problem of lifelong learning in a partially observable Markov decision process (a POMDP). We consider the most general setting where an agent begins in an unknown POMDP and desires to obtain near optimal reward. In this setting, the agent is forced to obey the dynamics of the environment, which, in general, do not permit resets.

The problem of lifelong learning has been well studied for observable MDPs. [59] provide the E^3 algorithm, which has finite (polynomial) time guarantees until the agent obtains near optimal reward. Unfortunately, such an algorithm is not applicable in the more challenging POMDP setting. In fact, none of the guarantees in the literature for learning in the limit for MDPs apply to POMDPs, for reasons which are essentially due to the partially observability.

Most of the existing learning algorithms such as [86, 87] either assume a goal state or assume a reset button. In fact, to the best of our knowledge, the literature does not contain even asymptotic results for general POMDPs which guarantee that the average reward of an agent will be near optimal in the limit.

Part of the technical difficulty is that there are currently no general results for belief state tracking with an approximate model showing that divergence in the belief state does not eventually occur. Crudely, the issue is that if a belief state is being tracked in an approximate manner, then it is important to show that this approximation quality does not continually degrade with time — otherwise the agent will eventually loose track of the belief state in the infinite horizon (this of course is not an issue in an MDP where the current state is observable). [18] address the issue of approximate belief state tracking, but in their setting the model is known perfectly and their goal is to keep a compact representation of the belief state. Note that approximate belief state tracking is much simpler if the agent is only acting over a fixed finite horizon, since then one can bound the error accumulation as a function of the horizon.

We present new algorithms for learning in POMDPs which guarantee that the agent will obtain the optimal average reward in the limit. Furthermore, we provide a finite time convergence rates for one of our algorithms which has an *exponential* dependence on a certain horizon time (of an optimal strategy) but has *no dependence* on the number of states in the POMDP. This result is reminiscent of the trajectory tree algorithm of [56] which has similar dependencies (though there they assumed access to a generative model, which allowed simulation of the POMDP). Given the plethora of complexity results in the literature on planning in POMDPs (see [71]), we feel these dependencies are the best one could hope for in the most general setting.

Central to our algorithms is the implementation of an approximate reset strategy or a *homing strategy*. The idea of a reset strategy is not new to the literature — homing sequences were used in the learning of deterministic finite automata (see [89], though there the sequence provided *exact* resets). Here, the agent follows a homing strategy in order to move *approximately* towards a reset. We show that such a strategy always exists, and our finite convergence rates also depend on a characteristic time it takes to approximately reset. However, note that existence of such a strategy alone does not imply that such a strategy will be useful. The reason is that the agent must take actions to reset, which might otherwise be better spent exploring or exploiting. It turns out that our algorithms use the homing strategy while *both* exploring and exploiting. In fact, they use the homing strategies infinitely often, which, unfortunately, detracts from exploiting. However, we are able to show that the ratio of the time these homing strategies are used compared to the time spent exploiting is decreasing sufficiently rapidly, such that near optimal average reward can be obtained.

10.1 Homing Strategies

Clearly, having an action which resets the agent to some designated state would be useful, as it would allow us to test and compare the performance of various policies, starting at the same start state. However, in general, such an action is not at our disposal.

Instead our algorithms utilize an approximate reset, which we show always exists. There are a few subtle points when designing such a reset. First, we must select actions to achieve the approximate reset, *i.e.* the approximate reset is done through the use of a *homing strategy*. Hence, while homing, the agent is neither exploring nor exploiting. Second, rather than moving to a fixed state, the homing strategy can only hope to move to toward a fixed (unknown) belief state. Third, as we shall see, since the POMDP might be periodic the stopping time must be a random variable[1]. To implement this randomized stopping time, we introduce *fictitious* 'stay' actions, in which the agent does not take an action that period. By this, we mean that if the homing strategy decides to take a 'stay' action at some time — which may not be possible if the true POMDP does not permit 'stay' actions — then the agent just ignores this 'stay' action and obtains another action from the homing strategy to execute. Hence, after the agent has taken t *homing actions* (which are either real or 'stay' actions), the agent has taken $t-m$ real actions in the POMDP and m stay actions. We now define an approximate reset strategy.

Definition 10.1. H is an (ϵ, k)-approximate reset (or homing) strategy if for every two belief states B_1 and B_2, we have $\|H_E(B_1) - H_E(B_2)\|_1 \leq \varepsilon$, where $H_E(B)$ is the expected belief state reached from B after k homing actions of H (so at most k real actions have been taken) and $H(B)$ is a random variable such that $H_E(B) =$

[1]Without randomizing over the stopping times (*i.e.* allowing 'stay' actions), the state transition matrix may be periodic and no stationary distribution may exist, *e.g.* if the states deterministically alternate between states 1 and 2.

$E_{H(B)\sim H,B}[H(B)]$.

The above definition only states that H will approximately reset, but this approximation quality could be poor. We now show how to amplify the accuracy of an approximate homing strategy, and then we show that an approximate homing strategy always exists.

Lemma 10.2. *Suppose that H is an (ε, k) approximate reset then H^ℓ is an $(\varepsilon^\ell, k\ell)$ approximate reset, where H^ℓ consecutively implements H for ℓ times. Furthermore, this implies there exists a unique belief state B_H such that $H_E(B_H) = B_H$.*

Proof. The proof is a standard contraction argument, and we use induction. For $l = 1$, the claim follows by definition. Assume now that $\|H_E^{\ell-1}(B_1) - H_E^{\ell-1}(B_2)\|_1 \leq \varepsilon^{\ell-1}$. Let $H_E^{l-1}(B_1) = Q_1$ and $H_E^{l-1}(B_2) = Q_2$, so $\sum_s |Q_1(s) - Q_2(s)| \leq \varepsilon^{\ell-1}$. For an arbitrary state s', and using the fact that H is a linear operator, we have

$$\begin{aligned}
&\|H_E^\ell(B_1) - H_E^\ell(B_2)]\|_1 \\
&\quad = \|H_E(Q_1) - H_E(Q_2)]\|_1 \\
&\quad = \|\sum_s (Q_1(s) - Q_2(s)) H_E(s)\|_1 \\
&\quad = \|\sum_s (Q_1(s) - Q_2(s)) H_E(s')\|_1 \\
&\qquad + \|\sum_s (Q_1(s) - Q_2(s))(H_E(s) - H_E(s'))\|_1 \\
&\quad = 0 + \sum_s |Q_1(s) - Q_2(s)|\varepsilon \\
&\quad \leq \varepsilon^\ell
\end{aligned}$$

where the first term is 0 since for any two distributions $\sum_s Q_1(s) - Q_2(s) = 1 - 1 = 0$ (and the vector $H(s')$ is constant in this sum), and we have used the fact that $\|H(s) - H(s')\|_1 \leq \varepsilon$ (by definition of H). □

We now show that the random walk strategy (including 'stay' actions) is an approximate reset strategy in every POMDP (including periodic ones), though with prior knowledge we might have better approximate reset strategies at our disposal.

Lemma 10.3. *For all POMDPs, the random walk strategy (using 'stay' actions) constitutes an* (ε, k) *approximate reset strategy for some* $k \geq 1$ *and* $0 < \varepsilon < \frac{1}{2}$.

Proof. By our connectivity assumption, for all states s and s', there exists some strategy that reaches s' from s with positive probability. This implies that under H (the random walk strategy), there is positive probability of moving from one state to another, *i.e.* the Markov chain is irreducible. Furthermore, since H performs 'stay' actions, then the Markov chain is aperiodic. Thus, there exists a unique stationary distribution. We choose k to be the time at which the error in convergence is less than 1/2, from all starting states. Hence, by linearity of expectation, the error is less than 1/2 from all belief states in k steps. □

10.2 Reinforcement Learning with Homing

We now provide two algorithms which demonstrate how near-optimal average reward can be obtained, with different rates of convergence. The key to the success of these algorithms is their use of homing sequences in *both* exploration and in exploitation. For exploration, the idea is that each time we attempt some exploration trajectory we do it after implementing our reset strategy — hence our information is (approximately) grounded with respect to the belief state B_H (recall $H_E(B_H) = B_H$). The idea of exploration is to find a good t-Markov strategy $\hat{\pi}^*_t$ from B_H. During exploitation, the goal is to use this t-Markov strategy. Unfortunately, we have only guaranteed that $\hat{\pi}^*_t$ performs well *starting from* B_H and only for t steps. Hence, after each time we exploit with $\hat{\pi}^*_T$, we run our homing sequence to get back close to B_H (and then we rerun $\hat{\pi}^*_t$). We gradually increase t in the process.

The problem is that while homing, we are wasting time and neither exploiting nor exploring. Furthermore, since we use the homing sequence between *every* run of $\hat{\pi}^*_t$, asymptotically we never stop homing. Nonetheless, we able to show that there exists an algorithm which obtains near optimal reward in a POMDP, since the ratio of the time spent exploiting vs. homing decreases sufficiently fast.

```
Input  : H /*a (1/2, K_H) approximate reset strategy */
for t = 1 to ∞ do
    /*Exploration in Phase t */;
    k_1^t = O( (1/ε_t^2) log(t^2 |Π_t|) );
    foreach Policy π in Π_t do
        for i = 1 to k_1^t do
            Run π for t steps;
            Repeatedly run H for log(1/ε_t) times;
        end
        Let v^π be the average return of π in from these k_1^t trials;
    end
    /*Exploitation in Phase t */;
    Let π̂_t^* = arg max_{π∈Π_t} v^π;
    k_2^t = O( (1/ε_t)([current time T]
             +[time in t + 1-th exploration phase]) );
    for i = 1 to k_2^t do
        Run π̂_t^* for t steps;
        Repeatedly run H for log(1/ε_t) times;
    end
end
```

Algorithm 14: Policy Search

Theorem 10.4. *There exists an algorithm $\mathcal{A}$, such that in any connected POMDP, $\mathcal{A}$ obtains the optimal average reward in the limit, with probability* 1.

We later provide an algorithm with a better convergence rate (see Theorem 10.8). However, we start with a simpler policy search algorithm which establishes the above Theorem.

10.2.1 Policy Search

Algorithm 14 takes as input a $(1/2, K_H)$-approximate reset strategy, which could be the random walk strategy with a very crude reset. The algorithm works in phases, interleaving exploration phases with exploitation phases. Let us start by describing the exploration phases. Let Π_t be the set of all t-Markov strategies. An estimate of the value of a policy $\pi \in \Pi_t$ can be found by first resetting and then running π for

t steps. The exploration phase consists of obtaining an estimate v^π of the return of each policy $\pi \in \Pi_t$, where each estimate consists of an average of k_1 trials (followed by approximate resets).

These estimates have both bias and variance. The variance is just due to the stochastic nature of the POMDP. The bias is due to the fact that we never can exactly reset to B_H. However, if we run H for $\log 1/\varepsilon_t$ times (where ε_t is an error parameter in the t-th phase, which will be fixed latter) then, by lemma 10.2, all expected belief states we could approach will be $(1/2)^{\log 1/\varepsilon_t} = \varepsilon_t$ close to B_H. The following lemma shows that accurate estimates can be obtained.

Lemma 10.5. *In phase t, if $k_1^t = O\left(\frac{1}{\varepsilon_t^2}\log(t^2|\Pi_t|)\right)$ and each reset consists of using the homing sequence $\log 1/\varepsilon_t$ times, then for all policies $\pi \in \Pi_t$, the estimated t-horizon reward, v^π, satisfies $|R_t^\pi(B_H) - v^\pi| \le 2\varepsilon_t$ with probability greater than $1 - \frac{1}{2t^2}$.*

Proof. First, let us deal with the bias. Any expected belief states b that results from using the homing sequence $\log 1/\varepsilon_t$ times must satisfy $\|b - B_H\| \le (1/2)^{\log 1/\varepsilon_t} \le \varepsilon_t$. Now it is straightforward to see that if b and B_H are belief states such that $\|b - B_H\|_1 \le \varepsilon$, then for every strategy π, $|R_t^\pi(b) - R_t^\pi(B_H)| \le \varepsilon$. To see this, let the belief states at time t be b^t and B_H^t, which result from following either π starting from b or B_H, respectively. By linearity of expectation, it follows that $\|b^t - B_H^t\|_1 \le \varepsilon$. This directly implies that $|R_t^\pi(b) - R_t^\pi(B_H)| \le \varepsilon$.

For the variance, the Hoeffding bound and our choice of k_1^t imply that the average return of each policy is ε_t close to its expectation (the expectation is both over the initial state and on the policy trajectory) with probability $1 - 1/(2t^2)$. □

Now during exploitation, the algorithm uses the policy $\hat{\pi}_t^*$ which had the highest return in the exploration phase (and by the previous lemma this is close to the policy with largest return). Note that we have only guaranteed a large return for executing $\hat{\pi}_t^*$ *from* B_H for t steps. However, we would like to exploit for a longer period of time than t. The key is that we again reset $\log(1/\varepsilon_t)$ times after each time we run $\hat{\pi}_t^*$, which resets us close ε_t close to B_H. Unfortunately, this means we spend $K_H \log 1/\varepsilon_t$ steps between *each* run of $\hat{\pi}^*$. Hence, our average return could be $O(\frac{1}{t}K_H \log 1/\varepsilon_t)$ less than we would like, since $O(\frac{1}{t}K_H \log 1/\varepsilon_t)$ is the fraction of time we spend resetting.

Note that this fraction could be large if we desire that ε_t be very small (thought this would guarantee very accurate resets).

Now, when we do exploit (and reset), we run the exploitation phase long enough (for k_2^t time) such that our *overall* average reward is comparable to the average reward in the last exploitation phase.

Lemma 10.6. *At any time T after phase t, the average reward from time 1 to time T satisfies: $\frac{1}{T}\sum_{i=1}^{T} r_i \geq R_t^*(B_H) - O(\varepsilon_t + \frac{1}{t}K_H \log(1/\varepsilon_t))$ with probability at least $1 - \frac{1}{t^2}$.*

Before proving this lemma, let us state a corollary from which Theorem 10.4 follows.

Corollary 10.7. *Let $\varepsilon_t = 1/t$. Then $\frac{1}{T}\sum_{i=1}^{T} r_i \geq R_t^*(B_H) - O(\frac{K_H \log(t)}{t})$ with probability at least $1 - \frac{1}{t^2}$.*

Importantly, note the loss term goes to 0 as t goes to infinity. Furthermore, for a large enough phase t, we know that $R_t^*(B_H)$ will approach the optimal average reward R^* (since R^* is independent of the starting B). Theorem 10.4 follows. Essentially, although we have to home infinitely often, the ratio of time spent homing to the time spent using our t-step exploitation policies is going as $O(\frac{\log t}{t})$, which goes to 0.

Proof. First, let us show that the average reward, $\frac{1}{T}\sum_{i=1}^{T} r_i \geq R_t^*(B_H)$, is no less than ε_t from the average reward obtained in the t-th exploitation phase. To do this, we set the time of exploitation phase, k_2^t, to be $1/\varepsilon_t$ times greater than previous amount of time spent in the MDP time plus the amount of time that will be spent in the next exploration phase (this latter factor accounts for the case in which time T lies in the exploration phase immediately after t).

Now we bound the average reward obtained in the exploitation phase. First. let us show that the t-average reward of the policy used, $R_t^{\hat{\pi}_t^*}$, satisfies $R_t^*(B_H) - R_t^{\hat{\pi}_t^*}(B_H) \leq 4\varepsilon_t$ with probability at least $1 - \frac{1}{2t^2}$. By Lemma 10.5 for each policy $\pi \in \Pi_t$, we have $|R_t^\pi(B_H) - v^\pi| \leq 2\varepsilon_t$ with probability at least $1 - \frac{1}{2|\Pi_t|t^2}$. Therefore, $\hat{\pi}_t^*$ is $4\varepsilon_t$-optimal with probability $1 - 1/(2t^2)$. Now the observed average return of $\hat{\pi}_t^*$ in the exploitation period is $2\varepsilon_t$ close to $R_t^*(B_H)$ with probability at least $1 - \frac{1}{2t^2}$, since our observed average return in exploitation is least as good as those used to find $v^{\hat{\pi}_t^*}$ (since $k_2^t > k_1^t$).

However, the average return during the exploitation phase is not the observed average return of $\hat{\pi}_t^*$, since we reset after each t exploitation steps for a number of steps that is $K_H \log(1/\varepsilon_t)$. The resets in the exploitation period can change the average reward by at most a fraction $\frac{1}{t} K_H \log(1/\varepsilon_t)$. □

10.2.2 A Model Based Algorithm

The previous algorithm was the simplest way to demonstrate Theorem 10.4. However, it is very inefficient, since it is testing all t-Markov policies — there are doubly exponential, in t, such polices[2]. Here, we provide a more efficient model based algorithm, which resembles the algorithms given in [56, 77], and is exponential in the horizon time, yet it still has no dependence on the number of states in the POMDP.

We now state a convergence rate in terms of τ, the ε-horizon time of an optimal policy and in terms of the homing time K_H (recall, such a time exists for every POMDP using a random walk policy).

Theorem 10.8. *There exists an algorithm $\mathcal{A}$, such that in any connected POMDP and with probability greater than $1-\delta$, $\mathcal{A}$ achieves an average reward that is 2ε close to the optimal average reward in a number of steps in the POMDP which is polynomial in $|A|$,$|O|$,K_H and $\log(1/\delta)$ and exponential in τ. Furthermore the computational runtime of this algorithm is polynomial in $|A|$,$|O|$, and $\log(1/\delta)$ and exponential τ.*

We provide such an algorithm in the next page. In exploration phase, the algorithm builds an approximate model of the transition probabilities after some history has occurred starting from B_H. In the t-th phase, it builds a model with respect to the set of all t-length histories, which we denote by $\mathcal{H}_t$. In the exploitation phase, it uses the best t Markov strategy with respect to this model. The use of homing strategies is similar to that in the previous algorithm.

[2] The number of histories of length t is exponential in t, and the number of t-Markov polices is exponential in the number of t-length histories

Input : H /*an $(1/2, K_H)$ approximate reset strategy */
Let $L = |A| \cdot |O|$;
for $t = 1$ *to* ∞ **do**
  $k_1^t = O\left(\frac{L^{4t}}{\varepsilon_t^2}\log(t^2|\mathcal{H}_t|)\right)$;
  for k_1^t *times* **do**
    Run `RANDOM` for t steps;
    Run H for $K_H \log(L^t/\varepsilon_t)$ steps.
  end
  for $h \in \mathcal{H}_t$, $a \in A$ *and* $o \in O$ **do**
    $\hat{\Pr}[o|h, B_0, a] = 0$;
    if $\hat{\Pr}[h(a,o)|B_0] \geq \frac{\varepsilon_t}{L^t}$ **then**
      $\hat{\Pr}[o|h, B_0, a] = \frac{\hat{\Pr}[h(a,o)|B_0]}{\hat{\Pr}[h|B_0]\hat{\Pr}[a]}$
    end
  end
  Compute $\hat{\pi}_t^*$ using $\hat{\Pr}[o|B_0, h, a]$;
  $k_2^t = O\Big(\frac{1}{\varepsilon_t}$([current time T]
      +[time in $t+1$-th exploration phase]$)\Big)$;
  for k_2^t *times* **do**
    Run $\hat{\pi}_t^*$ for t steps;
    Run H for $K_H \log(1/\varepsilon_t)$ steps;
  end
end

Algorithm 15: Model based

Let $L = |A||O|$, and note that $2L^t \geq |\mathcal{H}_t|$. In the exploration phase, the algorithm takes actions uniformly at random for t steps and then resets (running the homing strategy for $\log(L^t/\varepsilon_t)$ times). This is done k_1^t times.[3] Then using the empirical frequencies in these trajectories the algorithm forms estimates $\hat{\Pr}[o|h, B_H, a]$, which is just the empirical probability of observing o conditioned on history h followed by taking action a. For histories h which are unlikely, these empirical estimates could be very bad, though, as we shall see, we do not need accurate estimates of $\hat{\Pr}[o|h, B_H, a]$ for such histories. Let $h(a, o)$ be a history with h followed by (a, o).

[3]We can use in the algorithm any approximate homing strategy H. However, if H is simply the random policy, then the reset and exploration would both use the same policy, and the algorithm would slightly simplify.

Lemma 10.9. *In phase t, if $k_1^t = O\left(\frac{L^{4t}}{\varepsilon_t^2}\log(t^2|\mathcal{H}_t|)\right)$ and each reset consists of using the homing sequence $\log(L^t/\varepsilon_t)$ times, then: (1) $|\hat{\Pr}[h|B_0] - \Pr[h|B_0]| \leq \frac{\varepsilon_t}{L^{2t}}$, and (2) for every $h(a,o) \in \mathcal{H}_t$ such that $\Pr[h(a,o)|B_0] \geq \frac{\varepsilon_t}{L^t}$, we have $|\hat{\Pr}[o|h,B_0,a] - \Pr[o|h,B_0,a]| \leq \frac{2|A|}{L^t}$, with probability at least $1 - \frac{1}{t^2}$.*

Proof. We first note that, with probability $1 - 1/t^2$, for every history $h \in \mathcal{H}_t$ we have $|\hat{\Pr}[h|B_0] - \Pr[h|B_0]| \leq \frac{\varepsilon_t}{L^{2t}}$ (using the Hoeffding bound). The error, $|\Pr[o|h,B_0,a] - \hat{\Pr}[o|h,B_0,a]|$, is then

$$\begin{aligned}
&\left|\frac{\Pr[h(a,o)|B_0]}{\Pr[h|B_0]\hat{\Pr}[a]} - \frac{\hat{\Pr}[h(a,o)|B_0]}{\hat{\Pr}[h|B_0]\hat{\Pr}[a]}\right| \\
&\leq \frac{1}{\hat{\Pr}[a]}\left|\frac{\Pr[h(a,o)|B_0] + \frac{\varepsilon_t}{L^{2t}}}{\Pr[h|B_0] - \frac{\varepsilon_t}{L^{2t}}} - \frac{\Pr[h(a,o)|B_0]}{\Pr[h|B_0]}\right| \\
&= \frac{1}{\hat{\Pr}[a]}\left|\frac{\frac{\varepsilon_t}{L^{2t}}}{\Pr[h|B_0] - \frac{\varepsilon_t}{L^{2t}}} + \frac{2\Pr[h(a,o)|B_0]\frac{\varepsilon_t}{L^{2t}}}{\Pr[h|B_0](\Pr[h|B_0] - \frac{\varepsilon_t}{L^{2t}})}\right| \\
&\leq \frac{2|A|}{L^t},
\end{aligned}$$

where the first inequality holds with probability $1 - \frac{1}{t^2}$, and in the last inequality we used the fact that $\Pr[h|B_0] \geq \frac{\varepsilon_t}{L^t}$. □

The exploitation policy can be found using dynamic programming with the model. Note that the POMDP is equivalent to an MDP where the histories are states. In the exploitation phase, the algorithm uses the best t-Markov policy, $\hat{\pi}_t^*$, (with respect to the approximate model) interleaving it with $K_H \log(1/\varepsilon_t)$ homing steps.

Lemma 10.10. *In phase t, the exploitation policy $\hat{\pi}_t^*$, satisfies $|R_t^*(B_H) - R_t^{\hat{\pi}_t^*}(B_H)| \leq t(\varepsilon_t + \frac{2|A|}{L^t}) + (2\varepsilon_t + \frac{2\varepsilon_t}{L^t})$ with probability at least $1 - \frac{1}{t^2}$.*

Proof. (sketch) We observe that by ignoring all histories (which we view as nodes in a tree) such that $\hat{\Pr}(h|B_0) \leq \frac{\varepsilon_t}{L^t}$, the return of an optimal strategy in this empirical model is decreased by at most $2t(\varepsilon_t + \frac{\varepsilon_t}{L^t})$, due to the fact that the true history probability is bounded by $\frac{\varepsilon_t}{L^t} + \frac{\varepsilon_t}{L^{2t}}$, the return from each node is bounded by t and the total number of such nodes is bounded by $2L^t$. Next we prove that the return of the optimal policy in the empirical model loses at most $t^2(\varepsilon_t + \frac{2|A|}{L^t})$ due to the tree approximation on the other nodes (the other histories). Using backward induction, we show that the policy $\hat{\pi}_t^*$ has return not less than $k^2(\varepsilon_t + \frac{2|A|}{L^t})$ in comparison to the true optimal value, starting from $(t-k+1)$-length histories. The base case, for the leaves (the t-length histories), holds since the reward (which is encoded through the observations) is within $\varepsilon_t + \frac{2|A|}{L^t}$, where the first error is due to the imperfect reset and the second is due to the marginal distribution error that is bounded by $\frac{2|A|}{L^t}$ by Lemma 10.9. Assume the induction assumption holds for $k-1$. There are two sources of error, the first is due to the current estimation error (of both the marginal distribution and the immediate reward) which is bounded by $(\varepsilon_t + \frac{2|A|}{L^t})k$ and the second is due to errors from the previous levels and is bounded by $(k-1)^2(\varepsilon_t + \frac{2|A|}{L^t})$ by the induction assumption. Summing the terms completes the induction step. $\square$

Similarly to Subsection 10.2.1, we exploit long enough such that the overall average reward is essentially the average reward in the last exploitation period.

Lemma 10.11. *At any time T after phase t, the average reward from time 1 to time T satisfies: $\frac{1}{T}\sum_{i=1}^{T} r_i \geq R_t^*(B_H) - O(t\varepsilon_t + \frac{|A|t}{L^t} + (1/t)K_H \log(1/\varepsilon_t))$ with probability at least $1 - \frac{1}{t^2}$.*

Using the above lemma, Theorem 10.8 follows immediately if we set $\varepsilon_t = 1/t^2$.

Proof. (sketch) We first note that all exploitations and explorations from phases 1 to T and from the next, $(T+1)$-th, exploration phase can effect the average reward by at most ε_t. By Lemma 10.10, the exploitation policy is near optimal and satisfies, $|R_t^*(B_H) - R_t^{\hat{\pi}_t^*}(B_H)| \leq t(\varepsilon_t + \frac{2|A|}{L^t}) + (\varepsilon_t + \frac{\varepsilon_t}{L^t})$ with probability $1 - 1/(2t^2)$. As in Lemma 10.5, we observe that the bias of the exploitation policy is ε_t and the variance due to Hoeffding's bound and the large exploitation time is bounded by ε_t with probability $1-1/(2t^2)$. The last source for loss is the resets in the exploitation period and its effect can be bounded by $\frac{K_H \log(1/\varepsilon_t)}{t}$. $\square$

A direct and simple corollary from which Theorem 10.4 follows as well.

Corollary 10.12. *Let $\varepsilon_t = 1/t$. Then $\frac{1}{T}\sum_{i=1}^{T} r_i \geq R_t^*(B_H) - O(\frac{K_H \log(t)}{t})$ with probability at least $1 - \frac{1}{t^2}$.*

Chapter 11

planning in POMDPs

This chapter is based on [28]. Recently, an alternative and more concise representation for POMDPs, namely *Predictive State Representations* (PSRs) ([68, 96, 95]), has been proposed. The hope is that this representation will allow for more efficient approximation algorithms. In the PSR, the POMDP, "hidden state" structure is not explicitly represented and only the probabilities of future, "action - observation" sequences are specified.

In this work, we focus on the closely related representation of multiplicity automata, which were first introduced by [93] and were closely studied by [37, 22]. Multiplicity automata are generalizations of both deterministic and stochastic automata, and we show that they are also generalizations of the POMDPs.

The chapter first contribution is in showing that POMDPs are a special case of multiplicity automata. This is done through using a similar construction to the PSR construction. Similar to PSRs, we then show that a POMDP can be represented by multiplicity automaton of size r, where r is bounded by the number of the POMDP states. These results formally relate PSRs to a large extant literature on that of multiplicity automata, where many learning models have been considered (for instance, see [10]).

The chapter second contribution is to provide a planning algorithm which exploits this structure. In particular, we provide an algorithm which has runtime that is exponential in the multiplicity automata size rather than the number of states of the

POMDP. This result directly contributes to the extant PSR literature — essentially, in the PSR framework, our algorithm can be viewed as providing an approximate planing algorithm with runtime that is exponential in the PSR rank, which, to our knowledge, is a novel result in this literature.

Our algorithm differs from other grid based ones in that it uses a *modified belief MDP* and not the standard *belief MDP*. The *modified belief MDP* is built with respect to the multiplicity automaton construction. Since previous planning based algorithms have runtime which is exponential in the number of states of the POMDP, our algorithms provides a significant improvement whenever the multiplicity automata representation is logarithmic in the standard POMDP representation. In such cases, the planning problem becomes tractable. An example for POMDP that has a logarithmic PSR representation was given by [68] (the example is the POMDP extension of the factored MDP that was introduced by [94]).

11.1 Grid Based Algorithms for POMDPs

Grid based algorithms for POMDPs are a standard and useful technique in solving POMDPs. The grid based algorithms discretize the belief states in the Belief MDP into a finite set, and they differ by the way they discretize it. The discreteization is done by defining a mapping from the *belief states* to the grid states, i.e., $g : B \to G$, where G is the grid states. The grid mesh parameter is defined as $\delta = \sup_{b,b',g(b)=g(b')} \|b - b'\|_1$. Unfortunately, all grid based algorithms suffer from an exponential dependence on the number of states that is the grid is composed form $(\frac{1}{\delta})^n$ cells where δ is the grid mesh.

To give intuition we provide here a simple grid based algorithm and performance guarantees for it. Similar results can be found in the literature. First we discretize the *belief space* by having $(\frac{1}{\delta})^n$ grid states. Each state is a vector of the form $(\alpha_1, ..., \alpha_n)$, where $\alpha_i = m\delta$, $m \in \{0, 1, \ldots, 1/\delta\}$.

The MDP is constructed from these discretized states in the natural way. The transition probabilities are now just modified such that transition only occur between discretized belief states. We call the resulting MDP the δ-discretized MDP of the

POMDP. Under this modification, it is relatively straightforward to show the following theorem.

Theorem 11.1. *Let M be the δ-discretized MDP for a POMDP. Then the optimal policy in M is $\frac{2\delta}{(1-\gamma)^3}$-optimal policy in the POMDP.*

The following three simple technical lemma establish the proof.

Lemma 11.2. *Let x and y be any two belief states such that $\|x-y\|_1 \leq \delta$, then $|Q^*(x,a) - Q^*(y,a)| \leq \frac{\delta}{1-\gamma}$*

Proof. Consider any policy π and start it from state x and from state y. Since $\|x-y\|_1 \leq \delta$ the expected reward at time 1 differs by at most δ. Let x_t, y_t be the belief states at time t after starting at states x, y respectively and following policy π for t steps then $\|x_t - y_t\|_1 \leq \delta$. The proof is done by induction. The basis holds since $x_1 = x$ and $y_1 = y$. Assume the hypothesis holds for $t-1$ and prove for t, since $\|x_{t-1} - y_{t-1}\|_1 \leq \delta$, then after following π for one step w.p $1-\delta$ for both states we will have the same behavior, thus $\|x_t - y_t\|_1 \leq \delta$ and this implies that the reward difference is bounded by δ as well at time t. □

This leads to a standard results on the value attained by value iteration on aggregated states (see [12] page 350).

Lemma 11.3. *Let the grid G defined by a function $g: B \to G$, and suppose that for every b, b' such that $g(b) = g(b')$, $\|b - b'\|_1 \leq \delta$. If we take one state as representative in each cell and let $\tilde{V}^*$ be the optimal value in this MDP then we have*

$$|V^*(b) - \tilde{V}^*(g(b))| \leq \frac{\delta}{(1-\gamma)^2}$$

Next we use Lemma 5.3 from Chapter 5, this Lemma is also standard and relates the greedy policy loss to the value function approximation error (see [97, 12]). Now the proof of Theorem 11.1 follows, by Lemma 11.2 if the grid mesh is δ then the difference in Q^* in the same grid state is at most $\delta/(1-\gamma)$ and then we can simply apply Lemmas 11.3 and 5.3.

11.2 Multiplicity Automata

In this section, we define the multiplicity automata and provide examples of other automata which are special case of the multiplicity automata.

Starting with notation, let K be a field[1] (in this paper, we only use $K = \mathbb{R}$), Σ an alphabet, λ the empty string, and $f : \Sigma^* \to K$ be a function (here, Σ^* is the set of all strings comprised of elements from Σ). Let us now define the Hankel matrix, which corresponds to f (sometimes referred to as a formal series).

Definition 11.4. Associate with f an infinite Hankel matrix F, where each row is indexed by some $x \in \Sigma^*$ and each column is indexed by some $y \in \Sigma^*$. Let $F(x, y) = f(x \circ y)$, where $\circ$ denotes the concatenation operation.

Next we define multiplicity automata, which represents the function $f : \Sigma^* \to K$.

Definition 11.5. A multiplicity automaton A of size r defined with respect to a field K, an alphabet Σ, and an empty string λ consists of:

- a set of matrices $\{\mu_\sigma : \sigma \in \Sigma\}$, each of size $r \times r$ where $\mu_\sigma(i, j) \in K$.
- an r tuple $\gamma = (\gamma_1, \ldots, \gamma_r) \in K^r$.
- a mapping μ which maps strings in Σ^* to $r \times r$ matrices. More precisely, $\mu(\lambda) = I$ (where I is the identity matrix), and, for $w \in \Sigma^*$, $\mu(w) = \mu_{\sigma_1} \cdot \mu_{\sigma_2} \cdots \mu_{\sigma_n}$.
- a function $f_A(w) = [\mu(w)]_1 \cdot \gamma$, where $[\mu(w)]_i$ is the ith row of $\mu(w)$.

We now provide a concrete example of a deterministic automaton and show its multiplicity automaton representation.

The multiplicity automaton of Figure 11.1 is of size 3 and its matrices are:

$$\mu_a = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \mu_b = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

[1] Recall, a field is an algebraic structure in which the operations of addition, subtraction, multiplication, and division (except division by the zero element) are permitted, and the associative, commutative, and distributive rules hold.

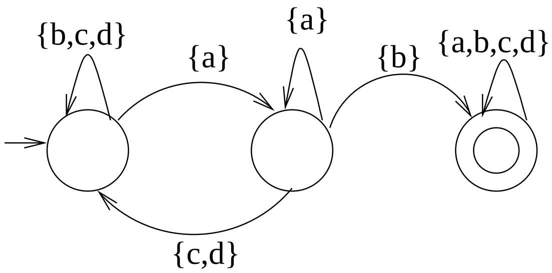


Figure 11.1: A deterministic automaton over $\Sigma = \{a, b, c, d\}$ that accepts a string only if it contains the substring ab, the accepting state is the rightmost state.

$$\mu_c = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \mu_c = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Set $\gamma = (0, 0, 1)$, since the rightmost state is the only accepting state.

We extend the previous example and show that probabilistic automata in general are a special case of multiplicity automata. Let the field be the reals and define $[\mu_\sigma]_{i,j}$ as the probability of moving from state q_i to state q_j with input σ, thus for a word $w = \sigma_1, \ldots, \sigma_n$, we have $\mu_w = \mu_{\sigma_1} \cdots \mu_{\sigma_n}$ and $[\mu_w]_{i,j}$ is the probability of moving from state q_i to state q_j with input string w. Let γ will be vector of 1 in accepting states and 0 otherwise. Assuming that initial state is s_1, then $[\mu_w]_1 \cdot \gamma$ is the probability that w is accepted by the probabilistic automaton.

We now introduce a theorem which relates the size of minimal automaton representing f to the rank of F, its associate matrix. Recall the rank of a matrix is the number of linearly independent rows (or, equivalently, columns) in the matrix.

Theorem 11.6. *([22, 37]) Let $f : \Sigma^* \to K$ such that $f \not\equiv 0$ and let F be the corresponding Hankel matrix. Then, the size r of the smallest automaton A such that $f_A \equiv f$ satisfies $r = rank(F)$ (over the field K)*

11.3 Multiplicity Automata and Partially Observable MDP

In this section, we connect POMDPs and multiplicity automata. We show that POMDPs are actually MA. The connection is done through using the Predictive State Representation (PSR), which has received much attention in recent years (See for example [68, 96, 95]).

The PSR of the POMDP describes the POMDP based on the possible futures rather than based on its structure. A future in a POMDP is a set of action observation pairs. A *test* is defined as a sequence of actions and observations $a_1, o_1, \ldots, a_k, o_k$, for every history[2] h the probability of test being successful is defined as $\Pr(t|h) = \Pr(o_k, \ldots o_1|a_k, \ldots a_1, h)$. It is clear that if we enumerated all possible tests for all possible initial histories, then the model would be fully specified. However, the size of the representation described so far is infinite. [68] showed that this representation can be done by a square matrix at size at most $|S|$. We will reprove this claim using the inherent connection between the PSR and the multiplicity automata.

Theorem 11.7. *A POMDP is a special case of multiplicity automata.*

Proof. We will show how to construct a multiplicity automaton as was defined in definition 11.5 from a POMDP. The size of the automaton will be $|S|$, and Σ is $A \times O \times R$. We let γ be the initial distribution over the states, and for every $a \in A$ and $o \in O \times R$ we define a matrix $\mu_{a,o}$, where $\mu_{a,o}(i,j) = P(s_j, o|s_i, a)$. Recall that $\mu_\lambda = I$ and $\mu_{a_1,o_1,\ldots a_k,o_k} = \mu_{a_1,o_1} \cdots \mu_{a_k,o_k}$. We prove by induction that for , $[\mu_{a_1,o_1,\ldots a_k,o_k}]_i \cdot \gamma = \Pr(o_k, \ldots, o_1|a_k \ldots, a_1, s_i)$, where $[M]_i$ is the ith row and $\gamma = (1, 1, 1 \ldots, 1)$. First we prove the induction base case,

$$[\mu_{a_1,o_1}]_i \cdot \gamma = \sum_{j=1}^{n} \Pr(s_j, o_1|a_1, s_i) = \Pr(o_1|a_1, s_i)$$

[2]We will allow for history of size 1 to be an initial states

Now assume the induction hypothesis holds for $k-1$ and prove for k

$$
\begin{aligned}
[\mu_{a_1,o_1,\ldots,a_k,o_k}]_i \cdot \gamma &= [\mu_{a_1,o_1}\mu_{a_2,o_2,\ldots,a_k,o_k}]_i \cdot \gamma \\
&= \sum_{j=1}^{n} [\mu_{a_1,o_1}]_{i,j} [\mu_{a_2,o_2,\ldots,a_k,o_k}]_j \cdot \gamma \\
&= \sum_{j=1}^{n} [\mu_{a_1,o_1}]_{i,j} \Pr(o_k,\ldots,o_2|a_k,\ldots,a_2,s_j) \\
&= \sum_{j=1}^{n} \Pr(s_j,o_1|a_1,s_i) \Pr(o_k,\ldots,o_2|a_k,\ldots,a_2,s_j) \\
&= \sum_{j=1}^{n} \Pr(o_k,\ldots,o_1,s_j|a_k,\ldots,a_1,s_i) \\
&= \Pr(o_k,\ldots,o_1|a_k,\ldots,a_1,s_i)
\end{aligned}
$$

□

Corollary 11.8. *POMDP can be represented as multiplicity automata of size r, $r \leq |S|$.*

Proof. By Theorem 11.7, we know that a POMDP is multiplicity automata by Theorem 11.6 its size is equal to the Hankel matrix rank. The rank of the Hankel matrix is bounded by $|S|$ since we can first write $|S|$ rows corresponding to all states, and these rows will span any other possible initial history. □

An important note here is that the Hankel matrix corresponding to the multiplicity automata is essentially the same matrix used in the PSR literature. Thus the inherent connection between the approaches. In the next section, similarly to the PSR works we show how to construct a minimal submatrix of the Hankel matrix with the Hankel matrix rank. We denote the minimal basis as the set of rows and columns which consists the minimal submatrix of the Hankel matrix.

Input : POMDP=(S, A, O, R, P)
Output: Basis $B = \{b_1, \ldots, b_r\}$ and test basis $T = \{t_1, \ldots, t_r\}$
Initialize $B \leftarrow \{s_1\}$, $T \leftarrow \{\lambda\}$;
while *Exists* $\sigma \in A \times O \times R$ *and* $b \in S \setminus B$, *such that*
1. $F_b^T = \sum_{b_i \in B} \alpha_i F_{b_i}^T$
2. $\exists y \in T : P(\sigma \circ y|b) \neq \sum_{b_i \in B} \alpha_i P(\sigma \circ y|b_i)$
do
  $B = B \cup \{b\}$;
  $T = T \cup \{\sigma \circ y\}$;
end

Algorithm 16: Construction a minimal matrix representing POMDP

11.4 A Minimal Basis

While in the previous section we pointed out that there exists a minimal basis in size at most $|S|$ of both histories and tests, in this section we show how it can be found efficiently. We also show how to find efficiently a basis with additional properties — namely, a basis set that only needs "small" weights to span all states. This "small" weight property allows us to construct our planning algorithm using such a basis.

11.4.1 POMDP to Minimal Matrix Representation

In this subsection, we construct a minimal size matrix corresponding to a given POMDP.

Starting with notation. Given a set of tests $T = \{t_1, \ldots, t_m\}$ we define for every initial belief state x, a vector F_x^T of size m, where $F_x^T(i) = \Pr(t_i|x)$, which denotes the probability of test t_i starting from belief state x. We also define $\Pr(\lambda|x) = 0$ for any belief state x (recall λ is the empty string). For a state $s \in S$, we slightly abuse notation by writing s to also mean the belief state which places all probability mass on the state s. So we write F_s^T, where the initial belief state has all probability mass on state s. For a set of belief states X and a set of Tests T, we denote F_X^T as the set $\{F_x^T|x \in X\}$. Also note that the POMDP is represented by infinite matrix, but the rank of the matrix is finite and the algorithm finds a matrix with the minimal rank.

Before proving that the algorithm constructs the minimal basis, let us provide intuition behind the algorithm. First, this algorithm resembles the one for learning MA from equivalence and member queries given in [10] and the algorithm of [68] where the predictive state representation was introduced. The algorithm begins by taking the test basis to be the empty test and for the initial history to be an arbitrary state. After phase k the algorithm has k histories and k tests which are linearly independent [3] and should decide whether they are a basis and if not it should find an additional test and history and add them to the basis. We show that if we did not find a basis yet then an additional history can be found in S and an additional test can be found as a one-step, extension test of the current basis tests. Thus, a basis can be computed efficiently.

Lemma 11.9. *Algorithm 16 builds a submatrix of the Hankel matrix with rank equal to the Hankel matrix rank.*

Proof. We first show that at any stage B is linearly independent with respect to the tests set T. We prove by induction, the basis holds trivially. Assume that the induction hypothesis holds for $B_\ell = \{b_1, ..., b_\ell\}$ and $T_\ell = \{t_1, ..., t_\ell\}$ and prove for $\ell+1$. Since $F_{B_\ell}^{T_\ell}$ are linearly independent then $F_{B_\ell}^{T_{\ell+1}}$ are linearly independent as well. By the way we choose $t_{\ell+1}$ and $b_{\ell+1}$ we are assured $F_{b_{\ell+1}}^{T_\ell+1}$ is linearly independent of $F_{B_\ell}^{T_{\ell+1}}$. Now we have to prove that our matrix is not smaller than the minimal matrix. By the construction of [10] we know that that if the matrix rank is less than the Hankel matrix rank, then there exists a counterexample w and a test $\sigma \circ y$, $y \in T$ such that

- $F_w^T = \sum_{b_i \in B} \alpha_i F_{b_i}^T$
- $P(\sigma \circ y|w) \neq \sum_{b_i \in B} \alpha_i P(\sigma \circ y|b_i)$

Since every w is a linear combination of $S = \{s_1, \ldots, s_n\}$, then F_w^T can be represented as linear combination F_S^T. Thus the rank of $s_1, \ldots, s_n$ is larger than the rank of B with respect to $T \cup \{\sigma \circ y\}$, and we can find a state s_i which is linearly independent of B with respect to T thus adding s_i to B increases its dimension. $\square$

Lemma 11.10. *Algorithm 16 terminates in*
$O(r^4|S|^2|A||O||R|)$ *steps.*

[3] This implies that the αs calculated by the algorithm are unique

Proof. Since we build a basis of size at most $|S|$, there at most $|S|$ iterations. In each iteration we check for every combination of state, action and observation if they satisfy condition 1 and 2 which takes $O(r^3)$ □

11.4.2 A Modified Belief MDP

Similarly to the *belief MDP*,we would like to define the *modified belief MDP*.

Definition 11.11. Let r be the size of a basis $B = (b_1, \ldots, b_r)$ and let $T = \{t_1, \ldots, t_r\}$ be the test basis. The modified belief MDP states are vector of size r such that $x = (\alpha_1, \ldots, \alpha_r)$ if and only if $F_x^T = \sum_{i=1}^r \alpha_i F_{b_i}^T$. Let $x = (\alpha_1, \ldots, \alpha_r)$ and $y = (\beta_1, \ldots, \beta_r)$ two modified belief states, then $\Pr(x, a, y) = \sum_{b' | F_{b'}^T = \sum_{i=1}^r \beta_i F_{b_i}^T} \Pr(b, a, b')$ for any $b = \sum_{i=1}^r \alpha_i F_{b_i}^T$. Let $x = (\alpha_1, \ldots, \alpha_r)$ and b a any belief state such that $F_b^T = \sum_{i=1}^r \alpha_i F_{b_i}^T$, then the reward of action u in state x is $\sum_{r' \in R} \Pr(r|u, b) r'$.

The optimal policy in the *modified belief MDP* as the optimal policy in the *belief MDP* is the optimal policy in the original POMDP, since in both cases the representation is exact.

Theorem 11.12. *The matrix representation is sufficient statistics*

Proof. First we note that $\Pr(o|s, a)$ can be found in the matrix and now all that remain to decode is the transition probability. Also note that if we find a state (or combination of states),s' such that for every test t $\Pr(t|s) = \Pr(t|s')$ then they are equivalent. Furthermore, if they are equivalent on every basis tests then they are equivalent. We consider $B = (s_1, \ldots, b_n)$ and the test basis $T = \{t_1, \ldots, t_r\}$. For every action a we look on the set $F = \{\Pr(t_j \circ a|s_i) | t_j \in T\}$ and find coefficients α_i such that $\alpha_i \in [0, 1]$, $\sum_{i=1}^n \alpha_i = 1$ and $F = \sum_{i=1}^n \alpha_i F_{s_i}^T$. Now $P(s, s_i, a) = \alpha_i$ and we are done. □

Note that although each state in the *modified belief MDP* is represented by a coefficients vector, this representation differs from the standard belief space representation as the coefficients of the basis states $F_s^T = \sum_{s_i \in B} \beta_i F_{s_i}^T$ can be very large and negative.

Therefore, building a grid with respect to the *modified belief MDP* will depend on the largest coefficient and on the matrix rank (which is at most $|S|$). In the next section we would show how to improve our basis and overcome the dependence on the largest coefficient.

11.4.3 Barycenteric Spanner - Improving the basis

In this subsection we introduce how to transform the basis outputted by Alg. 16 to a "better" basis by using [8] technique. More formally, the main theorem of the subsection shows that we can find a basis such that for any *belief state* largest coefficient is bounded by 2.

Theorem 11.13. *Let r be the rank of the multiplicity automata representing the POMDP and let T be the set of tests outputted by Alg. 16. There exists a set of states $B \subset S$, where $|S| = r$, such that, for any belief state b, the vector F_b^T can be uniquely represented as $F_b^T = \sum_{\tilde{b} \in B} \alpha_i F_{\tilde{b}}^T$, with $\max_i |\alpha_i| \leq 2$.*

We first provide a definition of barycenteric spanner.

Definition 11.14 ([8]). A C-barycenteric spanner is a basis B, which spans a convex set S, such that every $x \in S$ can be represented as $x = \sum_{b_i \in B} \alpha_i b_i$, where $|\alpha_i| \leq C$.

[8] showed that if S is a compact set that contained in $\mathbb{R}^d$ but is not contained in linear subspace of smaller dimension then there exists a 1- barycenteric spanner. If we could transform that basis outputted by Alg. 16 to a barycenteric spanner, we can the discretize the *modified belief MDP* with dependence only on the automaton rank since the largest coefficient is constant. The next algorithm shows how to construct efficiently an "almost" spanner as was done in [8]. We denote the matrix of basis initial histories B and Tests T as $M(B,T)$, where $M(B,T)_{i,j} = \Pr(t_j | b_i)$.

Note the spanner spans the the set $\{F_s^T : s \in S\}$, which is identical to span the set $\{F_b^T : b \text{ is a belief state}\}$.

This next lemma proves that the Algorithm builds a 2-Barycenteric spanner and its proof can found in [8]. For intuition, note that this algorithm must terminate since

Input : POMDP=(S, A, O, R, P)
Basis $B = \{b_1, \ldots, b_r\}$
and test basis $T = \{t_1, \ldots, t_r\}$
Output: Basis $\tilde{B} = \{\tilde{b}_1, \ldots, \tilde{b}_r\}$
Initialize: $\tilde{b}_i = b_i$, $\forall i$;
while $\exists x \in S \setminus \tilde{B}, i \in \{1, \ldots, r\}$ *satisfying* $det(M(x \cup B \setminus \tilde{b}_i, T)) > 2det(M(\tilde{B}, T))$ **do**
  $\tilde{b}_i = x$;
end

Algorithm 17: Construction a 2-Barycenteric spanner

there exists a 1-barycenteric spanner, which achieves the largest determinant, and in each iteration the determinant grows by at least a factor of two.

Lemma 11.15. *Alg. 17 constructs a 2-Barycenteric spanner for the set* $\{F_s^T : s \in S\}$.

Next we show that Algorithm 17 terminates in polynomial time.

Lemma 11.16. *Algorithm 17 terminates in* $O(|S|r^4 \log r)$ *steps.*

Proof. By Lemma 2.5 in [8] we know that the loop repeats $O(r \log r)$. In each iteration we calculate the determinant of a square matrix of size r at most S times, thus each iteration takes at most $|S|r^3$ time steps. □

Now we are ready to prove the main theorem of this subsection, which demonstrates the advantage of the basis produced by Alg. 17 over the basis produced by Alg. 16 in which α_i is unbounded.

Proof of Theorem 11.13:

Let $b = \sum_{i=1}^{|S|} \beta_i s_i$ any belief state.

$$\begin{aligned} F_b^T &= \sum_{i=1}^{|S|} \beta_i F_{s_i}^T = \sum_{i=1}^{|S|} \beta_i \sum_{j=1}^{r} \gamma_j(i) F_{\tilde{b}_j}^T \\ &= \sum_{j=1}^{r} \sum_{i=1}^{|S|} \beta_i \gamma_j(i) F_{\tilde{b}_j}^T = \sum_{j=1}^{r} \alpha_j F_{\tilde{b}_j}^T \end{aligned}$$

Now we observer that

$$\alpha_j = \sum_{j=1}^{r} \sum_{i=1}^{|S|} \beta_i \gamma_j(i) \le \max_i \gamma_j(i) \le 2,$$

where the first inequality comes from the fact that $\sum_{i=1}^{|S|} \beta_i = 1$ and that $\beta_i > 0$ for every i, and the second inequality follows from the fact that s_is were used in building the barycenteric spanner, and thus for every i, j $|\gamma_j(i)| \le 2$. □

11.5 Planning Algorithm

We are finally ready to describe our planning algorithm, Algorithm 18. The planning algorithm uses the first two algorithms to construct a basis of size r of initial states and r tests. With each *modified belief state* fully represented by a linear combination of the basis states with coefficient bounded in $[-2, 2]$, note that in contrast to the standard belief state the coefficient can be negative. Instead of discretizing the *Belief MDP* we discretize the *modified belief MDP*, which is built with respect to the basis (see definition 11.11). Our planning algorithm will then discretize it will compute the optimal policy for this MDP.

11.5.1 The Algorithm

Input : POMDP $P = (S, A, O, R, P)$

Output: An ε-optimal policy π

Run Alg. 16 on (P) and obtain $T = \{t_1, \dots t_r\}$ and $\tilde{B} = \{b_1, \dots, b_r\}$;

Run Alg. 17 on $(P, \tilde{B}, T)$ and obtain $B = \{b_1, \dots, b_r\}$;

Build a grid based MDP, $\tilde{M}$ with respect to the basis $B = \{b_1, \dots, b_r\}$ and tests $T = \{t_1, \dots, t_r\}$;

Each state in the MDP is a vector of size r, $(\alpha_1, \dots, \alpha_r)$, where $\alpha_i = m\tilde{\varepsilon}$, $m \in \{-2/\tilde{\varepsilon}, \dots, 0, \dots, 2/\tilde{\varepsilon}\}$, $\tilde{\varepsilon} = \varepsilon/r$;

Compute an optimal policy π for $\tilde{M}$.

Algorithm 18: Planning

Now we state our main theorem.

Theorem 11.17. *Given a POMDP, Algorithm 18 computes an $\frac{\varepsilon}{(1-\gamma)^4}$-optimal policy in time polynomial in $|S|, |A|, |O|, |R|, \varepsilon$ and exponentially only in r , where r is the size of the multiplicity automaton representing the POMDP.*

11.5.2 Algorithm Analysis

We first show that the algorithm is efficient, i.e., polynomial in $|S|, |A|, |O|$ and exponential in r the matrix rank. Later we provide performance guarantees for the the algorithm, i.e., that it produces an ε-optimal policy.

Lemma 11.18. *Algorithm 18 terminates in $O\left(r^4|S|^2|A||O||R|\log r + (\frac{2r}{\varepsilon})^{3r}\right)$ steps.*

Proof. Let's first analyze the time. By Lemmas 11.10 and 11.16 running Alg. 16 and Alg 17 takes $O(r^4|S|^2|A||O||R|\log r)$. The grid MDP contains $(\frac{2r}{\varepsilon})^r$ states and thus we can solve it in $(\frac{2r}{\varepsilon})^{3r}$. □

Similarly to Subsection 11.1 we would like to show that if two states are mapped to the same grid state then they are close to each other in the sense that the optimal value is close between them.

Lemma 11.19. *Let b and b' two belief states that are mapped to the same state in the grid MDP, then for every action $u \in A$ $|Q^*(b,u) - Q^*(b',u)| \leq \frac{\varepsilon}{1-\gamma}$*

Proof. We show here that if b and b' are mapped to the same state in the ε-discretized modified belief MDP then for every policy π[4], $|V^\pi(b) - V^\pi(b')| \leq \varepsilon$. Let b be a belief state such that $F_b^T = \sum_{i=1}^r \alpha_i F_{b_i}^T$ and b' such that $F_{b'}^T = \sum_{i=1}^r \beta_i F_{b_i}^T$, then since b and b' are in the same state in the discretized MDP $\|\alpha - \beta\|_1 \leq \varepsilon$. Let R^t be the return

[4]Note that π is not necessarily stationary

in test t, then we can rewrite

$$\begin{aligned}
V^\pi(b) &= \sum_t \Pr(t|b,\pi)R^t = \sum_t \sum_{i=1}^r \alpha_i \Pr(t|b_i,\pi)R^t \\
&\le \sum_t \sum_{i=1}^r \beta_i \Pr(t|b_i,\pi)R^t + \frac{\varepsilon}{1-\gamma} \\
&= V^\pi(b') + \frac{\varepsilon}{1-\gamma}
\end{aligned}$$

Similarly we can show that $V^\pi(b) \ge V^\pi(b') - \frac{\varepsilon}{1-\gamma}$. □

Now similarly to the proof of Theorem 11.1 we apply Lemmas 11.3 and 5.3 to derive the following lemma.

Lemma 11.20. *Algorithm 18 computes an $\frac{\varepsilon}{(1-\gamma)^4}$-optimal policy.*

Now the proof of Theorem 11.17 follows from Lemmas 11.18 and 11.20.

Chapter 12

Convergence Time To Nash Equilibrium

In this Chapter we consider the on-line balancing from perspective of selfish users. This Chapter is based on [30, 36].

In this Chapter we are concerned with the time it takes for the system to converge to a Nash equilibrium, rather than the quality of the resulting allocation. The question of convergence to a Nash equilibrium has received significant attention in the Game Theory literature (see [40]). Our approach is different from most of that line of research in a few crucial aspects. First, we are interested in quantitative bounds, rather than showing a convergence in the limit. Second, we consider games with many players (users) and actions (machines) and study their asymptotic behavior.

Third, we limit ourselves in this work to a subclass of games that arise from load balancing, for which there always exists a pure Nash equilibrium, and thus we can allow ourselves to study only deterministic policies.

Our Model. This paper deals with load balancing (see, [9]). Jobs (players) are allowed to select a machine to minimize their own cost. The cost that a job observes from the use of a machine is determined by the load on that machine. We consider weighted load functions, where each job has a corresponding weight and the load on a machine is sum of the weights of the jobs running on it. The Chapter is divided to two, the centralized control case and to the distributed settings. We start with the centralized control.

12.1 Centralized Control

In this section, similarly to the Elementary Stepwise System (ESWS) (see [84]), at every time step only one job is allowed to move, and a centralized controller decides which job would move in the current time step. In our notation a strategy is the algorithm used by the centralized controller for selecting which of the competing jobs would move. Due to the selfish nature of jobs, we assume that when a job migrates its observed load is strictly reduced, which we refer to as an *improvement* policy. We also consider the well known case of *best reply* policy, where each job moves to a machine in which its observed load is minimal.

12.1.1 Potential Games

Monderer and Shapley [82] defined few classes of potential games. Common to the various classes of potential games is a potential function P that maps a game state (system state in our notation) into the Reals. We give here the definition of the different classes of potential games using our notations. Let s_t be the system state at time t and P be the potential function.

Exact potential game when a job migrates from M_i to M_j at time t then the following holds: $T_j(t+1) - T_i(t) = P(s_{t+1}) - P(s_t)$.

Weighted potential game when job J migrates from M_i to M_j at time t then the following holds: $T_j(t+1) - T_i(t) = c(J)(P(s_{t+1}) - P(s_t))$, for some constant $c(J)$ which depends only on the job J.

Ordinal potential game when a job migrates from M_i to M_j at time t then the following holds: $T_j(t+1) - T_i(t) < 0 \iff P(s_{t+1}) - P(s_t) < 0$.

Generalized ordinal potential game when a job migrates from M_i to M_j at time t then the following holds: $T_j(t+1) - T_i(t) < 0 \Longrightarrow P(s_{t+1}) - P(s_t) < 0$.

The difference between a generalized ordinal potential game and an ordinal potential game is in the case that the normalized load of the migrating job does not change, i.e., $T_j(t+1) = T_i(t)$. An ordinal potential requires that in such a case we have $P(s_{t+1}) = P(s_t)$, while for a generalized ordinal potential this is not a requirement, and any relationship can hold between $P(s_{t+1})$ and $P(s_t)$. For all classes of potential games, it is easy to observe that any improvement policy would converge to a pure equilibrium, since the potential function is bounded.

12.1.2 Unrelated Machines

In this section we consider the unrelated machines case with the restricted assignment. We show that this case is a generalized ordinal potential game.

To show the convergence we define a sorted lexicographic order of the vectors describing the machine loads as follows. Consider the sorted vector of the machine loads. One vector is called "larger" than another if its first (after the common beginning of the two vectors) load component is larger than the corresponding load component of the second vector. Formally, given two load vectors ℓ_1 and ℓ_2, let $s_1 = sort(\ell_1)$ and $s_2 = sort(\ell_2)$ where $sort()$ returns a vector in the sorted order. We define $\ell_1 \succ \ell_2$ if $s_1 \succ s_2$ using a lexicographic ordering, i.e., $s_1[i] = s_2[i]$ for $i < k$ and $s_1[k] > s_2[k]$.

We demonstrate that the sorted lexicographic order of the load vector always decreases when a job migrates. We first observe that only two machine are influenced by the migration of the job J at time t, $M_i = M(J, t)$, where job J was before the migration and $M_j = M(J, t+1)$, the machine J migrated to. Furthermore $L_i(t) > L_j(t+1)$, otherwise job J would not have migrated. Also note that $L_i(t) > L_i(t+1)$ since job J has left M_i. Let $L = \max\{L_i(t+1), L_j(t+1)\}$. Since $L < L_i(t)$ one can show that the new machine loads vector is smaller in the sorted lexicographic order than the old machine loads vector. This is summarized in the following claim.

Claim 12.1. *The sorted lexicographic order of the machine loads vector decreases when a job migrates.*

We can use the lexicographic order to define a potential function (since it defines a complete order between system states). Therefore we establish the following.

Corollary 12.2. *Load balancing of jobs with unrelated machines is a generalized ordinal potential game.*

Note that in the general model, a job that migrates without changing its load might change the system state and therefore the potential function is not an ordinal potential function.

General Weights. In the general case, the number of different system configurations is at most m^n. Using the above argument, any improvement policy converges to a Nash equilibrium, and gives us an upper bound on the convergence time equal to the number different sorted machine loads vectors (which is trivially bounded by the number of different system configurations).

Corollary 12.3. *For any ESWS strategy with an improvement policy, the system of multiple unrelated machines with restricted assignment reaches a Nash equilibrium in at most m^n steps.*

Discrete Weights. For the discrete weight setting, the number of different weights is K. Let n_i be the number of jobs with weight w_i. The number of different configurations of jobs with weight w_i is bounded by $\binom{m+n_i}{m}$. Multiplying the number of configurations for the different weights bounds the number of different system configurations. Since, by definition, $\sum_{i=1}^{K} n_i = n$, we can derive the following.

Corollary 12.4. *For any ESWS strategy with an improvement policy, the system of multiple unrelated machines with restricted assignment under the discrete weight setting reaches a Nash equilibrium in at most*

$$\prod_{i=1}^{K} \binom{m+n_i}{m} \leq (c\frac{n}{Km} + c)^{Km},$$

steps for some constant $c > 0$.

Integer Weights. To bound the convergence time for the integer weight setting, we introduce a generalized ordinal potential function and demonstrate that it decreases when a job migrates. We define the potential of the system at time t, as $P^{un}(t) = \sum_{i=1}^{m} 4^{L_i(t)}$. After job J migrates from M_i to M_j then we have that $L_i(t) - 1 \geq L_j(t+1)$, since J migrated. Also, since we have integer weights, $L_i(t+1) \leq L_i(t) - 1$. Therefore, the reduction in the potential is at least,

$$P^{un}(t) - P^{un}(t+1) = 4^{L_i(t)} + 4^{L_j(t)} - [4^{L_i(t+1)} + 4^{L_j(t+1)}] \geq 4^{L_i(t)}/2 \geq 2. \qquad (12.1)$$

This establishes that P^{un} is a generalized ordinal potential function.

Theorem 12.5. *For the system of multiple unrelated machines under the integer weight setting, the function $P^{un}(t) = \sum_{i=1}^{m} 4^{L_i(t)}$ is a generalized ordinal potential function.*

Since in the initial configuration we have that $P^{un}(0) \leq 4^W$ and when we terminate we have $P^{un}(T) \geq m$, we derive the following theorem.

Theorem 12.6. *For any ESWS strategy with an improvement policy, the system of multiple machines under the integer weight setting reaches a Nash equilibrium in $4^W/2$ steps.*

Next we show that this bound can be reduced to $O(mW + m4^{W/m+w_{max}})$ when using the Max Load Machine strategy.

Theorem 12.7. *For Max Load Machine strategy with an improvement policy, the system of multiple machines under the integer weight setting reaches a Nash equilibrium in at most $4mW + m4^{W/m+w_{max}}/2$ steps.*

Proof. We divide the schedule into two phases with respect to the maximum load among the machines. The first phase continues until $L_{max}(t) \leq W/m + w_{max}$, and then the second phase starts. Note that in a second phase all jobs on the most loaded machine might want to stay on it. At the start of the second phase, at time T, the potential is at most $m4^{L_{max}(T)} \leq m4^{W/m+w_{max}}$. By (12.1), at every step the potential drops by at least two, therefore the duration of the second phase is bounded by $m4^{W/m+w_{max}}/2$. Thus, it remains to bound the duration of the first phase, namely T. At any time $t < T$ we have $L_{max}(t) > W/m + w_{max}$. Since $L_{min}(t) \leq W/m$, every job in the maximal loaded machine can benefit by migrating to the least loaded machine. The Max Load Machine strategy will choose one of those jobs. By (12.1), the decrease in the potential is at least $4^{L_{max}(t)}/2 \geq P^{un}(t)/2m$. Therefore, after T steps we have $P^{un}(T) \leq P^{un}(0)(1 - 1/2m)^T$. Since $P^{un}(0) \leq 4^W$ and $P^{un}(T) \geq 1$, it follows that $T \leq 4mW$, which establishes the theorem. □

12.1.3 Related Machines

In this section we consider the related machines. We show for the case of related machines that there is a weighted potential function, where the factor of job J is simply its weight $w(J)$.

We first consider restricted assignments and assume that all jobs follow an *improvement* policy. We first define the potential of the system as follows:

$$P^{rl}(t) = \sum_{i=1}^{m} \frac{(L_i(t))^2}{S_i} + \sum_{j=1}^{n} \frac{w_j^2}{S_{M(j,t)}} = \sum_{i=1}^{m} S_i(T_i(t))^2 + \sum_{j=1}^{n} \frac{w_j^2}{S_{M(j,t)}}$$

The following lemma shows that the game is a *weighted potential game*. Since the factor of job J, $c(J)$, is identical to its weight $w(J)$, if all the jobs have unit weight then the game is an exact potential game (with potential function $\sum_{i=1}^{m} \frac{(L_i(t))^2}{2S_i} + \sum_{j=1}^{n} \frac{w_j^2}{2S_{M(j,t)}}$).

Lemma 12.8. *When a job of size w migrates from machine i to machine j at time t then $P^{rl}(t+1) - P^{rl}(t) = 2w(T_j(t+1) - T_i(t)) < 0$.*

Proof. Let $\Delta(P^{rl}) = P^{rl}(t+1) - P^{rl}(t)$.

$$\begin{aligned}
\Delta(P^{rl}) &= \frac{(L_j(t+1))^2}{S_j} + \frac{(L_i(t+1))^2}{S_i} + \frac{w^2}{S_j} - \frac{(L_i(t))^2}{S_i} - \frac{(L_j(t))^2}{S_j} - \frac{w^2}{S_i} \\
&= \frac{(L_j(t)+w)^2}{S_j} + \frac{(L_i(t)-w)^2}{S_i} + \frac{w^2}{S_j} - \frac{(L_i(t))^2}{S_i} - \frac{(L_j(t))^2}{S_j} - \frac{w^2}{S_i} \\
&= \frac{(L_j(t))^2 + 2wL_j(t) + w^2}{S_j} + \frac{(L_i(t))^2 - 2wL_i(t) + w^2}{S_i} + \frac{w^2}{S_j} \\
&\quad - \frac{(L_i(t))^2}{S_i} - \frac{(L_j(t))^2}{S_j} - \frac{w^2}{S_i} \\
&= 2w\left(\frac{L_j(t)+w}{S_j} - \frac{L_i(t)}{S_i}\right) = 2w\left(\frac{L_j(t+1)}{S_j} - \frac{L_i(t)}{S_i}\right) \\
&= 2w\left(T_j(t+1) - T_i(t)\right)
\end{aligned}$$

□

We now like to bound the drop in the potential in each step. Clearly, if we are interested in ϵ-Nash equilibrium, then the drop is at least $2w\epsilon > \epsilon$. Considering a Nash equilibrium, for integer weights and speeds then the drop is at least $(S_{max})^{-2}$. Since the initial potential is bounded by W^2, we can derive the following Theorem.

Theorem 12.9. *For any ESWS strategy with an improvement policy, the system of multiple related machines with restricted assignment reaches an ϵ-Nash equilibrium in at most $O(W^2/\epsilon)$ steps, and reaches a Nash equilibrium, assuming both integer weights and speeds, in at most $O(W^2 S_{max}^2)$ steps.*

For unrestricted assignment, by forcing to move the job from the most loaded machine we can improve the bound as follows.

Theorem 12.10. *For Max Load Machine strategy with best reply policy, the system of multiple related machines with restricted assignment reaches an ϵ-Nash equilibrium in at most*

$$O\left(W\sqrt{mS_{max} + \frac{nw_{max}^2}{\epsilon}}\right)$$

steps.

Proof. Let $P^{rl}(t) = P_1^{rl}(t) + P_2^{rl}(t)$, where , $P_1^{rl}(t) = \sum_{i=1}^{m} \frac{(L_i(t))^2}{S_i}$ and $P_2^{rl}(t) = \sum_{j=1}^{n} \frac{w_j^2}{S_{M(j,t)}}$. Let $T = \frac{W}{\sum_{i=1}^{m} S_i}$. We can rewrite the potential function as follows,

$$\begin{aligned} P_1^{rl}(t) &= \sum_{i=1}^{m} \frac{(L_i(t))^2}{S_i} = \sum_{i=1}^{m} (T_i(t))^2 S_i \\ &= \sum_{i=1}^{m} (T_i(t) - T + T)^2 S_i \\ &= \sum_{i=1}^{m} S_i T^2 + \sum_{i=1}^{m} S_i (T_i(t) - T)^2 + 2T \sum_{i=1}^{m} S_i (T_i(t) - T) \\ &= \sum_{i=1}^{m} S_i T^2 + \sum_{i=1}^{m} S_i (T_i(t) - T)^2, \end{aligned}$$

where we used the fact that $\sum S_i T_i(t) = \sum L_i(t) = W = \sum S_i T$. The first term, $\sum_{i=1}^{n} S_i T^2$ is constant and therefore can be ignored. We can rewrite the potential as:

$$P^{rl}(t) = \sum_{i=1}^{m} S_i (T_i(t) - T)^2 + \sum_{j=1}^{n} \frac{w_j^2}{S_{M(j,t)}}$$

and redefine $P_1^{rl}(t) = \sum_{i=1}^{m} S_i (T_i(t) - T)^2$. By Lemma 12.8 when a job of size w migrates from machine i to machine j at time t then $\Delta(P^{rl}) = P^{rl}(t) - P^{rl}(t+1) = 2w(T_i(t) - T_j(t+1))$. We also define $\delta(t) = \max\{T_{max}(t) - T, T - T_{min}(t)\}$.

We divide the run of the algorithm to two phases. The first phase ends when either $\delta(t) < 2w_{max}$ or $P_1^{rl}(t) < 4mw_{max}^2$ and then the second phase starts. At time t in the first phase we have $P_1^{rl}(t) = \sum_i (T_i(t) - T)^2 S_i \leq m\delta(t)^2 S_{max}$, which implies that $\sqrt{\frac{P_1^{rl}(t)}{mS_{max}}} \leq \delta(t)$. Since $\Delta(P^{rl}) = 2w(T_i(t) - T_j(t+1)) \geq 2w(\delta(t) - w_{max}) \geq \delta(t)$ we can obtain the following recurrence

$$P_1^{rl}(t+1) \leq P_1^{rl}(t) - \sqrt{\frac{P_1^{rl}(t)}{mS_{max}}}.$$

Let $t_0 = 1$ and let t_i be the first time t at which $P_1^{rl}(t) \leq P_1^{rl}(t_0)/2^i$. Since $P_1^{rl}(t) \geq P_1^{rl}(t_{i-1})/2$ for $t \in [t_{i-1}, t_i]$, we have that, $t_i \leq t_{i-1} + \sqrt{mS_{max} P_1^{rl}(t_{i-1})/2}$. When

$P_1^{rl}(t) < 4mw_{max}^2$ we clearly finished the first phase. This implies that the duration of the first phase is bounded by,

$$t_\ell \leq \sum_{i=1}^{\ell} \sqrt{mS_{max}P_1^{rl}(t_0)/2^i} = O(\sqrt{mS_{max}P_1^{rl}(t_0)})$$

where $\ell = \log P_1^{rl}(t_0) \leq \log W^2 S_{max}$.

At the start of the second phase, at time τ, we bound the potential as follows. Clearly, $P_2^{rl}(\tau) \leq nw_{max}^2$. Since we ended the first phase either $P_1^{rl}(\tau)$ is bounded by $4mw_{max}^2$, or $\delta(\tau) < 2w_{max}$, which implies that $P_1^{rl}(\tau)$ is bounded by $4mw_{max}^2$. This implies that

$$P^{rl}(\tau) = O(mw_{max}^2 + nw_{max}^2).$$

Since we can assume that $n \geq m$, we obtain that

$$P^{rl}(\tau) = O(nw_{max}^2).$$

Since we are interested in ϵ-Nash, the minimal improvement is at least ϵ, and we derive a $P^{rl}(\tau)/\epsilon$ bound on the second phase. $\square$

Discrete Weights. We show that for any K integer weights there is an equivalent model in which w_{max} is bounded by $O(K(S_{max}n)^{4K})$, and therefore $W = O(Kn(S_{max}n)^{4K})$. This allows us to translate the results using W to the discrete weight model by replacing W by $O(Kn(S_{max}n)^{4K})$. (We do not need to calculate the equivalent weights, since they are only used for the convergence time analysis.) We first define what we mean by an equivalent set of weights.

Definition 12.11. Two discrete set of weights $w_1, \ldots, w_K$ and $\alpha_1, \ldots, \alpha_K$ are equivalent if for any two assignments, $n_1, \ldots, n_K$ and $\ell_1, \ldots, \ell_K$, where $\sum_{i=1}^K n_i \leq n$ and $\sum_{i=1}^K \ell_i \leq n$, we have $\sum_{i=1}^K n_i w_i > \sum_{i=1}^K \ell_i w_i$ if and only if $\sum_{i=1}^K n_i \alpha_i > \sum_{i=1}^K \ell_i \alpha_i$, and $\sum_{i=1}^K n_i w_i = \sum_{i=1}^K \ell_i w_i$ if and only if $\sum_{i=1}^K n_i \alpha_i = \sum_{i=1}^K \ell_i \alpha_i$.

Intuitively, the above definition implies that as long as we use only comparisons, we can replace $w_1, \ldots, w_K$ by $\alpha_1, \ldots, \alpha_K$. Most important for us is that we can use in the potential the α's rather than the w's. From the definition of an equivalent set of weights we can derive the following. Any strategy based on comparisons of job weights and machine loads and an improvement policy based on comparisons of machine loads (e.g., best reply) would produce the same sequence of job migrations starting from any initial configuration.

The following theorem, which is proven using linear integer programming techniques, bounds the size of the equivalent weights.

Theorem 12.12. *For any discrete set of integer weights $w_1, \ldots, w_K$ there exist an equivalent set of integer weights $\alpha_1, \ldots, \alpha_K$ such that $\alpha_K \leq K(cS_{max}n)^{4K}$ for some constant $c > 0$.*

Proof. Definition 12.11 defines equivalent weights. Let $\{w_1, \ldots, w_K\}$ be a set of K integer weights. An equivalent set of weights can also be viewed as follows: consider all possible assignments (of up to n) jobs from K different integer weights to a single machine. We can encode an assignment by $(n_1, \ldots, n_K)$, where $\sum_{i=1}^{K} n_i \leq n$, and the load on a machine is $L = \sum_{i=1}^{K} w_i n_i$.

The idea of equivalent weights is that rather of considering the exact load of each assignment, we are only interested in their relative load. Namely, we are only interested in comparing the load on two different machines. From definition 12.11 an equivalent set of weights is a set of weights, which keeps the relative order between every pair of assignments. Our aim is to write an linear integer program that will describe the constraints that for any two possible assignments the comparison between the loads is maintained.

The integer program for identical machines is defined as follows. Let $x_1, \ldots, x_K$ be the unknown (new) weights. For every two possible assignments $\vec{\alpha} = (\alpha_1, \ldots, \alpha_K)$ and $\vec{\beta} = (\beta_1, \ldots, \beta_K)$, such that $\sum_{i=1}^{K} \alpha_i \leq n$ and $\sum_{i=1}^{K} \beta_i \leq n$, we generate an inequality in the integer linear program. The inequality compares $\sum_{i=1}^{K} \alpha_i x_i$ and $\sum_{i=1}^{K} \beta_i x_i$. To decide the result of the comparison we compare $\sum_{i=1}^{K} \alpha_i w_i$ to $\sum_{i=1}^{K} \beta_i w_i$. If they are equal we add the equation $\sum_{i=1}^{K}(\alpha_i - \beta_i)x_i = 0$. If $\sum_{i=1}^{K} \alpha_i x_i > \sum_{i=1}^{K} \beta_i x_i$ we add $\sum_{i=1}^{K}(\alpha_i - \beta_i)x_i > 0$. Otherwise, we add $\sum_{i=1}^{K}(\alpha_i - \beta_i)x_i < 0$. In addition, we require that the weights are positive, namely, for every i we have an inequality $x_i > 0$.

For the related machines we need to take into account their speeds as well. (We assume that the speeds are integers.) Rather than generating an inequality for each pair of assignments, we generate an inequality for each pair of assignments and machines. For instance, if the assignments are $\vec{\alpha}$ and $\vec{\beta}$ and the machines are M_1 and M_2 we compare $\sum_{i=1}^{K} S_2\alpha_i w_i$ to $\sum_{i=1}^{K} S_1\beta_i w_i$. The generated inequality would depend, as before, on the output of the comparison.

Let A be the matrix that represent the inequalities. For identical machines note that the sum of the absolute entries in each row is bounded by $O(n)$, and for related machines by $O(nS_{max})$. The main observation that we use to bound the magnitude of the solution is based on the following lemma.

Lemma 12.13. *If B is a square submatrix of A, then $|det(B)| \leq (cS_{max}n)^k$, for some constant $c \geq 1$.*

The above lemma, with standard integer programming arguments (see [51]) allows us to derive a bound of $K(cnS_{max})^{4K}$, for some constant $c \geq 1$, on the maximum weight in the equivalent set of weights. □

Unit Weight Jobs. We show that for unit weight jobs, there exists a strategy that converges in mn steps. The unit weight jobs is a special case of [80] with a symmetric cost function, where was derived an upper bound of $O(mn^2)$ on the convergence time of a specific strategy. We follow the proof of [80] and obtain a better bound in our model.

Theorem 12.14. *There exists an ESWS strategy such that the system of multiple related machines with restricted assignment reaches a Nash equilibrium in at most mn steps in the case of unit weight jobs.*

Proof. We first give the definition of a push out and pull in paths as defined in [80].

Definition 12.15. A **Push out path** is a triplet (s, MM, JJ) where s is a system state, $MM = M_{i_0}, M_{i_1}, M_{i_2}, \ldots, M_{i_N}$ and $JJ = J_{i_1}, J_{i_2}, \ldots, J_{i_N}$ where the sequence of migration, starting at state s, and following it at step k having job J_{i_k} migrate to machine M_{i_k} from machine $M_{i_{k-1}}$ is a legal execution of an ESWS. Similarly, a **Pull in path** is a triplet (s, MM, JJ) where s is a system state, $MM =$

while $A(t) \neq \emptyset$ **do**

A Choose $J \in A(t)$. Let M be the machine with job J, U_1 be the set of machines with load smaller than that of M, and U_2 be the complement of U_1;

B Let J migrate to machine $M' \in U_1$ and continue with a push out path. (Namely, if some job J' now wants to migrate from M', add it to the push out path, and continue.);

C If there exists a job $I \in U_2$ that wants to migrate from M' to M, let it and follow a pull in path. (Namely, if some job I' now wants to migrate to M', add it to the pull in path, and continue.);

end

Algorithm 19: Push out - Pull in Algorithm

$M_{i_1}, M_{i_2}, \ldots, M_{i_N}, M_{i_{N+1}}$ and $JJ = J_{i_1}, J_{i_2}, \ldots, J_{i_N}$ where the sequence of migration, starting at state s, and following it at step k having job J_{i_k} migrate from machine M_{i_k} to machine $M_{i_{k+1}}$ is a legal execution of an ESWS.

In contrast the result in [80] we can bound the pull in path by m steps as well (the payoff is symmetric). We consider the following algorithm:

Lemma 12.16. *Each iteration of the loop (A - C) terminates after at most m steps.*

Proof. A is a single step. Since both in the pull in path and in the push out path no machine is repeated twice, we obtain the following, B is a push out path through U_1 thus it has at most $|U_1|$ steps. C is a pull in path in U_2 and thus it has at most $|U_2|$ steps. Since $|U_1| + |U_2| + 1 = m$ we conclude the Lemma. □

It still remains to show that in two times $t < \tilde{t}$ in which we are at start of an iteration, we have that $A(\tilde{t}) \subset A(t)$. We prove for any two consecutive times t, t' and $t' < t$ in which we were at start of an iteration we have that $A(t') \subset A(t)$. We note that after performing steps A - C the load is changed only in two machines, the last machines in the pull in and the push out paths. Since in the rest of machines the load remained unchanged, a job on them became unstable only if it wants to migrate to the last machine in the pull in path, but this is not possible.

□

The next theorem presents a lower bound of $\Omega(mn)$ on the convergence time of some ESWS strategy (different from that of Theorem 12.14).

Theorem 12.17. *There exists an ESWS strategy with an improvement policy such that for the system of multiple related machines with unrestricted assignment, there exists a system configuration that requires at least $\Omega(mn)$ steps to reach a Nash equilibrium in the case of unit weight jobs.*

Proof. Let $S_i = 1 + i/n$. The initial configuration has n unit weight jobs on M_1. Note that the only Nash equilibrium assigns n/m jobs on each machine. Now consider the strategy that each time takes a job from M_1 and moves it through all the machines to the that with the least number of jobs. The number of steps $\sum_{i=1}^{n/m}\sum_{j=1}^{m}(m-j) = \Omega(nm)$. □

12.1.4 Identical Machines

In this section we show improved upper bounds that apply to identical machines with unrestricted assignment. We also show a lower bound for K weights which is exponential in K. The lower bound is presented for the Min Weight Job policy. Clearly, this lower bound also implies a lower bound in all the other models. First we derive some general properties. The next observation states the minimal load cannot decrease.

Observation 12.18. For a system of multiple identical machines with unrestricted assignment, at every time step the minimal load among the machines either remains the same or increases.

Now we show that when a job moves to a new machine, this machine still remains a minimal marginal load machine for all jobs at that machine which have greater weight.

Observation 12.19. For a system of multiple identical machines with unrestricted assignment, if job J has migrated to its best response machine M_i at time t then M_i is a minimal marginal load machine with regard to any job $J' \in B_i(t)$ such that $w(J') \geq w(J)$.

Next we show that once a job has migrated to a new machine, it will not leave it unless a larger job arrives.

Claim 12.20. *For a system of multiple identical machines with unrestricted assignment, suppose that job J has migrated to its best response machine M at time t. If $J \in A(t')$ for $t' > t$ then another job J' such that $w(J') > w(J)$ migrated to machine M at time t'', and $t < t'' \leq t'$.*

Proof. Since M was the best response machine for J it implies that M is the minimal marginal load machine at time t with respect to J. By observation 12.18, the minimal load never decreases. Thus, the only reason that J wishes to migrate from machine M is that another job(s) migrated to machine M. By Observation 12.19 arrival of a smaller or equal weight job maintains M as the minimal marginal load machine with respect to J. Therefore, it must be the case that at least one job of weight greater than that of J migrated to M between $t + 1$ and t'. □

Next we present an upper bound on the convergence time of Max Weight Job strategy. (A similar claim (without proof) appears in [92].)

Theorem 12.21. *For the Max Weight Job strategy with best response policy, the system of multiple identical machines with unrestricted assignment reaches a Nash equilibrium in at most n steps.*

Proof. By Claim 12.20, once the job has migrated to a new machine, it will not leave it unless a larger job arrives. But under Max Weight Job strategy only smaller jobs can arrive in the subsequent time steps, so each job stabilizes after the first migration, and the theorem follows. □

Now we present a lower bound for the Min Weight Job strategy.

Theorem 12.22. *For Min Weight Job strategy with best response policy, there exists a system configuration that requires at least $(\frac{n}{K})^K/(2(K!)) \geq (n/K^2)^K$ steps to the system of multiple identical machines with unrestricted assignment to reach a Nash equilibrium, where $K = m - 1$.*

Proof. Consider the following scenario. There are $m-1$ classes of jobs $C_1, \ldots, C_{m-1}$ and each class contains $l = n/(m-1)$ jobs. The weights of the jobs are defined recursively: all jobs in class C_1 have weight 1 and all jobs in a higher class have weight larger than the total weight of all the jobs in the first $i-1$ classes, i.e. $w_k = l\sum_{i=1}^{k-1} w_i + 1$. (For example we can set $w_k = \ell^k(k-1)! + k$.) Initially, all jobs of class C_i $(i = 1, \ldots, m-1)$ are located at machine M_{i+1}. We show that for Min Weight Job strategy it takes at least $l^{m-1}/(2((m-1)!))$ steps to converge. Notice that the strategy, the best response policy and the initial configuration determine uniquely the whole schedule.

We start with a few useful notations. We denote by C_j^i all the classes $C_j, \ldots, C_i$. Similarly, M_j^i denotes the machines $M_j, \ldots, M_i$.

We divide the schedule into *phases*. A *k-phase* is defined as follows. Initially, all the jobs from classes C_1^k are located at a machine M' and there is a set S of k machines (not including M') participating in the phase that contain equal number of jobs from any class higher than C_k and this number is less than or equal to the number of jobs of the corresponding class located on any machine not in S. During the phase all but $l/(k+1)$ jobs from C_k are balanced between the machines in S. We will demonstrate that the duration of a phase grows exponentially with k. First we need the following observation.

Definition 12.23. For a set of machines S we define by $MIN_i(S)$ the subset of machines in S that contain the minimal number of jobs from C_i.

Observation 12.24. Suppose that at time t a job from C_k is selected by Min Weight Job strategy. Let C_r be a class in $\{C_1^{k-1}\}$ and let J be a job from C_r. It must be the case that $M(J,t) \in MIN_{r+1}(MIN_{r+2}(\ldots(MIN_{m-1}(\{M_1^m\}))))$.

The observation follows from the fact that no job from C_r wishes to change its machine and the weight of any job in a high class is greater than the total weight of all the jobs in the lower classes. Now we give a lower bound on the duration of a k-phase.

Lemma 12.25. *The duration of a k-phase is at least $l^k/(2(k!))$.*

Proof. The proof is by induction on the phase index.
Induction hypothesis. The duration of a k-phase is at least $l^k/(2(k!))$.
Basis $(k = 1)$**.** Let us consider 1-phase. Half of the jobs from C_1 migrate to another machine. Thus, the induction hypothesis trivially holds.
Step. Suppose that the induction hypothesis holds for all phases with index k' such that $k' < k$ and let us prove that it is also satisfied for a k-phase. After $k-1$ migrations of jobs from C_k, all jobs form the lower classes are concentrated at the same machine, since only one machine does not contain a job from class C_k, and by Observation 12.24 it implies that the best response is unique.

Thus, every k'th moving job from C_k would initiate a $(k-1)$-phase. Hence, the number of $(k-1)$-phases initiated by the jobs from C_k is l/k. By the induction hypothesis the duration of each $(k-1)$-phase is at least $l^{k-1}/(2((k-1)!))$. Therefore, the duration of a k-phase is at least

$$\frac{l}{k} \cdot \frac{l^{k-1}}{2((k-1)!)} = \frac{l^k}{2(k!)}.$$

□

It is easy to see that every $(m-1)$'th moving job from C_{m-1} generates a new $(m-2)$-phase. The duration of the convergence period follows by Lemma 12.25 after substituting $m-1$ instead of k. □

We also present a lower bound of $n^2/4$ on the convergence time of Min Weight Job and FIFO strategies for the case of two machines.

Theorem 12.26. *For the Min Weight Job and FIFO strategies with best response policy, there exists a system configuration that requires at least $n^2/4$ steps to the system of two identical machines with unrestricted assignment to reach a Nash equilibrium.*

Proof. Consider the following scenario. There are $n/2$ classes of jobs $C_1, \ldots, C_{n/2}$ and each class contains exactly 2 jobs and has weight $w_i = 3^{i-1}$. Notice that a job in C_i with weight $w_i = 3^{i-1}$ has weight equal to the total weight of all the jobs in the first $i-1$ classes plus 1.

Initially, all jobs are located at the same machine. We divide the schedule into *phases*. Let C_j^i denotes all jobs from classes $C_j, \ldots, C_i$. A *k-phase* is defined as follows. Initially, all jobs from classes C_1^k are located at one machine. During the phase these jobs, except one job from C_k, migrate to the other machine. Thus, the duration of a k-phase is $2k-1$. It is easy to see that the schedule consists of the phases $n/2, \ldots, 1$ for Min Weight Job strategy. One can observe that FIFO can generate the same schedule, if ties are broken using minimal weight. □

The following theorem shows a tight upped bound of $\Theta(n^2)$ on the convergence time of FIFO strategy.

Theorem 12.27. *For FIFO strategy with best response policy, the system of multiple identical machines with unrestricted assignment reaches a Nash equilibrium in at most $n(n+1)/2$ steps.*

Proof. We define a round to be a maximal sequence of jobs that migrate in which no job is repeated twice. Consider a round R and let J be the maximum weight job that wishes to migrate at the beginning of R. There exists a time t during round R in which job J either was selected by FIFO and migrated or become stable. According to Claim 12.20, J will not migrate in any of the consequent rounds. Therefore, the duration of k'th round is at most $n-k+1$. Thus, the total convergence time is bounded by $n(n+1)/2$. □

Similarly to FIFO, we bound the expected convergence time of Random strategy by $O(n^2)$.

Theorem 12.28. *For Random strategy with best response policy, the system of multiple identical machines with unrestricted assignment reaches a Nash equilibrium in expected time of at most $n(n+1)/2$ steps.*

Proof. We define round to be the sequence of jobs that migrate until the maximum weight job, that wants to migrate, migrates. Consider a round R and let J be the maximum weight job that wishes to migrate at the beginning of R. According to Claim 12.20, J will not migrate in any of the consequent rounds. Therefore, there are at most $n-k+1$ jobs that want to migrate in the k'th round and using Random strategy its expected time is at most $n-k+1$. Thus, the expected convergence time is bounded by $n(n+1)/2$. $\square$

Discrete Weights. For the discrete weight case, we demonstrate an upper bound of $O((n/K+1)^K)$ on the convergence time of any ESWS strategy, showing that the bound of Theorem 12.22 for the Min Weight Job is not far from the worst convergence time.

Theorem 12.29. *For any ESWS strategy with best response policy, the system of multiple identical machines with unrestricted assignment reaches a Nash equilibrium in $O((n/K+1)^K)$ steps.*

Proof. Suppose that we have K classes of jobs $C_1, \ldots, C_K$ with weights $w_1 < \ldots < w_K$. Notice that each job from K'th class moves at most once while the number of migrations of a job in a lower class is bounded by the number of job migrations in all the higher classes plus one.

Hence, the number of moves of a job is defined by the following recursion: $f(i) = \sum_{j=i+1}^{K}(f(j) \cdot |C_j|) + 1$, where $f(i)$ is the maximal number of moves of a job from class C_i. Notice that $f(K) = 1$. We argue that $f(i) = \prod_{j=i+1}^{K}(|C_j|+1)$ for $i < K$:

$$\begin{aligned} f(i) &= \sum_{j=i+1}^{K}(f(j) \cdot |C_j|) + 1 \\ &= f(i+1) \cdot |C_{i+1}| + \sum_{j=i+2}^{K}(f(j) \cdot |C_j|) + 1 \\ &= f(i+1) \cdot |C_{i+1}| + f(i+1) = f(i+1)(|C_{i+1}|+1), \end{aligned}$$

then we can continue recursively with $f(i+1)$ and so forth. Thus, the total number of job moves is bounded by $\sum_{i=1}^{K-1}\left(|C_i| \prod_{j=i+1}^{K}(|C_j|+1)\right) + |C_K|$. Using Lagrange multipliers we obtain that this expression is maximized when $|C_i| = n/K$ for all $1 \le i \le K$ since $\sum_{i=1}^{K}|C_i| = n$. This derives an upper bound of $\sum_{i=1}^{K-1}(\frac{n}{K}\prod_{j=i+1}^{K}(n/K+1)) + n/K = O((n/K+1)^{K-1}n/K) = O((n/K+1)^K)$. $\square$

Integer Weights. For the integer weight case, we show that the convergence time of any ESWS strategy is proportional to the sum of weights.

Theorem 12.30. *For any ESWS strategy with best response policy, the system of multiple identical machines with unrestricted assignment reaches a Nash equilibrium in $W + n$ steps.*

Proof. By Claim 12.20, once the job has switched to a new machine, it will not leave it unless a larger job arrives. Thus, all but one move of a job results from migrations of jobs with greater weight to its machine. Observe that when a job J moves to a machine M it can force other jobs with the total weight of at most $w(J) - 1$ to migrate from M. Otherwise, some job would leave a minimal marginal load machine. Then these jobs, in their turn, may cause migrations of other jobs on their destination machines. We claim that the total number of recursive migrations due to J is bounded by $w(J) - 1$. Let us define the push-out potential of the set of migrating jobs S, $PO(S) = \sum_{J \in S}(w(J) - 1)$. Initially, S consists of one job J and $PO(S) = w(J) - 1$. Then when a job $J' \in S$ migrates we remove it from S and add jobs on its target machine that would move due to J', but their total weight is less by at least least one than J' weight, thus PO decreases by at least one. Hence, the total number of migrations resulting from moves of all jobs is bounded by W and the theorem follows. ☐

Unit Weight Jobs. For the unit weight jobs, we present a lower bound on the convergence time of a specific strategy.

Theorem 12.31. *There exists an ESWS strategy with the improvement policy for which the worst case number of steps to the system of multiple identical machines with unrestricted assignment and unit weight jobs to reach a Nash equilibrium is at least $\Omega(\min\{mn, n \log n \frac{\log m}{\log \log n}\})$ steps.*

Proof. The initial configuration is as follows: n identical jobs on machine 1 none on the other machines. First we show for $m \leq \log(n)$ a strategy which requires at least $\frac{(m-1)n}{2}$ steps to reach Nash equilibrium. The strategy works as follows. First it moves $n/2$ jobs to machine M_2. Now recursively, considering machines M_2 to m we have $m-1$ machines and $n/2$ jobs (on machine 2). After balancing the $n/2$ jobs on machines M_2^m we reconsider machine 1. Now every machine has at least $n/(2(m-1))$ jobs. Therefore we can continue recursively with m machines and $n/2 - n/(2(m-1))$ jobs.

We let $f(n, m)$ be the number of steps in which the algorithm reaches Nash equilibrium. Then we have $f(n, m) = \frac{n}{2} + f(n/2, m-1) + f(\frac{n}{2} - \frac{n}{2(m-1)}, m)$. Next we prove by induction that that $f(n, m) \geq \frac{(m-1)n}{2}$. For the basis we have that $f(k, 1) = 0$ and since we have that $m \leq \ln(n)$ this is suffice for the basis. We assume that the induction holds for $k' \leq k$, $l' \leq l$ and prove for (l, k).

$$\begin{aligned} f(l,k) &= \frac{l}{2} + f(\frac{l}{2}, k-1) + f(\frac{l}{2} - \frac{l}{2(k-1)}, k) \\ &= \frac{l}{2} + \frac{\frac{l}{2}(k-2)}{2} + \frac{(\frac{l}{2} - \frac{l}{2(k-1)})(k-1)}{2} \\ &= \frac{l}{2} + \frac{lk}{4} - \frac{l}{2} + \frac{lk}{4} - \frac{l}{4} - \frac{l}{4} \\ &= \frac{lk}{2} - \frac{l}{2} = \frac{l(k-1)}{2} \end{aligned}$$

For the case where $m > \log(n)$. We consider the case where the strategy first balances the jobs on machines $M_1^{\log(n)}$, requiring $\Omega(n \log n)$ steps. Latter we consider $\log n$ independent problems, each has $m \log n$ machines and $n/\log n$ jobs, all starting on one machine. Each such level would require requiring $\Omega(n \log n)$ steps. There are $\log m/\log\log n$ such levels, therefore we have a lower bound of $\Omega(n \log(n) \frac{\log m}{\log \log n})$. □

12.2 Distributed Settings

In this section we consider the distributed settings. We consider algorithms that have no centralized unit and decisions are made locally and concurrently. The users (jobs) are anonymous and selfish. The decisions are no longer deterministic but are stochastic and a user action is a distribution over the machines. Therefore, if an

Input : Machines: M_1, M_2, Users: $X = \{x_1, \ldots, x_n\}$;
Let $t = 1$; $d_1 = |L_1(t) - L_2(t)|$;
`over` $= \arg\max\{L_1(t), L_2(t)\}$;
while $d_t \geq 2$ **do**
  for x_i $s.t$ $g_t(x_i) = M_{\texttt{over}}$ **do in parallel**
    Move with probability $d_t/(2L_t(M_{\texttt{over}}))$
  end
  $t = t + 1; d_t = |L_1(t) - L_2(t)|$;
  `over` $= \arg\max\{L_1(t), L_2(t)\}$;
end

Algorithm 20: Distributed algorithm, `BALANCE`, for two identical machines.

equilibrium point is not ex-post Nash[1], several users will try to migrate in the next time step.

Although the Elementary Step System has attractive theoretical properties, it has also very clear drawbacks. First, since only one user is allowed to rebalance in each stage, the convergence time is at least $\Omega(n)$. Second, the implementation of a centralized controller is unattractive and fairly difficult in a truly distributed system. Third, by avoiding the issue of concurrency, many realistic issues that are involved in the decision to rebalance "disappear", such as the fear of a user that after the rebalancing she would be worse off.

12.2.1 Two Machines

We start with the simple case of n users and 2 machines. We will show that the algorithm and insights of this simple case reflects later in the general case as well.

Two Identical Machines

In this subsection we analyze the convergence rate of a distributed algorithm for two identical machines.

We first introduce a probabilistic Lemma that is used through the paper and provides a bound on the probability of hitting the mean.

[1] An ex-post Nash remain an equilibrium even *after*, every user observe all the random choices of all other users.

Lemma 12.32. *Let $Z_1, \ldots, Z_n$ be i.i.d $\{0,1\}$-random variables with* $\Pr(Z_i = 1) = p$ *and* $Z = \sum_{i=1}^{n} Z_i$. *Then* $\Pr(Z = \lceil pn \rceil) \geq \frac{1}{c\sqrt{\lceil pn \rceil}}$ *for some constant* $c > 0$ [2]. *If* $pn = q$ *is an integer then,* $\Pr(Z = q) \geq \frac{1}{\sqrt{2\pi q}}$

We first give a technical lemma.

Lemma 12.33. *For algorithms that satisfy*

1. *if* $d_t \geq c\ln(1/\delta)$, *then* $d_{t+1} \leq \sqrt{cd_t \ln(1/\delta)}$ *with probability at least* $1-\delta$
2. *if* $d_t \leq c\ln(1/\delta)$, *then* $d_{t+1} \leq c\ln(1/\delta)$ *with probability at least* $1-\delta$.
3. *The system reach Nash equilibrium in one step with probability at least* $O(1/\sqrt{d_t})$. *at time* t.

Given that the difference at time t *is* d_t, *then by time* $t+\ell+1$ *algorithm terminates with probability at least* $(\frac{1}{2})^{\ell}\sqrt{1/(c')^{(\frac{1}{2})^{\ell}}}$.

Proof. By applying ℓ times the relative Chernoff bound (Lemma 2.4), with $\delta = 1/2$ each time, we obtain that the difference at time $t+\ell$ is bounded by $(cd_t)^{(\frac{1}{2})^{\ell}}$ with probability $(\frac{1}{2})^{\ell}$. Using Lemma 12.32, we obtain that the probability of terminating in the last stage is at least $O\left(\sqrt{1/(c')^{(\frac{1}{2})^{\ell}}}\right)$. □

Theorem 12.34. *With probability at least* $1-\delta$ *algorithm* `BALANCE`, *terminates within* $O(\ln\ln(n)) + \tilde{O}(\ln(1/\delta))$ *stages.*

Proof. Let ℓ be an upper bound on the number of stages of algorithm `BALANCE` until reaching Nash equilibrium. (We latter fix ℓ and show that it is bounded with high probability.) We divide the execution to two phases, the first is while $d_t > 3\ln(1/\delta')$ and the second starts when $d_t \leq 3\ln(1/\delta')$, where $\delta' = \delta/(2\ell)$. We first show that the first phase ends with probability $1-\delta/2$ within $O(\ln\ln(n))$ stages. By Lemma 2.4 with probability at least $1-\delta'$, $d_{t+1} \leq \sqrt{3d_t \ln(2/\delta')}$ for every stage $t \leq \ell$ in the first phase. Since d_0 is bounded by n, then $d_{t+1} \leq 3n^{1/2^{t+1}} \ln(1/\delta')$ with high probability. Therefore, after $O(\ln\ln(n))$ stages d_t is bounded by $3\ln(1/\delta')$ and once $d_t \leq 3\ln(1/\delta')$

[2] This holds only for $p \geq \frac{1}{c'n}$ but we do not use smaller probabilities.

it remains so with probability $1-\delta/2$ until the end of stage ℓ by Lemma 2.4. For the second phase, we apply Lemma 12.33 with $\ell = O(\ln\ln\ln(1/\delta'))$, and obtain that algorithm `BALANCE` terminates with probability $O(\frac{1}{\ln\ln(1/\delta')})$ in $O(\ln\ln\ln(1/\delta'))$ stages. Therefore, if we make additional $O(\ln\ln(1/\delta')\ln\ln\ln(1/\delta')\ln(2/\delta))$ stages, we will terminate with probability at least $1-\delta/2$. Solving the following equation

$$\ell = c_1 \ln\ln(n) + c_2 \ln(\ln(\ell/\delta)) \ln\ln(\ln(\ell/\delta)) \ln(2/\delta) + c_3,$$

we arrive at $\ell = O(\ln(1/\delta)\ln(\ln(1/\delta)\ln\ln(\ln(1/\delta)) + \ln\ln(n)) = O(\ln\ln(n)) + \tilde{O}(\ln(1/\delta))$.

□

A direct and simple corollary from the theorem is regarding the expected time of convergence.

Corollary 12.35. *Algorithm* `BALANCE` *terminates within expected* $O(\ln\ln(n))$ *stages.*

The following lemma shows that although algorithm `BALANCE` is not a Nash rebalancing policy, the deviation is bounded by 1.

Lemma 12.36. *At every stage* t *algorithm* `BALANCE` *is a* 1*-Nash rebalancing policy.*

Proof. Consider any stage before reaching Nash equilibrium, where there are $n+d$ users on machine 1 and $n-d$ on the machine 2, i.e., $d_t = 2d$. User on the overloaded machine rebalances with probability $\frac{d}{n+d}$. We compare the expected load on the machines, excluding one user on the overloaded machine.

$$\begin{aligned} L_1 &= (n+d-1)(1-\frac{d}{n+d}) = n-1+\frac{d}{n+d} \\ L_2 &= n-d+(n+d-1)(\frac{d}{n+d}) = n-\frac{d}{n+d}. \end{aligned}$$

Therefore the difference is bounded by 1 and can be as large as $1-\frac{2}{n+1}$. It is easy to see that the best response of users on the underloaded machine is to remain on their current machine.

□

Input : Machines: M_1, M_2, Users: $X = \{x_1, \ldots, x_n\}$;
Let $t = 1$; $d_1 = |L_1(1) - L_2(1)|$;
$\mathtt{over} = \arg\max\{L_1(t), L_2(t)\}$;
while $d_t \geq 2$ **do**
  for x_i *s.t* $g_t(x_i) = M_{\mathtt{over}}$ **do in parallel**
    Move with probability $(d_t - 1)/(2L(M_{\mathtt{over}}) - 2)$
  end
  $t = t + 1$; $d_t = |L_1(t) - L_2(t)|$; $\mathtt{over} = \arg\max\{L_1(t), L_2(t)\}$;
end

Algorithm 21: Distributed algorithm `NashTwo` for two identical machines.

Next we present a modification of algorithm `BALANCE`, algorithm `NashTwo` which is a Nash rebalancing policy in every stage, the algorithm appears at the top of the next page.

Lemma 12.37. *At every stage Algorithm* `NashTwo` *is a Nash rebalancing policy.*

Proof. As in Lemma 12.36 we compare the load on two machines, one with load $n+d$ and the other with load $n-d$, when excluding one user on the overloaded machine. Since $d_t = 2d$, we have

$$\begin{aligned} L_1 &= (n+d-1)(1 - \frac{2d-1}{2(n+d)-2}) = n - 1/2 \\ L_2 &= n - d + (n+d-1)(\frac{2d-1}{2(n+d)-2}) = n - 1/2. \end{aligned}$$

Therefore, a greedy user on the overloaded machine has no incentive to deviate from the joint strategy. As before, users from the underloaded machine can only lose by trying to rebalance to the overloaded machine. $\square$

Our next aim is to show that the convergence rate of `NashTwo` is similar to that of `BALANCE`. (The Theorem is a private case of Theorem 12.53 and we omit its proof)

Theorem 12.38. *Algorithm* `NashTwo` *is a Nash rebalancing policy and with probability at least* $1-\delta$*, reaches a Nash Equilibrium, within* $O(\ln\ln(n)) + \tilde{O}(\ln(1/\delta))$ *stages. Also Algorithm* `NashTwo` *terminates within expected* $O(\ln\ln(n))$ *stages.*

Different Speed Machines

In this subsection we consider machines with different speeds, specifically, $s_1 = 1$ and $s_2 = \alpha \geq 1$. The resulting Nash equilibrium is not necessarily unique. More precisely, any $n_t(1)$ is Nash equilibrium if and only if $n_t(1) \in [(n-\alpha)/(1+\alpha), (n+1)/(1+\alpha)]$. Recall that $\bar{L} = n/(1+\alpha)$.

Input : Machines: M_1, M_2 with speeds $s_1 = 1$ and $s_2 = \alpha > 1$, Users: $X = \{x_1, \ldots, x_n\}$;
Let $t = 1$; $\texttt{over} = \arg\max\{L_1(M_1), L_1(M_2)\}$;
$\texttt{under} = \arg\min\{L_1(M_1), L_1(M_2)\}$;
while $|L_t(M_1) - L_t(M_2)| > 1/s_{\texttt{under}}$ **do**
  for x_i $s.t$ $g_t(x_i) = M_{\texttt{over}}$ **do in parallel**
    Move with probability $p(n_t(\texttt{over}), s_{\texttt{over}})$
  end
  $t = t + 1$; $\texttt{over} = \arg\max\{L_t(M_1), L_t(M_2)\}$,
  $\texttt{under} = \arg\min\{L_t(M_1), L_t(M_2)\}$;
end

Algorithm 22: Distributed algorithm for two machines with speeds. `BalanceSpeeds` is derived by taking $p(n_t(\texttt{over}), s_{\texttt{over}}) = \frac{n_t(\texttt{over}) - s_{\texttt{over}}\bar{L}}{n_t(\texttt{over})}$ and `NashTwoSpeeds` is derived by taking $p(n_t(\texttt{over}), s_{\texttt{over}}) = \frac{d_t - 1/s_{\texttt{over}}}{(1+1/\alpha)(n_t(\texttt{over})-1)}$

We first derive the convergence rate of algorithm `BalanceSpeeds` (The Theorem is a private case of Theorem 12.53 and we omit its proof)

Theorem 12.39. *Algorithm* `BalanceSpeed` *is a* 1*-Nash rebalancing policy and with probability at least* $1 - \delta$ *it terminates within* $O(\ln\ln(n)) + \tilde{O}(\ln(1/\delta))$ *stages.*

We first derive the convergence rate of algorithm `NashTwoSpeeds` (The Theorem is a private case of Theorem 12.53 and we omit its proof).

Theorem 12.40. *Algorithm* `NashTwoSpeeds` *is Nash rebalancing policy and with probability at least* $1 - \delta$ *it terminates within* $O(\ln\ln(n)) + \tilde{O}(\ln(1/\delta))$ *stages.*

Multiple Weights

In this subsection we consider a slightly different model and assume that each user has a weight and as a consequence in a Nash rebalancing policy only users with same weight on a machine must share a policy. We show that there exists a set of weights and an initial assignment such that any Nash rebalancing policy converges within $\Omega(\sqrt{n})$ even for two machines. [3] We consider n users, where n is a square of an even number. Each user belongs to one of the $\sqrt{n}$ weights classes, where the weight of user in the k class is n^{2k} and the size of each class is $\sqrt{n}$. We say that class i is larger than class j if $i > j$, i.e. the weight of a user in class i is larger than the

weight of those in class j. A Nash rebalancing policy is fully described by $p_1, \ldots, p_{\sqrt{n}}$, where p_i is the probability of user from class i to migrate from the overloaded machine to the underloaded machine.

It is straightforward to see that the only Nash equilibrium of this game is that each class is partitioned equally on the machines. We say that a class i is balanced only if it is partitioned equally and every class j such that $j > i$ is balanced. We also note that after a class is balanced, its users will not move under any Nash rebalancing policy.

The next lemma shows that if a user from the largest unbalanced class is on the underloaded machine, then only the other members of her class move stochastically in the system.

Lemma 12.41. *Let ℓ be the largest unbalanced class, then if there exists a class $k < \ell$ such that $p_k \in (0,1)$ then all users of class ℓ are on the overloaded machine.*

Proof. Suppose by contradiction that there exists a configuration g in which both users from class ℓ and class k move stochastically and that there is at least one user from class ℓ on the underloaded machine. Let $L_u(L_o)$ be the expected load after the users move on the underloaded(overloaded) machine. Since the policy is Nash rebalancing, the following must be satisfied

$$\begin{aligned} L_u - p_\ell w_\ell &= L_o - (1-p_\ell)w_\ell \\ L_u - p_k w_k &= L_o - (1-p_k)w_k. \end{aligned}$$

Simplifying these equations we obtain that $(1-2p_\ell)w_\ell = (1-2p_k)w_k$. Together with the fact that in a Nash rebalancing policy users in the largest unbalanced class move

[3] This result can be extended to a boarder class of policies.

with probability at most $1/2$, we obtain that users in every other unbalanced class that move stochastically move with probability at most $1/2$. Writing $p_i = 1/2 - \varepsilon_i$ for some $\varepsilon_i > 0$, we have $\varepsilon_\ell w_\ell = \varepsilon_k w_k$. Since $w_k/w_\ell \leq 1/n^2$ we have that $\varepsilon_\ell/\varepsilon_k < 1/n^2$, which implies $p_\ell > 1/2 - 1/n^2$. By our assumption all user with weight larger than w_ℓ are already balanced. If X is the current number of users from class ℓ on the overloaded machine, then the expected load after move due to class ℓ on the overloaded machine is at most $X(1/2+1/n^2)w_\ell$ while the expected load on the underloaded machine is at least $(X(1/2-1/n^2)+1)w_\ell$ (using the contradiction assumption). Since the expected load of all users from classes $1, \ldots, \ell-1$ is bounded w_ℓ/n, the difference between L_u and L_o is at least $w_\ell/2$ and users from class k will move deterministically to the overloaded machine, a contradiction to the lemma assumption. □

Next, we complete the characterization of the Nash rebalancing policy.

Lemma 12.42. *Let ℓ be the largest unbalanced class, if there are users from class ℓ on the underloaded machine, then all users from smaller classes move to the underloaded machine deterministically.*

Proof. We first note that by Lemma 12.41 only users of class ℓ move stochastically. We denote $k = \sqrt{n}/2$, and assume that on the overloaded machine that are $k+d$ users from class ℓ and on the underloaded machine there are $k-d$ users, where $1 \leq d \leq k-1$. For any policy we let w_u and w_o be the expected weight of classes $1, \ldots, \ell-1$ on the underloaded and overloaded machines. Hence, $p_\ell = \frac{d-1/2}{k+d-1} + \frac{w_o - w_u}{(k+d-1)n^{2\ell}}$, together with the fact that the load of all users from classes $1, \ldots, \ell-1$ is bounded by $n^{2\ell}/n$, we obtain that

$$\frac{1}{3\sqrt{n}} < \frac{1}{2k} - \frac{1}{n} \leq p_\ell \leq \frac{k-1.5}{2k-2} + \frac{1}{n} < \frac{1}{2} - \frac{1}{3\sqrt{n}}.$$

This assures that the expected load of the overloaded machine in the next stage is at least larger by $O(n^{2\ell-1/2})$ from the expected load on the underloaded machine, thus all users of classes $1, \ldots, \ell-1$ will move deterministically to the underloaded machine. □

In the following proof we use the previous lemma to show that the classes are balanced sequentially.

Theorem 12.43. *There exists a set of weights and an initial assignment such that every Nash rebalancing policy converges in $\Omega(\sqrt{n})$ steps with probability at least $1 - O(1/n^{1/4})$.*

Proof. The initial configuration is that all users are on one machine. We let s_i be the configuration in which all users from the unbalanced classes $1, \ldots, i$ are on one machine and classes $i+1, \ldots, \sqrt{n}$ are balanced and let $S = \{s_1, \ldots, s_{\sqrt{n}}\}$ be the set of these configurations. Although most of the time the configuration is not expected to be inside the set S, we show that under any Nash rebalancing policy the configuration is in S for $\Omega(\sqrt{n})$ times with high probability. We note that the in state s_i in any Nash rebalancing policy, all users from classes $1, \ldots, i$ move with probability $1/2$ and users from classes $i + 1, \ldots, \sqrt{n}$ stay deterministically.

Each time that the system is in $s_i \in S$,($i \geq 3$), the probability that the largest three classes are balanced at once is at most $O\left((\frac{1}{n^{1/4}})^3\right)$. The probability that we do not balance the three largest classes at once is at least $1 - O(1/n^{1/4})$ in first $\sqrt{n}$ times we are in S. Therefore, by Lemma 12.42 we reach s_i for $i \in [\ell - 4, \ell - 1]$. Thus, we reach at least $\Omega(\sqrt{n})$ configurations in the set S with probability at least $1 - O(1/n^{1/4})$. $\square$

12.2.2 Multiple Machines

Multiple identical machines, $n = m$

In this subsection we consider the case where the number of users is equal to the number of machines and all machines are identical. The algorithm that we analyze this section is different from other algorithms in this paper, in the sense that a user can rebalance from an overloaded machine to a different overloaded machine. Algorithm `RandomJump` works as follows: in each stage a user from an overloaded machine assigns equal probability to rebalance to any unbalanced machine (overloaded or underloaded). This algorithm was first given in [3], later it was demonstrated to converge empirically in logarithmic time [46] but only a linear upper bound was given there. We show a logarithmic convergence bound.

Theorem 12.44. *For $n = m$ algorithm* `RandomJump` *terminates within $O(\ln(n) + \ln(1/\delta))$ with probability at least $1 - \delta$.*

Proof. We note that for this special case an underloaded machine is an empty machine and a stable machine is a machine with one user on it. Therefore, every user on an overloaded machine with load $d + 1$ rebalances with probability $\frac{d}{d+1}$ and the target machine is chosen randomly. We first show that the probability that an underloaded machine becomes balanced in one step is $\frac{1}{e}$. Suppose that in the k*th* iteration there are τ underloaded machines, then there $\tau + \ell$ users in ℓ overloaded machines. We let E be the event that an underloaded machine becomes stable, i.e. only one user jumps to it. This probability can be lower bounded as follows,

$$\begin{aligned} \Pr(E) &= \binom{\ell+\tau}{1}(1 - \frac{1}{\ell+\tau})^{\ell+\tau-1}\frac{1}{\ell+\tau} \\ &= (1 - \frac{1}{\ell+\tau})^{\ell+\tau-1} \geq \frac{1}{e} \end{aligned}$$

Our second argument is that each machine becomes either stable or underloaded with probability $\frac{1}{2}$ in the next iteration. This is true due to the fact that for each machine we have a binomial distribution, where its mean is the stable point. Therefore, the probability that a machine becomes stable in two iterations is at least $\frac{1}{2e}$. Using the union bound we obtain that after $2e(\ln(m) + \ln(1/\delta))$ all machines are stable with probability at least $1 - \delta$. ☐

Multiple Machines

In this subsection we consider multiple machines of different speeds. We assume that the speeds are sorted, i.e., $s_1 = 1 \leq s_2 \leq \cdots \leq s_m$. We derive a variant of the `Balance` algorithm for the case of multiple machines with different speeds.

We first note that algorithm `BalanceMultipleSpeed` is a 1-Nash rebalancing policy.

Lemma 12.45. *Algorithm* `BalanceMultipleSpeed` *is a 1-Nash rebalancing policy.*

Proof. Consider any stage before reaching Nash equilibrium. Let $M_1, \ldots, M_k$ be the underloaded machines, $M_k, \ldots, M_\ell$ the balanced machines and $M_{\ell+1}, \ldots, M_m$ be

Input : Machines: $M_1, \ldots, M_m$, users $X = \{x_1, \ldots, x_n\}$
$t = 1$; For every $i \in \{1, \ldots, m\}$ let $d_t(i) = n_t(i) - s_i \bar{L}$;
$\texttt{OVER}_t = \{i | d_t(i) > 0\}$; $\texttt{UNDER}_t = \{i | d_t(i) < 0\}$; $d(t) = \sum_{i \in \texttt{OVER}_t} d_t(i)$;
while $\exists i, j : L_t(M_i) - L_t(M_j) > 1/s_j$ **do**
  for $g_t(x_i) \in \texttt{OVER}_t$ **do in parallel**
    Move with probability $d_t(g_t(x_i))/n_t(g_t(x_i))$
    if x_i *moves* **then**
      Move to machine $j \in \texttt{UNDER}_t$ with probability $\frac{|d_t(j)|}{d(t)}$
    end
  end
  $t = t + 1$; For every $i \in \{1, \ldots, m\}$ let $d_t(i) = L_t(M_i) - s_i \bar{L}$;
  $\texttt{OVER}_t = \{i | d_t(i) > 0\}$; $\texttt{UNDER}_t = \{i | d_t(i) < 0\}$ and $d(t) = \sum_{i \in \texttt{OVER}_t} d_t(i)$;
end

Algorithm 23: Distributed algorithm `BalanceMultipleSpeed` for multiple machines

the overloaded machines. Consider a job on an overloaded machine, M_m then it moves with probability $p_1, \ldots, p_k$ to machines $M_1, \ldots, M_k$ respectively and let $q = p_1 + \cdots + p_k$. Now the load on machines $M_1, \ldots, M_k$ excluding the job is $L_i = n/m - p_i$. the load of machines $M_{k+1}, \ldots, M_{m-1}$ is $L_i = n/m$ and the load on machine M_m is $L_m = n/m - (1 - q)$, thus from this job perspective the policy is 1-rebalancing. From the perspective of a job on the underload and balanced machines all load are n/m except the one she observes which is $n/m - 1$. □

Next we characterize a class of policies, which converge fast.

Definition 12.46. A policy is `fast convergent` if, for some constant $c > 0$, it has the following properties:

1. For any configuration, g, machine M, time t and $\delta < 1$, if $d_t = |L(M) - \bar{L}| \geq c \ln(1/\delta)$, then $d_{t+1} \leq \sqrt{c d_t \ln(1/\delta')}$ with probability at least $1 - \delta$.

2. For any configuration, g, machine M, time t and $\delta < 1$, if $d_t \leq c \ln(1/\delta)$, then $d_{t+1} \leq c \ln(1/\delta)$ with probability at least $1 - \delta$.

3. For any configuration g, and machine M, if $d = |L(M) - \bar{L}|$, then M is balanced in on stage with probability at least $O(1/\sqrt{d})$.

Next we show that given a machine under any fast convergent policy, the machine is balanced with high probability in $\tilde{O}(\ln\ln(n) + \ln(1/\delta))$ stages.

Lemma 12.47. *Given a machine M, and any initial configuration, M is balanced after $O(\ln\ln(n)) + \tilde{O}(\ln(1/\delta))$ stages with probability at least $1-\delta$ under any fast convergent policy.*

Proof. Let ℓ be an upper bound on the number of stages of a fast convergent algorithm until M is balanced. (We latter fix ℓ and show that it is bounded with high probability.) We divide the execution to two phases, the first is while $L_t(M) - n/m = d_t > c\ln(1/\delta')$ and the second starts when $d_t \le c\ln(1/\delta')$, where $\delta' = \delta/(2\ell)$. We first show that the first phase ends with probability $1-\delta/2$ within $O(\ln\ln(n))$ stages. By Lemma 2.4 with probability at least $1-\delta'$, $d_{t+1} \le \sqrt{cd_t \ln(2/\delta')}$ for every stage $t \le \ell$ in the first phase. Since d_0 is bounded by n, then $d_{t+1} \le cn^{1/2^{t+1}}\ln(1/\delta')$ with high probability. Therefore, after $O(\ln\ln(n))$ stages d_t is bounded by $c\ln(1/\delta')$ and once $d_t \le c\ln(1/\delta')$ it remains so with probability $1-\delta/2$ until the end of stage ℓ by Lemma 2.4. For the second phase, we apply Lemma 12.33 with $\ell = O(\ln\ln\ln(1/\delta'))$, and obtain that a fast convergent algorithm balances M with probability $O(\frac{1}{\ln\ln(1/\delta')})$ in $O(\ln\ln\ln(1/\delta'))$ stages. Therefore, if we make additional $O(\ln\ln(1/\delta')\ln\ln\ln(1/\delta')\ln(2/\delta))$ stages, we will terminate with probability at least $1-\delta/2$. Solving the following equation

$$\ell = c_1\ln\ln(n) + c_2\ln(\ln(\ell/\delta))\ln\ln(\ln(\ell/\delta))\ln(2/\delta) + c_3,$$

we arrive at $\ell = O(\ln(1/\delta)\ln(\ln(1/\delta)\ln\ln(\ln(1/\delta)) + \ln\ln(n)) = O(\ln\ln(n)) + \tilde{O}(\ln(1/\delta))$. $\square$

The next proposition shows that a fast convergent policy indeed converges fast to Nash equilibrium, later we show that both `BalanceMultipleSpeed` and Nash rebalancing policy are fast convergent policies and enjoy rapid convergence rate.

Proposition 12.48. *If in every stage a fast convergent policy is used, the all machines are balanced after $O(\ln\ln(n)) + \tilde{O}(\ln(m/\delta))$ stages with probability at least $1-\delta$.*

Proof. By Lemma 12.47, each machine is balanced with probability at least $1-\frac{\delta}{m}$ in $O(\ln\ln(n))+\tilde{O}(\ln(m/\delta))$ stages. Taking the union bound we obtain that all machines are balanced in $\tilde{O}\left(\ln\ln(n)+\ln(m/\delta)\right)$ stages with probability at least $1-\delta$. □

The next lemma shows that Algorithm `BalanceMultipleSpeed` produces a fast convergent policy.

Lemma 12.49. *The policy generated by Algorithm* `BalanceMultipleSpeed` *is a fast convergent policy.*

Proof. Given a machine M at time t, for any configuration g we let $d_t = L_t(M) - L$. We show that the policy generated by Algorithm `BalanceMultipleSpeed` satisfies all three properties of the fast convergent policy class. We begin by showing that if $|d_t| \geq 3\ln(1/\delta)$, then $|d_{t+1}| \leq \sqrt{3|d_t|\ln(1/\delta)}$. When $d_t \geq 3\ln(1/\delta)$ we apply Lemma 2.4 with $n/m+d_t$ i.i.d random variables, $Z_1,\ldots,Z_{n/m+d_t}$, where $\sum_{i=1}^{n/m+d_t} E[Z_i] = \frac{d_t}{n/m+d_t}$, thus $d_{t+1} \leq \sqrt{3d_t\ln(1/\delta)}$ with probability at least $1-\delta$. When $d_t \leq -3\ln(1/\delta)$ we apply the relative Chernoff bound with K independent random variables, $Z_1,\ldots,Z_K$, where $n/m+d_t < K < n-n/m+d_t$ and $\sum_{i=1}^{K} E[Z_i] = d_t$, to obtain that $|d_{t+1}| \leq \sqrt{3|d_t|\ln(1/\delta)}$. Showing that if $|d_t| \leq 3\ln(1/\delta)$, then $|d_{t+1}| \leq 3\ln(1/\delta)$, is done similarly using the relative Chernoff bound. By Lemma 12.32 the probability of terminating in one stage (for both underloaded and overloaded machines) is at least $O(\sqrt{1/|d_t|})$. □

By Lemma 12.49 and Proposition 12.48 we derive the following theorem.

Theorem 12.50. *With probability at least* $1-\delta$ *algorithm* `BalanceMultipleSpeed` *terminates within* $O(\ln\ln(n))+\tilde{O}(\ln(m/\delta))$ *stages, and it terminates in expected* $O(\ln\ln(n)+\ln(m))$ *stages.*

12.2.3 Nash Rerouting policies for Multiple Machines

We define a very natural class of policies, where users rebalance only from overloaded machines and only to underloaded machines. In addition we require that the deviation from a Nash rebalancing policy would not be too large. This set clearly includes the various `BALANCE` policies. We will show that this set also includes a Nash rebalancing policy. In addition, any rebalancing, which at every stage uses some policy from this set, converges fast.

Definition 12.51. For any $g \in G$ let `OVER`(`UNDER`) be the set of overloaded (underloaded) machines in g. Let $\Pi(g)$ be the set of user policies satisfying the following properties for $P \in \Pi(g)$:

1. Every user on a machine in `OVER` does not move to a different machine in `OVER`. I.e., for every i such that $g(i), j \in$ `OVER` and $g(i) \neq j$ we have $p_i(j) = 0$.

2. Every user on a machine in `UNDER` does not move to any other machine. I.e., for every $i \in$ `UNDER` we have $p_i(g(i)) = 1$.

3. The expected load of every machine M_i, with respect to P, is in the interval $(\bar{L} - 1/s_i, \bar{L} + 1/s_i)$

4. All the users on a machine share the same policy. I.e., for every M_i for all users j and k such that $g(j) = i = g(k)$ we have $P_j = P_k$.

The lemma next relates the class $\Pi(g)$ to the fast convergent policies class.

Lemma 12.52. *If for every configuration, g, a policy in $\Pi(g)$ is used then the resulting policy is a fast convergent policy.*

Proof. We start by showing that the properties are satisfied for both overloaded and underloaded machine machine, since either $\lfloor \bar{L} \rfloor = n/m$ or $\lceil \bar{L} \rceil = n/m$ and jobs only move from overloaded machines, we can apply the relative Chernoff bound with respect to d_t and thus properties 1 and 2 of the fast convergent, and apply Lemma 12.32 and obtain property 3. □

The following theorem relates $\Pi(g)$ to the convergence rate and its proof follows immediately from Lemma 12.52 and Proposition 12.48.

Theorem 12.53. *If during every stage of the rebalancing some policy in $P^t \in \Pi(g_t)$ is used, then the system converges to Nash equilibrium, with probability at least $1 - \delta$, within $O(\ln \ln(n)) + \tilde{O}(\ln(m/\delta))$ stages.*

Next we show that for any configuration $g \in G$, the set $\Pi(g)$ contains a Nash rebalancing policy for g. We first give characterization of every Nash equilibria which will be useful in the proof of Theorem 12.55.

Lemma 12.54. *In every Nash equilibrium, in which all users on the same machine share the same policy, if a user from an overloaded machine M_i rebalances with positive probability to an underloaded machine M_j at time t, then $E[L_{t+1}(M_i)] \geq E[L_t(M_j)]$*

Proof. Since the system is in Nash equilibrium, then the expected load on the machines excluding one user on M_i is identical.
Suppose by contradiction that $E[L_{t+1}(M_i)] < E[L_t(M_j)]$, thus $p_i^t(j) > p_i^t(i)$ but this implies that load on M_j is larger than the load on M_i excluding the last user, which is contradiction. □

Theorem 12.55. *For every configuration $g \in G$, there exists a Nash rebalancing policy in $\Pi(g)$.*

Proof. We first note that any Nash equilibrium must satisfy the third condition. By restricting users actions in the game we can obtain the first and second condition. Therefore, it remains to prove that there exists an equilibrium in which all the users on a machine share the same policy. We prove it by using Brouwer Fixed Point Theorem [21]. (The proof is almost identical to the Nash equilibrium existence proof.) Let Δ be the set of all possible policies, i.e. if $P \in \Delta$ then $P_{i,j}$ is the probability of the user i to move from its machine $g(i)$ to machine j and $\sum_j P_{i,j} = 1$. For every policy, P we define the policy $\bar{P}_{i,j} = \frac{\sum_{i':g(i)=g(i')} P_{i',j}}{\sum_{i':g(i)=g(i')} \mathbf{1}_{P_{i',j}}}$, where $\mathbf{1}_{P_{i',j}}$ is 1 if $P_{i',j} > 0$ and 0 otherwise. Intuitively, $\bar{P}$ averages the policies of users that share the same machine, it remains to show that $\bar{P}$ can be an equilibrium. We define the utility function of the distribution $\bar{P}$ as,

$$u_i(\bar{P}) = E_{j \backsim \bar{P}_{i,j}} E_{\bar{P}_{-i}}[L(M_j)] + 1/s_j$$

Next we define the utility of switching deterministically to machine j.

$$Z_{i,j} = \max\{u_i(j, \bar{P}_{-i}) - u_i(\bar{P}), 0\}$$

After defining these rewards define the mapping $y : \Delta \longmapsto \Delta$, with i, j^{th} component

$$y_{i,j} = \frac{p_{i,j} + Z_{i,j}}{1 + \sum_j Z_{i,j}}$$

Note that this mapping is continuous and Δ is a compact and convex set. Therefore, there exists a fixed point for this mapping, we would like to show that there exists at least one fixed point which is an equilibrium and all the users on the same machine share the same policy. We show that if P is an equilibrium then $\bar{P}$ is an equilibrium. If P is an equilibrium then for any machine j such that $P_{i,j} > 0$ the move from $g(i)$ to j is best response with respect to $\bar{P}$. Thus every distribution on these machine is a best response including $\bar{P}$. It remains to show that an equilibrium in the restricted game is an equilibrium in the original game as well. By Lemma 12.54, if a user on an overloaded machine would like to rebalance to a different overloaded machine, then there is also an underloaded machine with less or equal load that he would like to move to, thus the system was not in equilibrium in the restricted game. If a user on an underloaded machine would like to rebalance then there exists a user on an overloaded machine that would like to rebalance to that machine as well, which is contradiction to the the fact that it is an equilibrium in the restricted game. □

Theorem 12.53 and Theorem 12.55 imply that there exists a Nash rebalancing policy that converges in expected $O(\ln \ln n + \ln m)$ stages to Nash equilibrium.

The next observation is regarding the Nash rebalancing policies structure.A two phases rebalancing policy is a policy that behaves as follows: every user from an overloaded machine k leaves machine k with probability p_k. Any user that leaves an overloaded machine, moves to an underloaded machine j with probability q_j. (Note that the various `Balance` algorithms are a two phase rebalancing policy.) We show that for some configurations no two phase rebalancing policy is a Nash rebalancing policy.

Lemma 12.56. *There exists a configuration $g \in G$, in which no two phases rebalancing policy is a Nash rebalancing policy.*

Proof. We consider a case where there are four machines; Two overloaded machines

M_1, M_2 with load $n + d_1, n + d_2$ and two underloaded machines M_3, M_4 with load $n - d_3, n - d_4$. We let $p_1(p_2)$ be the probability leaving $M_1(M_2)$ and $q_3(q_4)$ be the probabilities of reaching $M_3(M_4)$ if a user moves. Clearly all probabilities are strictly positive. A Nash equilibrium must satisfy the following equations:

$$
\begin{aligned}
(n + d_1 - 1)(1 - p_1) &= n - d_3 + p_1(n + d_1 - 1)q_3 + p_2(n + d_2)q_3 \\
(n + d_1 - 1)(1 - p_1) &= n - d_4 + p_1(n + d_1 - 1)q_4 + p_2(n + d_2)q_4 \\
(n + d_2 - 1)(1 - p_2) &= n - d_3 + p_1(n + d_1)q_3 + p_2(n + d_2 - 1)q_3 \\
(n + d_2 - 1)(1 - p_2) &= n - d_3 + p_2(n + d_1)q_4 + p_2(n + d_2 - 1)q_4
\end{aligned}
$$

By subtracting Eq.1 form Eq.2 and Eq.3 from Eq.4 we obtain,

$$d_4 - d_3 = p_1(n + d_1 - 1)(q_4 - q_3) + p_2(n + d_2)(q_4 - q_3)$$

$$d_4 - d_3 = p_1(n + d_1)(q_4 - q_3) + p_2(n + d_2 - 1)(q_4 - q_3).$$

Therefore, $p_1(q_4 - q_3) = p_2(q_4 - q_3)$ which implies that either $p_1 = p_2$ or $q_3 = q_4$. By taking $d_1 = n, d_2 = 1, d_3 = 1, d_4 = n$, we show that both conditions cannot be satisfied. If $q_3 = q_4$ then the load on M_4 would be strictly smaller which is contradiction. If $p_1 = p_2$, then either the load on M_1 is greater then $n + 2$ or the load on M_2 is smaller than $n - 1$. In the first case a user on M_1 would leave with probability 1 and in the second case a user on M_2 would stay with probability 1. $\square$

Next we present an algorithm, which computes Nash rebalancing policy. The algorithm is based on the fact that given the support of each user, we can compute the Nash rebalancing policy by solving set of linear equations. Thus by enumerating all possible supports of all users, we can compute a Nash rebalancing policy.

Theorem 12.57. *Computing a Nash rebalancing policy can be done in* $O(2^{m^2})$.

Proof. The main observation is that once we have the set of probabilities $p_i(j) > 0$, we can compute the actual probabilities by solving a linear system. A bruteforce search for the support requires $O(2^{m^2})$. $\square$

Bibliography

[1] W. A. Aiello, Y. Mansour, S. Rajagopolan, and A. Rosen. Competitive queue policies for differentiated services. In *INFOCOM*, 2000. (To appear in J. of Algorithms).

[2] E. Allender, S. Arora, M. Kearns, C. Moore, and A. Russell. Note on the representational incompatabilty of function approximation and factored dynamics. In *Advances in Neural Information Processing Systems 15*, pages 431–437, 2002.

[3] S. Alpern and D. Reyniers. Spatial dispersion as a dynamic coordination problem. *Theory & Decision*, 53:1:29–59, 2002.

[4] D. Angluin and L. G. Valiant. Fast probabilistic algorithms for hamiltonian circuits and matchings. *Journal of Computer and System Sciences*, 18:155–193, 1979.

[5] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. Gambling in a rigged casino: The adversarial multi-armed bandit problem. In *Proc. 36th Annual Symposium on Foundations of Computer Science*, pages 322–331. IEEE Computer Society Press, 1995.

[6] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. The non-stochastic multi-armed bandit problem. *SIAM J. on Computing*, 32(1):48–77, 2002.

[7] Peter Auer, Nicolo Cesa-Bianci, and Claudio Gentile. Adaptive and self-confident on-line learning algorithms. *Journal of Computer System Sciences*, pages 48–75, 2002.

[8] Baruch Awerbuch and Robert D. Kleinberg. Adaptive routing with end-to-end feedback: distributed learning and geometric approaches. In *STOC*, pages 45–53. ACM Press, 2004.

[9] Y. Azar. *On-line Load Balancing Online Algorithms - The State of the Art*, chapter 8, pages 178–195. Springer, 1998.

[10] Amos Beimel, Francesco Bergadano, Nader H. Bshouty, Eyal Kushilevitz, and Stefano Varricchio. Learning functions represented as multiplicity automata. *J. ACM*, 47(3):506–530, 2000.

[11] D.A. Berry and B. Fristedt. *Bandit Problems*. Chapman and Hall, 1985.

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

[13] D.P. Bertsekas and J.N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1995.

[14] A. Blum and A. Kalai. Universal portfolios with and without transaction costs. *Machine Learning*, 35:193–205, 1999.

[15] B. Bonet. An ϵ-optimal grid-based algorithm for Partially Observable Markov Decision Processes. In *Proc. 19th International Conf. on Machine Learning*, pages 51–58. Morgan Kaufmann, 2002.

[16] T. Boulogne, E. Altman, and O. Pourtallier. On the convergence to nash equilibrium in problems of distributed computing. *Annals of Operation research*, 109(1-4):279–291, 2002.

[17] Craig Boutilier, Thomas Dean, and Steve Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

[18] X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In *Proceedings of 14th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 206–215, 1998.

[19] R. Brafman. A heuristic variable grid solution method for pomdps. In *AAAI-97*, pages 727–733, 1997.

[20] R. Brafman and M. Tennenholtz. R-max – a general polynomial time algorithm for near optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231, 2002.

[21] L. E. J. Brouwer. Uber abbildung von mannigfaltigkeiten. *Mathematische Annalen*, 71:97–115, 1912.

[22] J.W Carlyle and A. Paz. Realization by stochastic finite automaton. *J. Comput. Syst. Sci.*, 5:26–40, 1971.

[23] A. Czumaj, P. Krysta, and B. Vocking. Selfish traffic allocation for server farms. In *Proceedings of the 34th Symposium on Theory of Computing*, pages 287–296, 2002.

[24] A. Czumaj and B. Vocking. Tight bounds on worse case equilibria. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 413–420, 2002.

[25] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150, 1989.

[26] Infinite-Horizon Policy-Gradient Estimation. J. baxter and p.l bartlett. *Journal of Artificial Intelligence Research*, 15:319–350, 2001.

[27] E. Even-Dar, Sham Kakade, and Y. Mansour. Experts in a Markov decision process. In *Proceedings of Advances in Neural Information Processing Systems 17 (NIPS)*, pages 401–408, 2004.

[28] E. Even-Dar, Sham Kakade, and Y. Mansour. Planning in pomdps using multiplicity automata. In *Proceedings of 21st Conference on Uncertainty in Artificial Intelligence (UAI)*, 2005.

[29] E. Even-Dar, Sham Kakade, and Y. Mansour. Reinforcement learning in pomdps without resets. In *Proceedings of Nineteenth International Joint Conference on Artificial Intelligence (IJCAI)*, 2005.

[30] E. Even-Dar, A. Kesselman, and Y. Mansour. Convergence time to nash equilibria. In *Proceedings of the 30th International Conference on Automata, Languages and Programming (ICALP)*, pages 502–513, 2003.

[31] E. Even-Dar, S. Mannor, and Y. Mansour. Pac bounds for multi-armed bandit and Markov decision processes. In *Proceedings of Fifteenth Annual Conference on Computational Learning Theory (COLT)*, pages 255–270, 2002.

[32] E. Even-Dar, S. Mannor, and Y. Mansour. Action elimination and stopping condition for reinforcement learning. In *Proceedings of 20th International Conference on Machine Learning (ICML)*, pages 162–169, 2003.

[33] E. Even-Dar and Y. Mansour. Convergence of optimistic and incremental Q-learning. In *Proceedings of Advances in Neural Information Processing Systems 14 (NIPS)*, pages 1499–1506, 2001.

[34] E. Even-Dar and Y. Mansour. Approximate equivalence of Markov decision processes. In *Proceedings of Sixteenth Annual Conference on Computational Learning Theory (COLT)*, pages 581–594, 2003.

[35] E. Even-Dar and Y. Mansour. Learning rates for Q-learning. *Journal of Machine Learning Research*, 5:1–25, 2003. (A preliminary version appeared in the Fourteenth Annual Conference on Computation Learning Theory (2001), 589-604.).

[36] E. Even-Dar and Y. Mansour. Fast convergence of selfish rerouting. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 772–781, 2005.

[37] M. Fliess. Matrices dehankel. *J. Math. Pures Appl.*, 53:197–222, 1974.

[38] D. Fotakis, S. Kontogiannis, E. Koutsoupias, M. Mavronicolas, and P. Spirakis. The structure and complexity of nash equilibria for a selfish routing game. In *Proceedings of the 29th ICALP*, pages 123–134, 2002.

[39] Y. Freund and R. Schapire. Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, 29:79–103, November 1999.

[40] D. Fudenberg and D. Levine. *The theory of learning in games.* MIT Press, 1998.

[41] J. Gittins and D. Jones. A dynamic allocation index for the sequential design of experiments. In J. Gani, K. Sarkadi, and I. Vincze, editors, *Progress in Statistics*, pages 241–266. North-Holland, Amsterdam, 1974.

[42] Robert Givan, Thomas Dean, and Matthew Greig. Equivalence notions and model minimzation in markov decision processes. *Artificial Intelligence*, pages 163–223, 2003.

[43] Robert Givan, Sonia Leach, and Thomas Dean. Bounded parameter markov decision processes. *Artificial Intelligence*, 122:71–109, 2000.

[44] Paul Goldberg. Bounds for the convergence rate of randomized local search in multiplayer games, uniform resource sharing game. In *In Proceedings of the Twenty-Third PODC*, pages 131–140, 2004.

[45] T.F. Gonzalez. Clustering to minimize the maximum inter-cluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

[46] T. Grenager, R. Powers, and Y. Shoham. Dispersion games: general definitions and some specific learning results. In *Eighteenth national conference on Artificial intelligence*, pages 398–403, 2002.

[47] Carlos Guestrin, Daphne Koller, and Ronald Parr. Max-norm projections for factored MDPs. In *IJCAI*, pages 673–682, 2001.

[48] J. Hannan. Approximation to Bayes risk in repeated play. In M. Dresher, A. W. Tucker, and P. Wolde, editors, *Contribution to The Theory of Games, III*, pages 97–139. Princeton University Press, 1957.

[49] M. Hauskrecht. A heuristic variable-grid solution method for pomdps. In *AAAI-97*, pages 734–739, 1997.

[50] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[51] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[52] R. Howard. *Dynamic programming and Markov decision processes*. MIT press, 1960.

[53] Tommi Jaakkola, Michael I. Jordan, and Satinder P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation, 6*, 1994.

[54] S. Kakade. *On the Sample Complexity of Reinforcement Learning*. PhD thesis, University College London, 2003.

[55] A. Kalai and S. Vempala. Efficient algorithms for on-line optimization. *Proceedings of Sixteenth Annual Conference on Computational Learning Theory (COLT)*, pages 26–40, 2003.

[56] M. Kearns, Y. Mansour, and A. Ng. Approximate planning in large pomdps via reusable trajectories. In *NIPS 12*, 1999.

[57] M. Kearns and S. Singh. Finite-sample convergence rates for q-learning and indirect algorithms. In *NIPS*, pages 996–1002, 1998.

[58] M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2-3):209–232, 2002. (A preliminary version appeared in ICML (1998), 260-268.).

[59] M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2-3):209–232, 2002. (A preliminary version appeared in ICML (1998), 260-268.).

[60] Michael Kearns and Satinder P. Singh. Finite-sample convergence rates for Q-learning and indirect algorithms. In *Neural Information Processing Systems 10*, pages 996–1002, 1998.

[61] Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory*. The MIT Press, 1994.

[62] A. Kesselman, Z. Lotker, Y. Mansour, B. Patt-Shamir, B. Schieber, and M. Sviridenko. Buffer overflow management in QoS switches. *SIAM J. on Computing*, 33(3):563–583, 2004. (A preliminary version appeared in ACM Symposium on Theory of Computing (2001), 520-529.).

[63] Daphne Koller and Ronald Parr. Computing factored value functions for policies in structured MDPs. In *IJCAI*, pages 1332–1339, 1999.

[64] E. Koutsoupias and C. H. Papadimitriou. Worst-case equilibria. In *Proceedings of 16th STACS*, pages 404–413, 1999.

[65] T.L. Lai and H. Robbins. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6:4–22, 1985.

[66] L. Libman and A. Orda. Atomic resource sharing in noncooperative networks. *Telecommunication Systems*, 17:4:385–409, 2001.

[67] N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994.

[68] Michael Littman, Richard Sutton, and Satinder Singh. Predictive representations of state. In *Advances in Neural Information Processing Systems 14 (NIPS)*, pages 1555–1561, 2001.

[69] Michael L. Littman and Gaba Szepesvári. A generalized reinforcement-learning model: Convergence and applications. In L. Saitta, editor, *Proceedings of the 13th International Conference on Machine Learning (ICML-96)*, pages 310–318, Bari, Italy, 1996. Morgan Kaufmann.

[70] W. S. Lovejoy. Computationally feasible bounds for partially observed markov decision processes. *Operations Research*, 39(1):162–175, 1991.

[71] C. Lusena, J. Goldsmith, and M. Mundhenk. Nonapproximability results for partially observable markov decision processes. *Journal of Artificial Intelligence Research*, 14:83–103, 2001.

[72] Omid Madani, Steve Hanks, and Anne Condon. On the undecidability of probabilistic planning and infinite-horizon partially observable markov decision problems. In *AAAI/IAAI*, pages 541–548, 1999.

[73] S. Mannor and J. N. Tsitsiklis. The sample complexity of exploration in the multi-armed bandit problem. *Journal of Machine Learning Research*, 5:623–648, 2004. (A preliminary version appeared in the Sixteenth Annual Conference on Computation Learning Theory (2003), 418-432.).

[74] Yishay Mansour and Satinder Singh. On the complexity of policy iteration. In *UAI*, pages 401–408, 1999.

[75] P. Marbach. Priority service and max-min fairness. *IEEE/ACM Trans. Netw.*, 11:5:733–746, 2003.

[76] P. Marbach and J. Tsitsiklis. Simulation-based optimization of markov reward processes. *IEEE Trans. on Automatic Control*, 46(2):191–209, 2001.

[77] D. McAllester and S. Singh. Approximate planning for factored pomdps using belief state simplification. In *In Proceedings of the Fifteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 409–416, 1999.

[78] H. McMahan, G. Gordon, and A. Blum. Personal communication. 2003.

[79] H. McMahan, G. Gordon, and A. Blum. Planning in the presence of cost functions controlled by an adversary. In *Proceedings of 20th International Conference on Machine Learning (ICML)*, 2003.

[80] I. Milchtaich. Congestion games with player-specific payoff functions. *Games and Economic Behavior*, 13:111–124, 1996.

[81] V. Mirrokni and A. Vetta. Convergence issues in competitive games. In *Proceedings of the 7th International Workshop on Approximation Algorithms for Combinatorial Optimization*, 2004.

[82] D. Monderer and L. S. Shapley. Potential games. *Games and Economic Behavior*, 14:124–143, 1996.

[83] J. F. Nash. Non-cooperative games. *Annals of Mathematics*, 54:286–295, 1951.

[84] A. Orda, N. Rom, and N. Shimkin. Competitive routing in multi-user communication networks. *IEEE/ACM Transaction on Networking*, 1:614–627, 1993.

[85] A. Orda, R. Rom, and N. Shimkin. Competitive routing in multi-user communication networks. *IEEE/ACM Transaction on Networking*, 1:614–627, 1993.

[86] R. Parr and S. Russell. Approximating optimal policies for partially observable stochastic domains. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1995.

[87] L. Peshkin, K. Kim, N. Meuleau, and L.P. Kaelbling. Learning to cooperate via policy search. In *16th Proceedings of UAI*, pages 307–314, 2000.

[88] M. Puterman. *Markov Decision Processes*. Wiley-Interscience, 1994.

[89] R. Rivest and R. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 103(2):299–347, 1993.

[90] H. Robbins. Some aspects of sequential design of experiments. *Bull. Amer. Math. Soc.*, 55:527–535, 1952.

[91] T. Roughgarden and E. Tardos. How bad is selfish routing? *Journal of the ACM*, 49(2):236–259, 2002.

[92] P. Schuurman and T. Vredeveld. Performance guarantees of local search for multiprocessor scheduling. In *Proceedings IPCO*, pages 370–382, 2001.

[93] M.P Shlitzenberger. On the definition of a family of automata. *Inf. Control*, 4:245–270, 1961.

[94] Satinder Singh and David Cohn. How to dynamically merge markov decision processes. In *Advances in Neural Information Processing Systems 10 (NIPS)*, pages 1057–1063, 1998.

[95] Satinder Singh, Michael R. James, and Matthew R. Rudary. Predictive state representations: A new theory for modeling dynamical systems. In *Uncertainty in Artificial Intelligence: Proceedings of the Twentieth Conference (UAI)*, pages 512–519, 2004.

[96] Satinder Singh, Michael Littman, Nicholas Jong, David Pardoe, and Peter Stone. Learning predictive state representations. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML)*, pages 712–719, 2003.

[97] Satinder P. Singh and Richard C. Yee. An upper bound on the loss from approximate optimal-value functions. *Machine Learning*, 16(3):227–233, 1994.

[98] E. Sondik. *The optimal control of partially observable processes over a finite horizon.* PhD thesis, Stanford University, Stanford, California, 1971.

[99] R. Sutton and A. Barto. *Reinforcement Learning. An Introduction.* MIT Press, Cambridge, MA, 1998.

[100] J. Tsitsiklis. Asynchronous stochastic approximation and q-learning, 1994.

[101] J. N. Tsitsiklis and B. Van Roy. Feature-based methods for large scale dynamic programming. *Machine Learning*, 22:59–94, 1996.

[102] L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984.

[103] C. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, 1989.

[104] C. Watkins and P. Dyan. Q-learning. *Machine Learning*, 8(3/4):279 –292, 1992.

[105] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3-4):229–256, 1992.