Multi-Armed Bandits with Applications to Markov Decision Processes and Scheduling Problems

A Dissertation Presented

by

Isa M. Muqattash

to

The Graduate School

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in

Applied Mathematics and Statistics
(Operations Research)

Stony Brook University

December 2014

Stony Brook University

The Graduate School

Isa M. Muqattash

We, the dissertation committee for the above candidate for the Doctor of Philosophy degree, hereby recommend acceptance of this dissertation.

This dissertation is accepted by the Graduate School.

Abstract of the Dissertation

Multi-Armed Bandits with Applications to Markov Decision Processes and Scheduling Problems

by

Isa M. Muqattash

Doctor of Philosophy

in

Applied Mathematics and Statistics (Operations Research)

Stony Brook University

2014

The focus of this work is on practical applications of stochastic multi-armed bandits (MABs) in two distinctive settings.

First, we develop and present REGA, a novel adaptive sampling-based algorithm for control of finite-horizon Markov decision processes (MDPs) with very large state spaces and small action spaces. We apply a variant of the ϵ-greedy multi-armed bandit algorithm to each stage of the MDP in a recursive manner, thus computing an estimation of the "reward-to-go" value at each stage of the MDP. We provide a finite-time analysis of REGA. In particular, we provide a bound on the probability that the approximation error exceeds a given threshold, where the bound is given in terms of the number of samples collected at each stage of the MDP. We empirically compare REGA against other sampling-based algorithms and find that our algorithm is competitive. We discuss measures to aid against the curse of dimensionality due to the backwards induction nature of REGA, necessary when the MDP horizon is large.

Second, we introduce e-Discovery, a topic of extreme significance to the legal industry, which pertains to the ability of sifting through large volumes of data in order to identify the "needle in the haystack" documents relevant to a lawsuit or investigation. Surprisingly, the topic has not been explicitly investigated in academia. Looking at the problem from a scheduling perspective, we highlight the main properties and challenges pertaining to this topic and outline a formal model for the problem. We examine an approach based on related work from the field of scheduling theory and provide simulation results that demonstrate the performance of our approach against a very large data set. We also provide an approach based on list-scheduling that incorporates a side multi-armed bandit in lieu of standard heuristics. Necessarily, we propose the first MAB algorithm that accounts for both sleeping bandits and bandits with history. The empirical results are encouraging.

Surveys of multi-armed bandits as well as scheduling theory are included. Many new and known open problems are proposed and/or documented.

Contents

List of Figures

List of Tables

Acknowledgements

To me this has been a journey like no other. I have overcome many difficulties in the past, but this work was completed under special circumstances that have spread my attention very thin across different aspects of life. Nonetheless, it is difficult for me to comprehend that this journey has come to an end. Yes, arriving at the final destination is important and is mine to keep forever, but the steps taken towards this goal will be missed, and I am certain that at some point in my life a portion of the details will become forgotten. I will continue to look back and reflect in awe.

This journey would have been impossible without the significant support that I have received from many. First and foremost, I would like to take a moment to thank my advisor **Professor Jiaqiao Hu** for all the mentoring, support, and guidance that he has provided me over the last few years. I realize how busy you have been. Yet, you somehow always managed to find the time to meet with me and provide me with the necessary feedback. Your ability to quickly spot my mistakes continues to impress me. Thank you for everything that you have done for me.

I would also like to thank the Chair of my dissertation committee **Professor Esther (Estie) Arkin**. Since the time that I first met you in your Linear Programming course, you have been helpful each and every time that I have spoken with you. Your dedication and willingness to help your students is applaudable. Thank you for your "open door" policy, it is the most any student can ask for.

I would like to also thank my committee member **Professor Yuefan Deng**. The time when we met over lunch for a discussion of my thesis topic while I was still struggling to put my thoughts together, you (subconsciously) said something to me that has stuck with me ever since: Since I had been working in the "field" for a little while before returning to academia, you reminded me of the need to reduce the complexity of my models and to make simplifying assumptions in order to be able to tackle more manageable problems. I will never forget what you said. Thank you for that eye-opening comment.

Special thanks to my external committee member **Professor Luis E. Or-**

tiz for taking the time out of your busy schedule. Even though the timing of my defense conflicted with a NSF function in which you were taking part, you insisted that I need not change my defense date. Thank you for pointing out important literature related to the work that appears within this manuscript. I am very glad to have met you during one of the Game Theory sessions that we both attended. I am still very much interested in Game Theory and hope that we can work together in the future.

In parallel to working towards my degree, I have been working fulltime as a software engineer. Leading two different lifestyles at once has not been easy, and a few co-workers in particular deserve special mention. First, my previous manager **Howard Robinson**. I enrolled at Stony Brook University shortly after I started working for you. With my office being 50 miles away in New York City, your flexibility and arranging for me to work some days from "home" on a weekly basis were key in enabling me to attend classes at Stony Brook. You continue to mentor me until this day even though we no longer work together. I'd also like to thank my current manager **Jonathan Cagle** for flexibility in my work schedule in the last few weeks while I focused on writing my dissertation. I truly enjoy working with you. Also to thank is my colleague **Malcolm Pollack** for your frequent words of encouragement. You telling me once that "the five (plus) years would pass by whether you did your PhD or not" has helped propel me since joining this program.

If there is anyone to be thanked for giving me that extra push to "plant the seed" and join a PhD program, that would be my father **Mithqal Muqattash** and mother **Samia Haddad**. Your initial and continued encouragement throughout the entire process have been paramount.

I'd also like to thank my parents-in-law **Thomas** and **Kathie Melton** for their encouragement, support, and understanding each and every time they asked to spend time with me. Yes, I really was that busy! I promise to make up for lost time.

It would be complete injustice to not mention my wife **Lauren Muqattash**. You have endured more than anyone else over the last few years and had to make many selfless sacrifices. On many occasions you had to fill in for me when I was not available. Many times you would wake up in the wee hours of the morning to make sure I was okay. As with everything else in our lives, you have shared with me the entire PhD process day after day. Thank you, and I love you!

My son **Andrew**, at two years old you are too young to understand why Daddy was too busy to spend more time with you. Your frequent visits into my office to give me hugs and kisses are the most sincere a parent can ask for. I owe you a lot of "makeup" time buddy!

To all the others that I have not mentioned, you are too many but each and every one of you knows whom you are. Thank you for being my friends and family.

Chapter 1

Multi-Armed Bandits

1.1 Introduction

Consider a gambler faced with a slot machine with multiple arms, each returning an average reward that is unknown to the gambler. The gambler starts out with a fixed number of tokens and has to decide which arm to play next in order to maximize her total profit. If the average reward of each arm is known, then the optimal strategy is to simply always play the arm with the highest expected average reward. However, the average rewards are unknown to the gambler. Before each play, the gambler has to decide between exploring the payoff models of the arms *vs.* playing the machine believed to be optimal based on her prior observations.

Picture a clinical trial where alternative treatments are possible for some type of disease. The medical staff have to decide which treatment to prescribe to the next patient in order to maximize the likelihood of curing the patient. A decision has to be made whether to test the effectiveness of one of the promising drugs as opposed prescribing the drug found to be most effective to date.

Consider the example of online advertisement placement where the objective is to maximize the total number of advertisement clicks by the website visitors. Several options for product brand, color, font, *etc.* are to be chosen when the next visitor views the web site in order to increase the odds of the visitor clicking the advertisement.

The scenarios outlined above are all examples of sequential decision making with limited available information. They are all examples of the so-called *multi-armed bandit* (MAB) problem. In the most basic setting, a fixed set of actions are available, and the objective of the decision maker is to balance between learning the payoff distribution of each arm (exploration) and taking advantage of prior observations to play next the arm believed to be optimal (exploitation). As we will discuss later, it is not necessarily optimal that the exploration phase be completed before the exploitation phase can begin. This is particularly clear from the advertisement placement example as the set of available actions (arms) can change over time. The study of MABs dates back to the 1930s (*c.f.* [139] pertaining to the aforementioned clinical trials example), and continues to be applied to newer technologies today. The term "bandit" is attributed to the first (classical) casino example mentioned in the beginning of this chapter.

In the remainder of this chapter, we give a brief survey of applications of MABs, provide a light introduction to the MAB framework, summarize the most common types of bandit models, and highlight the most recent developments in the field. [37, 84] provide excellent surveys on the topic of MABs. While [84] focuses on bandits with switching costs, [37] provides a broader "state of the art" coverage of the subject. Sections 1.2 - 1.4.2 are primarily based on their work.

1.2 Sample Applications of the Bandit Model

The work in [84] surveys a wide range of settings where the bandit model lends itself naturally. The reader is encouraged to refer to the original manuscript for the detailed survey, as our objective is limited to providing some examples of applications of MABs. The examples that follow all appear in [84].

One example of the bandit model appears when studying the behavior of workers deciding to migrate to one of several cities since their decisions can be influenced by the (unknown) potential offerings of each city (*c.f.* [103]).

Another obvious example of bandits is optimal search, where the researcher has to decide on the next location to look for an instance of a resource. An example is when mining for oil or other natural resources under finite budget constraints. The trade-off in this case between exploration and exploitation is obvious, and example papers covering this area include [25, 26, 148].

Yet another area where bandits are applicable is when buyers and sellers interact with a market where there is incomplete information pertaining the quality or fair price of a commodity. This setting falls under the broader category of experimentation and learning, and the potential to make future decisions based on past observations naturally fits into the bandit model. Examples of work in this area include [19, 104, 119].

Bandits have several applications in the field of game theory. In section 1.4.2, we discuss the class of adversarial bandits which is motivated by a game-theoretic setting where an agent plays a game against an adversary. Usually, the agent's objective is to achieve a minimax regret while repeating experiments in a setting with incomplete information. This setting can be thought of as a two-person zero-sum game where the adversary (or Mother Nature) attempts to maximize the regret while the agent attempts to minimize the regret (*c.f.* the related work of [122]).

1.3 Regret of a Forecaster

Consider a MAB problem with $K \geq 2$ arms. Suppose that each arm $i = 1, \dots, K$ returns a reward sequence $\{X_{i,t}\}$ at time $t = 1, 2, \dots$. Consider a policy or forecaster that plays a sequence of arms $\{I_t\}$. We define the *regret* of the policy after n time steps by

$$\rho_n = \max_{i=1,\dots,K} \sum_{t=1}^{n} X_{i,t} - \sum_{t=1}^{n} X_{I_t,t}. \tag{1.1}$$

Both the rewards $X_{i,t}$ and policy actions I_t can be stochastic, and so two notions of regret can be defined, with the expectation taken with respect to the randomness in $X_{i,t}$ and I_t: *Expected regret* $\mathbb{E}\left[\rho_n\right]$ and *pseudo-regret* $\overline{\rho}_n$, given in equations 1.2 and 1.3, respectively. In general, $\overline{\rho}_n \leq \mathbb{E}\left[\rho_n\right]$. That is, the expected regret provides a stronger notion of regret since the expectation is taken with respect to the optimal action observed in the sequence of reward realizations, whereas the pseudo-regret compares the optimal action in expectation [37].

$$\mathbb{E}\left[\rho_n\right] = \mathbb{E}\left[\max_{i=1,\dots,K} \sum_{t=1}^{n} X_{i,t} - \sum_{t=1}^{n} X_{I_t,t}\right]. \tag{1.2}$$

$$\overline{\rho}_n = \max_{i=1,\dots,K} \mathbb{E}\left[\sum_{t=1}^{n} X_{i,t} - \sum_{t=1}^{n} X_{I_t,t}\right]. \tag{1.3}$$

In the most general case, the best possible bound on the expected regret $\mathbb{E}\left[\rho_n\right]$ is $O(\sqrt{n})$, whereas the pseudo-regret $\overline{\rho}_n$ can be bounded by $O(\log n)$. Since it is more natural for a forecaster to compete against the optimal action in expectation, the pseudo-regret $\overline{\rho}_n$ is preferable in comparison to the expected regret $\mathbb{E}\left[\rho_n\right]$ [37].

1.4 Bandit Models

MAB problems can be categorized into three types based on the properties of their reward processes: Stochastic, adversarial and Markovian. We cover the stochastic and adversarial settings in sections 1.4.1 and 1.4.2, respectively, both essentially a summary of the survey work of [37]. We omit discussions on Markovian bandits. In section 1.4.3, we cover other special types of bandits.

1.4.1 Stochastic Bandits

The *stochastic bandit* framework is the most basic and original bandit model. In this model, each arm $i \in \{1, \ldots, K\}$ has an unknown reward density ν_i with mean μ_i. The forecaster draws a reward from arm $I_t \in \{1, \ldots, K\}$ at each time step $t = 1, 2, \ldots$ and receives a reward $X_{I_t,t} \sim \nu_{I_t}$ that is independent of all previous rewards. That is, the reward realizations of any given machine are i.i.d of previous rewards of that machine. Moreover, the rewards are independent across machines. Let $\widehat{\mu}_{i,s}$ denote the sample average reward returned by arm i after it is drawn s times, then $\widehat{\mu}_{i,s} = \frac{1}{s}\sum_{t=1}^{s} X_{i,t}$ since the rewards are i.i.d.

Let $i^* = \underset{i \in \{1,\ldots,K\}}{\operatorname{argmax}}\, \mu_i$ denote the optimal arm, and let $\mu^* = \mu_{i^*}$ and $\nu^* = \nu_{i^*}$ denote the optimal arm's average reward and reward probability density function, respectively. Let $\Delta_i = \mu^* - \mu_i$ denote the average regret of drawing arm i, and let $\Delta = \min_{i \neq i^*} \Delta_i$ be the regret of the second best arm. For simplicity, it is common in the literature to assume that there exists a unique optimal arm so that $\Delta > 0$. Several well-known results become invalidated and new challenges are exposed when multiple optimal arms exist.

If the optimal arm is known, then the problem becomes trivial and the optimal strategy for the forecaster is to always pull the optimal arm. Therefore, it is assumed that the optimal arm is unknown. Hence, the goal of the forecaster is to determine a sequence of arms to draw with objective of minimizing the pseudo-regret solely based on the rewards observed in previous drawings. To that end, let $\mathbb{1}\{\}$ be the indicator function and denote by $T_i(n) = \sum_{t=1}^{n} \mathbb{1}\{I_t = i\}$ the number of times arm i is drawn in the first n time steps, then the pseudo-regret is given by

$$\overline{\rho}_n = \sum_{i=1}^{K} \Delta_i \, \mathbb{E}\left[T_i(n)\right]. \tag{1.4}$$

Due to interaction with an uncertain environment, the best a forecaster can do is to optimize her actions based on a heuristic. A simple yet effective heuristic for stochastic bandit settings is the principle of *optimism in the face of uncertainty.* The principle is generic and applicable to many sequential decision settings. At a high level, the principle devises a solution in three steps: First, at a given time step, a set of possible environments is constructed based on the prior observations of the forecaster. Then, the most likely environment is identified. Finally, the forecaster draws the arm optimal in the environment identified as most likely [37].

The Upper Confidence Bounds (UCB) Method

A well-known method based on the principle of optimism in the face of uncertainty is the *upper confidence bounds* (UCB) method, which was introduced by Lai and Robbins in 1985 in the classical paper [93] to provide an asymptotic analysis of the regret of stochastic bandit problems. In essence, the method computes an upper confidence index for each machine based on all reward realizations obtained from that machine up to that point in time. The index is then used as an estimate for the expected reward of the machine, and the machine with the highest index is chosen by the forecaster at the next time step. All prior realizations are needed in order to compute the index, and so the computation is difficult in general [13]. For MAB problems where the reward models belong to a specific family of distributions, [93] introduced UCB-based policies and proved that the policies suffer logarithmic regret. In particular, they showed that for any suboptimal arm $i \neq i^*$, their policies satisfy

$$\mathbb{E}\left[T_i(n)\right] \leq \left(\frac{1}{D\left(\nu_i||\nu^*\right)} + o(1)\right)\log(n),$$

where $D\left(\nu_i||\nu^*\right) = \int \nu_i \log\left(\frac{\nu_i}{\nu^*}\right)$ is the Kullback-Leibler (KL) divergence between the two densities ν_i and ν^* and $o(1) \to 0$ as $n \to \infty$ [13].

In [37], the authors discuss a UCB-based algorithm called $(\alpha - \psi)$-UCB for the case of (unbounded) reward functions under proper conditions on the moments of the distributions of rewards X. In particular, they assume that there exists a convex function ψ satisfying $\log \mathbb{E}\left[e^{\lambda(X-\mathbb{E}[X])}\right] \leq \psi(\lambda)$ and $\log \mathbb{E}\left[e^{\lambda(\mathbb{E}[X]-X)}\right] \leq \psi(\lambda)$ for all $\lambda \geq 0$. The algorithm draws at time t an arm I_t satisfying

$$I_t \in \underset{i\in\{1,\ldots,K\}}{\operatorname{argmax}} \left[\widehat{\mu}_{i,T_i(t-1)} + \left(\psi^*\right)^{-1}\left(\frac{\alpha\log(t)}{T_i(t-1)}\right)\right],$$

where $\psi^*(\epsilon) = \sup\limits_{\lambda\in\mathbb{R}}\left(\lambda\epsilon - \psi(\lambda)\right)$ is the Legendre-Fenchel transform of ψ and $\alpha >$

0 is a parameter of the algorithm. The authors prove an upper bound on the regret attained by the algorithm for rewards drawn from general distributions, as well as a lower bound for the special case where the rewards are drawn from Bernoulli distributions. A more general lower bound can be found in [93]. The $(\alpha - \psi)$-UCB algorithm generalizes the method outlined in [13] for the case of bounded rewards. The algorithm is referred to as α-UCB for the special case of bounded rewards. The α-UCB algorithm is near-optimal in general, but when Δ_i^2 is significantly smaller than $D(\nu_i || \nu^*)$, the gap between the upper and lower bounds of α-UCB can be significant. The recently-proposed KL-UCB method, independently designed by [63] and [97], addresses this situation. The KL-UCB method strictly dominates the α-UCB method and is optimal for the case of unbounded Bernoulli-distributed rewards [37].

The ϵ-Greedy Method

In [13], Auer *et al.* provided several simple and efficient MAB algorithms that achieve logarithmic regret uniformly over time. One simple algorithm is based on the well-known ϵ-greedy heuristic borrowed from the field of *reinforcement learning* (*c.f.* [135]). In its most basic version, the ϵ-greedy algorithm prescribes to play with probability $1 - \epsilon$ the machine with highest observed average reward, and an arbitrary arm with probability ϵ. A constant exploration rate ϵ will yield linear regret, but decreasing ϵ at a rate of $1/t$ allows for a proof of logarithmic bound on the regret, where t denotes the index of the current play. Figure 1.1 describes the algorithm as it appears in the original text (*c.f.* Figure 3 of [13]).

The following theorem (*c.f.* Theorem 3 of [13]) gives a bound on the regret achieved by the ϵ-greedy method. We emphasize the fact that the theorem gives a bound on the "instantaneous" regret achieved by the algorithm. The proof of the theorem can be found in the original manuscript.

Theorem 1.4.1. *Let $K > 1$. For all reward distributions $\nu_1, \ldots, \nu_K$ with support in $[0, 1]$, suppose that the ϵ-greedy algorithm is run with $0 < d \le \min_{i:\mu_i < \mu^*} \Delta_i$. Then the probability that a suboptimal machine is chosen after any number $n \ge cK/d^2$ of plays is at most*

$$\frac{c}{d^2 n} + 2\left(\frac{c}{d^2}\log\left(\frac{(n-1)d^2 e^{1/2}}{cK}\right)\right)\left(\frac{cK}{(n-1)d^2 e^{1/2}}\right)^{c/(5d^2)} + \frac{4e}{d^2}\left(\frac{cK}{(n-1)d^2 e^{1/2}}\right)^{c/2}.$$

As the authors point out, for $c > 5$, the bound given in theorem 1.4.1 is of order $c/(d^2 n) + o(1/n)$. This is indeed the case since the last two terms are $O(1/n^{\phi+1})$ for some $\phi > 0$.

The Randomized ϵ-greedy Algorithm

Input: Parameters $c > 0$ and $0 < d < 1$. **Initialization:** For $t = 1, 2, \ldots$, define the sequence $\epsilon_t \in (0, 1]$ by

$$\epsilon_t = \min\left\{1, \frac{cK}{d^2 t}\right\}.$$

```
Loop: For each $t = 1, 2, \ldots$
{
    1. Let $\widehat{\mu}_t$ be the machine with the current highest reward.
    2. Play machine $\widehat{\mu}_t$ with probability $1 - \epsilon_t$, and play a random arm with
       probability $\epsilon_t$.
}
```

Figure 1.1: The randomized ϵ-greedy algorithm

Finally, we point out that empirical tests reported in [13] reveal that the regret of the ϵ-greedy algorithm is highly dependent on the choice of parameter c. While a well-tuned algorithm is shown to perform well in most settings, it is not possible to pick a single value of c that works well for all types of problems. Certainly, without proper tuning, the regret can degrade rapidly. Furthermore, in the case of problems with many suboptimal machines with severely varied expected rewards, the uniform exploration of the algorithm can result in large regret [13].

Recent Developments

Stochastic bandits have continued to receive attention in recent years. State of the art algorithms with tightest distribution-free bounds are MOSS (*c.f.* [11, 12]) and improved-UCB (*c.f.* [15, 112]). The long-forgotten approach of Thompson sampling (*c.f.* [139]) has also received much attention lately as a very promising approach, and significant progress has been made in this area. In particular, [4] was the first to prove a logarithmic bound on the regret of Thompson sampling, and [87] showed that the regret is comparable to that of α-UCB. The KL-UCB method mentioned earlier is also recent. These and other recent developments pertaining to stochastic bandits are given in chapter 2 of [37].

1.4.2 Adversarial Bandits

A class of bandit problems that arises in the field of *game theory* is the so-called *adversarial bandit (or game theoretic bandit).* Consider a forecaster faced with the task of playing an unknown game. In particular, consider a casino with slot machines that return a sequence of non-stochastic rewards controlled by some adversary or mechanism. Since the rewards are not i.i.d, this setting is different than that of stochastic bandits. The mechanism controlling the sequence of rewards can behave independently of the forecaster's strategy, in which case the mechanism is called *oblivious.* On the other hand, the mechanism can be reactive to the forecaster's moves and is called *non-oblivious.* We provide a brief summary based on the information found in [37].

The connection between game theory and the bandit problem is nowadays well-known. In game theory, the problem of repeatedly playing a game has been studied by many since the 1950s, including the often cited work of Hannan, Blackwell and Banõs (*c.f.* [20, 71]). In particular, Banõs considered the problem where the game is unknown and the player observes her own payoff but not that of her opponent, which is precisely equivalent to the non-oblivious adversarial bandit problem. In [14], Auer *et al.* re-discovered the same problem and coined the term *non-stochastic multi-armed bandit.* With that, the connection between regret minimization in bandits and equilibria in games was made. The work in [61] is also closely related. More recently, regret minimization for the setting of reactive opponents has been studied in [9, 115].

For adversarial bandits, the performance of the player's strategy against the forecaster after n plays is measured via the same notion of regret $\rho_n = \max_{i=1,\ldots,K} \sum_{t=1}^{n} X_{i,t} - \sum_{t=1}^{n} X_{I_t,t}$ as previously given in equation 1.1. The objective is to obtain regret bounds that are sublinear in the number of plays, but that is impossible to achieve when the forecaster's strategy is deterministic. For instance, consider the following example taken from [37] where an adversary plays the following strategy to force a regret $\rho_n \geq n/2$:

$$\begin{aligned}&\text{For } I_t = 1, \text{ set } X_{2,t} = 1 \text{ and } X_{i,t} = 0\,\forall\, i \neq 2.\\&\text{For } I_t \neq 1, \text{ set } X_{1,t} = 1 \text{ and } X_{i,t} = 0\,\forall\, i \neq 1.\end{aligned}$$

Therefore, it's imperative that a forecaster be randomized in the adversarial bandit setting, and so the regret ρ_n becomes a random variable. Thus, the objective is to bound the regret with high probability with respect to randomization in the forecaster and opponent's strategies. The pseudo-regret $\overline{\rho}_n$ as previously defined in equation 1.3 is easier to bound, and [14, 37] give the so-called *exponential weights for exploration and exploitation* (Exp3) algorithm which achieves $\overline{\rho}_n = O\left(\sqrt{nK \log K}\right)$ for a problem with K arms. Although an improved constant factor is shown for the case when the number of plays n is known to the forecaster, a bound with the same order holds for the *anytime* setting when n is not known to the forecaster, albeit with a larger constant factor.

As for bounds on the regret ρ_n, a variant of the *Exp3* method called *Exp3.P* also appears in [14, 37] which yields with probability $1 - \delta$ a regret $\rho_n \leq \sqrt{\frac{nK}{\log K}} \log \delta^{-1} + 5.15\sqrt{nK \log K}$ for an arbitrary confidence level δ. *Exp3.P* is also shown to have expected regret $\mathbb{E}\left[\rho_n\right] \leq 5.15\sqrt{nK \log K} + \sqrt{\frac{nK}{\log K}}$.

In [14, 37], lower bounds on the pseudo-regret $\overline{\rho}_n$ are provided, and it is shown that the upper bounds outlined above are tight up to a factor $\log K$. This logarithmic gap was tightened in [11] with the introduction of the so-called *implicitly normalized forecaster* (INF) class of strategies which have pseudo-regret of order $\overline{\rho}_n = O\left(\sqrt{nK}\right)$. As for the case of bounding the regret ρ_n with high probability and in expectation, [12] suggests that it is possible to achieve a log-free upper bound in the case of an oblivious adversary, and the authors conjecture that a log free bound is not possible for the case of a non-oblivious adversary. To the best of our knowledge, the logarithmic gap for the non-oblivious case remains an open problem to date.

Finally, with aim of arriving at a flexible strategy that applies to both stochastic and adversarial bandits, the recent work of [38] introduced the *stochastic and adversarial optimal* (SAO) algorithm which guarantees the optimality achieved by both *UCB* and *Exp3.P*. The idea is that if the problem is likely not consistent with i.i.d rewards (based on certain conditions and checks), then the algorithm switches to the *Exp3.P* method under the assumption that the problem is a stochastic bandit problem.

This section is based on chapter 3 of [37]. We refer the reader to the original manuscript for details of the results summarized herein.

1.4.3 Other Bandits

Many other modifications to the aforementioned bandit models appear in the literature. With too many variants to mention, we restrict our discussion to some flavors of interest to our work.

Sleeping Bandits

In many practical settings, not all bandits are available at each stage of the game. For example, consider the classical casino setting. There are usually many gamblers on the casino floor, and so a gambler wishing to pull a particular arm may find that arm occupied by another player. Similarly, as discussed later in section 3.6, a scheduling policy that selects the next action (or arm) based on feedback from a side-bandit model may find that not all actions are available at each iteration of the algorithm.

When some bandits are unavailable at one or more decision steps, the problem becomes the so-called *sleeping bandit*. [92] studies this problem in both the incomplete information setting (called sleeping bandits) and the setting of complete information (called sleeping experts). For the bandit setting, the authors cover both the stochastic and adversarial bandits with objective of minimizing the expected regret. However, since not all actions are available at all times, the formula for expected regret given in equation 1.2 must be modified to only consider actions available at each stage. To that end, consider a pool of K arms, where at time-step $t = \{1, \ldots, n\}$ a subset of arms $A_t \subseteq \{1, \ldots, K\}$ are available to the player. Suppose that selecting action $i \in A_t$ returns a bounded reward $X_{i,t} \in [0, 1]$. Let i_t^* be the best arm amongst those in A_t (per some utility function), then the expected regret after n steps of a sleeping bandit that plays at time t action I_t is given by

$$\mathbb{E}\left[\rho_n\right] = \mathbb{E}\left[\sum_{t=1}^{n} X_{i_t^*,t} - \sum_{t=1}^{n} X_{I_t,t}\right].$$

Suppose that the K possible arms are ordered per their true (yet unknown) mean rewards such that $\mu_1 > \mu_2 > \ldots > \mu_K$. [92] gives a variant of the UCB1 method of [13] for the stochastic bandit having regret bounded from above by $(64 \log n) \sum_{i=1}^{K-1} (\mu_{i+1} - \mu_i)^{-1}$ after n iterations. With a theoretical lower bound on the regret given by $\Omega\left(\log n \sum_{i=1}^{K-1} (\mu_{i+1} - \mu_i)^{-1}\right)$, the provided algorithm is within constant factor of optimality.

Bandits with History

In most of the work that appears in the literature, assumptions are made that the bandit model starts from "scratch" without availability of historical observations. However that is often not the case in practice. For example, when a clinical study is temporarily suspended and is only permitted by a judge to be resumed under new conditions, it is neither efficient nor practical to throw away all experimental results collected prior to the court rulings.

In a novel approach, [124] recently considered stochastic bandits with historical information, and extended three well-known algorithms to take advantage of historical information. The authors show that a logarithmic amount of historical information can reduce the regret by a logarithmic factor. Since the algorithms extended in their work are known to attain logarithmic regret, the result implies that a logarithmic amount of historical data can result in constant regret for the case of finite historical data, and zero regret in case of an infinite amount of historical data. Such result is very powerful. The authors claim that exploiting historical data cannot aid in reducing the regret in an adversarial setting.

This untapped area of study is particularly interesting with ample opportunities for a fresh perspective on a new class of bandits. The trick is to determine how the historical data per arm is incorporated into existing bandit algorithms. For example, as before, let $T_i(t)$ be the number of times arm i has been played in the first t time-steps, and let $\overline{X}_{i,T_i(t)}$ denote the sample average observed for arm i during these plays, then the UCB1 method of [13] plays at time t the arm i that maximizes $\overline{X}_{i,T_i(t-1)} + \sqrt{\frac{2\log t}{T_i(t-1)}}$. On the other hand, the historical version presented by [124], called HUCB1, selects at time t the arm i that maximizes $\overline{X}^h_{i,T_i(t-1)} + \sqrt{\frac{2\log(H_i+t)}{H_i+T_i(t-1)}}$, where H_i is the number of historical samples for arm i and $\overline{X}^h_{i,T_i(t)}$ is the average sample mean including all $H_i + T_i(t)$ historical and "new" observed samples. Using this upper confidence bound, the authors were able to show a tighter upper bound on the expected regret than for the most natural algorithm that plays arm i that maximizes $\overline{X}^h_{i,T_i(t-1)} + \sqrt{2\log\left(\sum_{i=1}^{K} H_i + t\right)(H_i + T_i(t-1))^{-1}}$.

Recall the ϵ-greedy algorithm of [13] previously outlined in figure 1.1. Although one would expect the historical version to replace the rate $\frac{c}{d^2 t}$ with $\frac{c}{d^2(t+H_i)}$, such change does not allow the authors to provide strong theoretic guarantees. The historical version given by [124] is outlined in figure 1.2. Theorem 1.4.2 is the equivalent historical version of theorem 1.4.1.

Theorem 1.4.2. *Let $K > 1$. Given reward distributions $\nu_1, ..., \nu_K$ with support in $[0,1]$, suppose that the HUCB3 algorithm is run with $0 < d \leq \min_{i:\mu_i<\mu^*} \Delta_i$. For $c \geq 10$, the probability that a suboptimal machine j is chosen after any number $n \geq cK/d^2$ of plays is at most $\dfrac{c}{cK\exp\left(H_j d^2/c\right) - cK + d^2 n} + o\left(\dfrac{1}{n}\right)$.*

The Randomized ϵ-greedy Algorithm with Historical Information (HUCB3)

Input: Parameters $c > 0$ and $0 < d < 1$. **Initialization:** For $t = 1, 2, ...,$ define the sequence $\epsilon_t \in (0, 1]$ by

$$\epsilon_t^i = \begin{cases} 1/K, & t \leq cK/d^2; \\ \left(K\left(\exp\left(H_j d^2/c\right)\right) - 1) + d^2 t/c\right)^{-1}, & t > cK/d^2. \end{cases}$$

Loop: For each $t = 1, 2, ...$

```
{
    1. Let $\widehat{\mu}_t$ be the machine with the current highest reward $\overline{X}^h_{\mu,T_\mu(t-1)}$.
    2. Play machine $\widehat{\mu}_t$ with probability $1 - \sum_{i=1}^{K} \epsilon_t^i$, and play each arm $j$ with
       probability $\epsilon_t^j$.
}
```

Figure 1.2: The randomized ϵ-greedy algorithm with historical information (HUCB3)

An open problem posed by [124], based on the authors' choice to incorporate the historical data into the upper confidence bound in the form of $\log\left(t + \sum_{i=1}^{K} H_i\right)$ as opposed to $\log(t + H_i)$, is whether or not it is possible for HUCB1 to be improved such that the per-arm historical data can be better exploited.

Other Bandits

Several other bandit models appear in the literature. It is not unreasonable in many practical setting to pay some cost associated with switching between bandit arms. For example, consider the scenario where a print shop has a machine that takes a single dye. Orders arrive for various products per some arrival distribution that is unknown to the machine operator. The objective is to decide at any time whether or not to continue to make products using the current dye installed in the machine. Switching to a different dye is associated with downtime which can be thought of as a switching cost. This is an example of the so-called *multi-armed bandit with switching costs*. [2, 3] study this subject and provide algorithms that are efficient in the asymptotic sense for both stochastic and Markovian rewards. The main idea behind their approach is to group the samples collected for each bandit arm into blocks of increasing size in such a way that the regret associated with switching costs is marginal. In particular, it is possible to sample suboptimal arms at most $O(\log n)$ times during the first n iterations of the algorithm, which in turn bounds the number of arm switches to $o(\log n)$. This results in a logarithmic bound on the regret (asymptotically) up to constant factor, which is the same as that for bandits without switching costs. A necessary condition is for the switching costs to either be fixed or bounded, and a fundamental assumption is made such that there exists a unique optimal arm. If the latter does not hold, then an arbitrarily large amount of switching can occur between the optimal arms.

Another variation on the classical stochastic multi-armed bandit is the so-called *restless bandit*. While in the original MAB only the chosen arm evolves at each stage of the algorithm, in the restless bandit arms that are not played can also evolve at each iteration. This introduces additional complexity that needs to be dealt with. Although not generally the case, an UCB-based algorithm results in logarithmic regret under the right conditions (*c.f.* [110, 137]).

Yet another variation is the *max k-armed bandit*. In contrast with the original bandit problem which aims at minimizing the regret in a cumulative sense, the max k-armed bandit aims at maximizing the maximum payoff. While in the classical stochastic MAB an efficient algorithm plays the optimal arm exponentially more frequently than the other arms, in the max k-armed bandit the optimal arm must be played at a rate that increases doubly exponentially relative to the other (suboptimal) arms. [134] gives a simple algorithm under simplified assumptions. The max k-armed bandit problem was first introduced by [52].

1.5 Open Problems

The study of multi-armed bandits in its various flavors continues to be an interesting and active area of study. We herein highlight some open problems for future work.

1) When the objective is to bound the regret ρ_n with high probability or in expectation, [12] suggests that it's possible to achieve a log-free upper bound in the case of an oblivious adversary, and it is conjectured that a log-free bound is not possible for the case of a non-oblivious adversary. To the best of our knowledge, this remains an open problem to date.

2) As [92] points out, although theoretical guarantees are given for the al-

gorithms that they present for adversarial bandits, these algorithms are computationally inefficient. The authors pose as an open problem the question of whether or not efficient algorithms exist for this setting which can achieve regret that grows polynomially in the number of bandit arms.

3) Following the ideas of [124], taking advantage of historical data from previous plays of a bandit can be used to significantly reduce the MAB regret. This opens a direction for new work where historical data can be incorporated into various types of bandits to achieve new generalized algorithms.

4) The long-forgotten area of Thompson sampling is yet another promising approach for minimizing bandit regret. This approach has picked up momentum in recent years.

Chapter 2

An ϵ-Greedy Multi-Armed Bandit Approach to Markov Decision Processes

In this chapter we present REGA, a novel adaptive sampling-based algorithm for control of finite-horizon Markov decision processes (MDPs) with very large state spaces and small action spaces. We apply a variant of the ϵ-greedy multi-armed bandit algorithm to each stage of the MDP in a recursive manner, thus computing an estimation of the "reward-to-go" value at each stage of the MDP. We provide a finite-time analysis of REGA. In particular, we provide a bound on the probability that the approximation error exceeds a given threshold, where the bound is given in terms of the number of samples collected at each stage of the MDP. We empirically compare REGA against two other sampling-based algorithms by running simulations against the *SysAdmin* benchmark problem with 2^{10} states. The results show that REGA is a competitive algorithm. Moreover, REGA is found to empirically outperform an implementation of the algorithm that uses the "original" ϵ-greedy algorithm that commonly appears in the literature.

2.1 Introduction to (PO)MDPs

Markov decision processes (MDPs), named after Andrey Markov, provide a mathematical framework for modeling decision-making in situations where outcomes associated with an agent's actions are controlled stochastically. An MDP can be fully described by the tuple (S, A, T, R, γ): S is the set of system states. We use the notation $s_t \in S$ to denote the state of the system at time t. A is the set of available actions, with $A_s \subseteq A$ being the subset actions available to the agent from each state $s \in S$. $p(s'|s,a) = T(s_{t+1} = s'|s_t = s, a_t = a)$ captures the transition model, *i. e.*, the probability of arriving at state s' at time $(t+1)$ if action a is taken at state s at time t. Finally, R denotes the set of rewards, with $r_a(s,s') \in R$ being the immediate (expected) reward when action a is taken at state s, thus resulting in transition to state s' with probability $p(s'|s,a)$. $\gamma \in [0,1]$ is a discount factor that is optional for the finite horizon setting but necessary for the infinite horizon setting.

In an MDP, at each decision-making moment, taking action a at state s advances the system to state s' with probability $p(s'|s,a)$, and provides the agent with an immediate reward $r_a(s,s')$. Let π be a policy that maps states to actions, then a value function V is used to measure the quality of π over the (possibly infinite) planning horizon H. Let $\pi(s_t)$ denote the action taken at time $t = 0, 1, \ldots, H-1$ when the system is at state s_t, then starting at an initial state s_0, the value function is given by

$$V^{\pi}(s_0) = E\left[\sum_{t=0}^{H-1} \gamma^t r_{\pi(s_t)}(s_t, s_{t+1})\right].$$

We consider the objective of determining an optimal policy, π^*, that maximizes total reward over the planning horizon. In the literature, there are three well-known algorithms that solve for an optimal policy: *Value iteration*, *policy iteration*, and *linear programming*.

The usual way to solve for an optimal policy of an MDP is via the *Bellman equation*, also known as the *dynamic programming equation*. Chapter III.3 of [21] describes *Bellman's principle of optimality* as follows:

> **Principle of optimality**: An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

To that end, using the notation of [114], the *value iteration* algorithm computes the optimal policy via *Bellman's recursion*, as follows:

$$\pi^*(s) = \arg\max_{a \in A} Q^*(s,a),$$

where

$$V_h(s) = r_a(s) + \max_{a \in A} Q_h(s,a)$$

and

$$Q_h(s,a) = \gamma \sum_{s' \in S} p(s'|s,a) V_{h-1}(s').$$

Here, any quality with an asterisk (*i. e.*, π^*, V^*, Q^*, *etc.*) denotes the optimal value of the function. The subscript in V_h and Q_h denotes the remaining "to-go" value of the functions at stage h of the MDP. The special case of $h = 0$ denotes the value of the function at the initial state. When the context is clear, we drop the subscript and use $V(s)$ instead of $V_0(s_0)$.

A fundamental assumption in the MDP framework is that the state of the system is known at all times. However, in many cases, including the e-Discovery scheduling problem (which we will cover in chapter 3), the agent can only make a partial observation of the state of the system. The *partially observable (PO)*MDP framework (*c.f.* [10, 57]) handles this case by introducing the notion of a *belief*, which is a probability distribution over the state space. A POMDP is fully defined by the tuple $(S, A, O, T, \Omega, R, \gamma)$, with S, A, T, R, γ all as in the MDP setting. The two additional components are a set of observations O, and a probability distribution $\Omega(o|s)$ that gives the probability of observing $o \in O$ when the system is in state $s \in S$.

The belief is updated as the agent interacts with the system. Following the notation in [114], let b be the current belief of the agent, if taking action a results in observation o, the updated belief about the new state s' is given by

$$b^{a,o}(s') = \frac{p(o|s')}{p(o|b,a)} p(s'|b,a),$$

where $p(s'|b,a)$ is given by the following formula (replace the integral with a sum for the discrete case)

$$p(s'|b,a) = \int_{s \in S} p(s'|s,a) b(s) ds.$$

For a POMDP, the *Bellman recursion* is given by

$$V_h(b) = \sup_{a \in A} \{ < r_a, b > + \gamma \int_o p(o|b,a) V_{h-1}(b^{a,o}) do \},$$

where $< r_a, b >$ is defined per the following formula (replace the integral with a sum for the discrete case)

$$< f, b > = \int_{s \in S} f(s) b(s) ds.$$

It is worth noting that any belief-based POMDP can be converted to a continuous-state MDP, with one dimension per state. In the case of a continuous-state POMDP, the resulting MDP has an infinite-dimensional continuous state space. This follows from the fact that the belief contains sufficient information for optimal planning [29].

As far as solving POMDPs for optimal policies is concerned, the vast majority of the literature covers the discrete case. Some papers on value iteration algorithms include [40, 49, 85, 107, 113, 129, 149]. [130] introduced the `PERSEUS` algorithm, which is a point-based value-iteration algorithm that has been shown to be efficient for discrete POMDPs.

More recently, the case of continuous POMDPs has been considered. [114] showed that the value function for arbitrary continuous POMDPs is in general convex, and piecewise-linear and convex in the special case where the state space is continuous but the action and observation spaces are discrete. Moreover, the paper extends the `PERSEUS` algorithm to cover the cases of continuous state space (*c.f.* [114], section 5), as well as the case of continuous action and observation spaces (*c.f.* [114], section 6). We refer the reader to the original paper for the details of the extended sampling approach.

In the aforementioned formulation of MDPs and POMDPs, the system dynamics are stochastic. However, when the goal is to simulate an MDP or POMDP, another equivalent definition is more useful to capture the model. Consider a finite horizon MDP with non-negative rewards. As before, let A and S be finite action and state spaces, respectively, and let H denote the horizon of the MDP. For any stage $t = 0, \dots, H-1$ and state $s_t \in S$, choosing a control $a_t \in A$ will (stochastically) lead to a future state s_{t+1}. The system dynamics can be captured by $s_{t+1} = f(s_t, a_t, w_t)$, where $w_t \sim U(0,1)$ is a uniform random variable representing the uncertainly of the system (*c.f.* [72, 116]). Let $R(s_t, a_t, w_t)$ denote the immediate reward, and let Π be the set of all possible deterministic non-stationary Markovian policies $\pi = \{\pi_t | \pi_t : S \to A, t = 0, \dots, H-1\}$. To simplify our notation, let $w = (w_t, w_{t+1}, \dots, w_{H-1})$. Given a policy $\pi \in \Pi$, the reward-to-go value function is given by $V_t^\pi(s_t) = E_w^\pi\left[\sum_{i=t}^{H-1} R\left(s_i, \pi_i(s_i), w_i\right)\right]$, with $V_H^\pi(s) = 0$, $\forall\, s \in S$. Let the optimal reward-to-go value function be denoted by $V_t^*(s) = \max\limits_{\pi \in \Pi} V_t^\pi(s)$, with $V_H^*(s) = 0$, $\forall\, s \in S$, then $V_t^*(s)$ can be computed recursively using

$$V_t^*(s) = \max_{a \in A} Q_t^*(s,a)$$
$$Q_t^*(s,a) = E_w^\pi\left[R(s,a,w) + V_{t+1}^*(f(s,a,w))\right].$$

When clear from the context, we omit the subscript for the initial horizon and state, and use $V^*(s)$ instead of $V_0^*(s_0)$.

2.2 Related Work

Recent years have seen theoretical progress in the area of simulation-based methods for solving large finite-horizon MDPs. Two algorithms closely related to the method we propose in this chapter are the *recursive automata sampling algorithm* (RASA) of [43] and the *adaptive multi-stage sampling algorithm* (AMS) of [44]. RASA recursively applies a *learning automata* algorithm at each stage of the MDP in order to calculate the reward-to-go value function. When applied to the initial state of the MDP, the algorithm produces an approximated value for the MDP. The objective is to maximize the total reward criterion, but the algorithm and results can be trivially restated for minimization problems. In [43], the authors give two probabilistic bounds on the performance of RASA as a function of the number of samples at each stage. In particular, a lower bound on the probability that the algorithm will sample the optimal action, and an upper bound on the probability that the error of the algorithm exceeds a given threshold.

The AMS algorithm is very similar to RASA, except that learning automata is replaced with a UCB-based multi-armed bandit algorithm to drive the simulation at each stage of the MDP. In [44], the authors showed that the MDP value function estimator produced by AMS is asymptotically unbiased. This asymptotic convergence result is weaker than the convergence in probability result shown for RASA. The value function estimator required to show the theoretical results for AMS is based on weighted averaging of the Q-function estimates of each sampled action, where the weights are taken based on the number of times each action is chosen for simulation by the underlying MAB algorithm. Thus, the resultant value function estimator is biased below the true value function of the MDP. Other simulation-based approaches for solving MDPs include stochastic annealing [76], neuro-dynamic programming [27], stochastic neural networks [121] and Q-learning [146, 147].

In this section, we propose a novel algorithm called the *recursive ϵ-greedy algorithm* (REGA), that combines ideas from RASA and AMS. Like AMS, the sampling strategy at each stage of the MDP is based on a multi-armed bandit model. In particular, we use a generalized variant of the ϵ-greedy algorithm by Auer *et al.* (*c.f.* [13]). On the other hand, our theoretical analysis and the value function estimator that we use are similar to those of RASA. We give a finite-time analysis of REGA following the same ideas found in [43]. In particular, we give an upper bound on the probability that the approximated reward-to-go generated by REGA has an absolute error larger than a tolerated threshold. We provide a convergence rate in terms of the number of samples captured at each stage of the MDP. Much credit goes to [43] for the ideas behind our theoretical analysis.

The rest of the chapter is organized as follows: Section 2.2.1 covers the adaptive sampling framework and the AMS algorithm. Our REGA algorithm is introduced and finite-time analysis is provided in section 2.3. A similar analysis of a variant that we call OREGA is covered in section 2.4. Section 2.5 discusses some practical limitations and potential workarounds. Empirical results in comparison to the RASA and AMS algorithms appear in section 2.6. A discussion pertaining to Auer's ϵ-greedy method and the attained regret for the traditional stochastic bandit problem is given in section 2.7. We conclude with some remarks in section 2.8.

2.2.1 Adaptive Sampling MDP Framework and AMS

[75] describes a simulation-based framework for approximately solving general finite horizon MDPs with large state spaces. In this section, we outline the proposed framework and main results from chapter 3 of [75]. All theorems are listed without proof. We refer the reader that is interested in the proofs to the original manuscript.

Consider a MDP with large state space and finite horizon H. This simulation-based framework treats the MDP as a decision tree with depth H, where each node in the tree represents a state of the MDP, the root note represents the initial state, and the leaf nodes represent the state of the MDP at horizon $H-1$. Via a depth-first search, the algorithm generates sample trajectories from the root of the tree to the leaf nodes. Based on the sample averages of the Q-functions, backwards induction is used to estimate the value function at previous stages further up the tree. The calculated value at the root node represents the estimate of the value function for the entire MDP.

Consider a single stage $i = 0, 1, \dots, H-1$ and current state $s \in S$. Let $\hat{Q}_{i,M_i}(s,a)$ and $\hat{V}_{i,M_i}(s) = \operatorname*{argmax}_{a \in A(s)} \hat{Q}_{i,M_i}(s,a)$ denote estimates of $Q_i^*(s,a)$ and $V_i^*(s)$, respectively, when M_i simulations are allocated to stage i. When clear from the context, we drop the suffix M_i and use the simplified notation $\hat{Q}_i(s,a)$ and $\hat{V}_i(s)$. By convention, we set $\hat{V}_H(s) = 0$. Let $M_a^i(s)$ denote the number of times action a is sampled at stage i. It always holds that $M_i = \sum_{a \in A(s)} M_a^i(s)$. The objective is to estimate $V^*(s) = V_0^*(s)$, the optimal value of the MDP at the initial state.

Figure 2.1 outlines the basic implementation of the adaptive multi-stage sampling algorithm. We point out that in step 4 of the loop, the algorithm calls itself recursively and that the estimates for the Q-functions, $\hat{Q}_{i,M_i}(s,\hat{a})$, are updated via backwards induction. Notice also that the algorithm does not sample every state of the MDP. In particular, the only states sampled are those reached by the next-state functions $f(s, \hat{a}, w)$. It thus follows that the runtime complexity of the algorithm is in general independent of the (possibly large) size of the state space.

```
General Adaptive Multi-Stage Sampling Algorithm

Input: Stage $i < H$, state $s \in S$, $M_i > 0$, algorithm $\Upsilon$ to determine the next action to sample, parameters specific to algorithm $\Upsilon$.

Initialization:
    1. Max horizon stopping rule: Set $\hat{V}_{H,M_H}(s) = 0$.
    2. Perform any $\Upsilon$-specific initializations.

Loop ($N_i$ times):
{
    1. Determine next action $\hat{a}$ to play via algorithm $\Upsilon$.
    2. Simulate the next state via $f(s, \hat{a}, w)$, where $w \sim U(0, 1)$.
    3. Increment the number of times $\hat{a}$ has been sampled: $M^i_{\hat{a}}(s) = M^i_{\hat{a}}(s) + 1$.
    4. Backwards induction and recursion: Update $\hat{Q}_{i,M_i}(s, \hat{a})$ based on $\acute{R}(s, \hat{a}, w)$ and $\hat{V}_{i+1,M_{i+1}}(f(s, \hat{a}, w))$.
    5. Update any parameters specific to algorithm $\Upsilon$.
}

Output: $\hat{V}_{i,M_i}(s)$ based on the estimates of Q-functions $\{\hat{Q}_{i,M_i}(s, a)\}$.
```

Figure 2.1: Outline of the adaptive multi-stage sampling algorithm

The adaptive multi-stage sampling algorithm determines the next action to sample via some generic mechanism Υ. We note that this mechanism is intentionally not specified. In related work, [89] uses a similar recursive tree sampling structure to create an on-line near-optimal planning algorithm for solving large MDPs. However, in their work, each action is sampled a fix number of times. Under tight computational and time budgets, it is imperative to sample actions in a more strategic fashion. In particular, it is desirable to sample more frequently the actions that either have high reward variability or are more likely to produce higher rewards [75]. To achieve this goal, [75] uses a MAB-based algorithm for Υ in order to adaptively choose the next action to sample based on the observed variability of the rewards from previous iterations of the simulation.

The intuition behind the sampling-based algorithm is as follows: Consider a single stage of the finite horizon MDP. For an arbitrary current state s, redefine the regret to be the difference between the true optimal value at s and the approximate value yielded by the sampling process. A MAB algorithm is used to minimize the regret for the current state, which equivalently minimizes the error of approximating the optimal value of the state. The one-stage approximations are then combined in a recursive manner in order to provide a multi-stage approximate solution to the entire MDP.

For each one-stage problem, the MAB algorithm chosen by [75] is the UCB1 algorithm of [13]. They call the resultant method the adaptive multi-stage sampling algorithm (AMS), which overloads the term and can thus cause ambiguity between the UCB-based AMS and the generic AMS algorithm outline in figure 2.1. In the remainder of this chapter, we use AMS to denote the UBC-based AMS algorithm.

It is well-known that efficient MAB algorithms play the optimal arm exponentially more frequently than other arms [93]. As such, AMS estimates the value of the MDP by weighing each $\hat{Q}(s,a)$ using the number of times action a is sampled by the algorithm. That is, $\hat{V}_{i,M_i}(s) = \sum_{a \in A(s)} \frac{M_a^i(s)}{M_i} \hat{Q}_{i,M_i}(s,a)$. Figure 2.2 outlines the AMS algorithm.

The runtime complexity of AMS is $O\left(\left(|A| \max_{i=0,\ldots,H-1} M_i\right)^H\right)$. Most notable is that the bound is independent of the (possibly large) size of the state space of the MDP, but is dependent on the size of the action space $|A|$ due to the requirement that each action be sampled at least once at each state sampled by the algorithm, as well as the update step for the current optimal action $\hat{a}$.

In [75], it is shown that AMS achieves an approximate solution for the MDP that is asymptotically unbiased, and that the bias is bounded by a quantity that converges to zero at a rate of $O\left(\sum_{i=0}^{H-1} \frac{\log M_i}{M_i}\right)$. In particular, $\forall s \in S$ and $a \in A(s)$, let $\theta(s) = \min_{a \neq a^*}(V^*(s) - Q^*(s,a)) > 0$ denote the difference between the largest and second largest (strictly suboptimal) expected bandit rewards, then the main result of [75] is given in theorem 2.2.1.

Theorem 2.2.1. *Suppose there exists a constant $c > 0$ such that $\inf_{s \in S} \theta(s) \geq c$,*

```
The UCB-Based Adaptive Multi-Stage Sampling Algorithm (AMS)

Input: Stage $i < H$, state $s \in S$ and $M_i > \max_{s \in S} |A(s)|$.

Initialization:

    1. Max horizon stopping rule: $\hat{V}_{H,M_H}(s) = 0$.
    2. Simulate the next state $f(s, a, w_1^a)$ for each action $a \in A(s)$, where
       $\{w_j^a\} \sim U(0,1)$ is the random number sequence for action $a$.
    3. Set the overall number of samples so far: $\bar{m} = |A(s)|$.
    4. Set the number of times each action $a \in A(s)$ has been sampled so far:
       $M_a^i(s) = 1$.
    5. Set $\hat{Q}_{i,M_i}(s,a) = \acute{R}(s, a, w_1^a) + \gamma \hat{V}_{i+1,M_{i+1}}(f(s, a, w_1^a)), \quad \forall a \in A(s)$.

Loop (until $\bar{m} = M_i$):
{
    1. Let $\hat{a}$ be the current estimate of the optimal action. That is,

       $$\hat{a} = \underset{a \in A(s)}{\operatorname{argmax}} \left( \hat{Q}_{i,M_i}(s,a) + R_{\max} \sqrt{\frac{2 \log \bar{m}}{M_a^i(s)}} \right),$$

       where

       $$\hat{Q}_{i,M_i}(s,a) = \frac{1}{M_a^i(s)} \sum_{j=1}^{M_a^i(s)} \left[ \acute{R}(s, a, w_j^a) + \gamma \hat{V}_{i+1,M_{i+1}}(f(s, a, w_j^a)) \right]. \quad (2.1)$$

    2. Update $M_{\hat{a}}^i(s) = M_{\hat{a}}^i(s) + 1$. Generate $w_{M_{\hat{a}}^i(s)}^{\hat{a}} \sim U(0,1)$ and update
       $\hat{Q}_{i,M_i}(s, \hat{a})$ per 2.1 using simulated next state $f\left(s, \hat{a}, w_{M_{\hat{a}}^i(s)}^{\hat{a}}\right)$. Update
       $\bar{m} = \bar{m} + 1$.
}

Output:

    $$\hat{V}_{i,M_i}(s) = \sum_{a \in A(s)} \frac{M_a^i(s)}{M_i} \hat{Q}_{i,M_i}(s,a).$$
```

Figure 2.2: The UCB-based adaptive multi-stage sampling algorithm (AMS)

and suppose that $|A(s)| > 1$ *for all* $s \in S$. *If AMS is run with input* M_i *at each stage* $i = 0, ..., H-1$ *starting at an arbitrary state* $s \in S$, *then it holds that* $\lim_{M_0\to\infty} \lim_{M_1\to\infty} \dots \lim_{M_{H-1}\to\infty} E\left[\hat{V}_{0,M_0}(s)\right] = V_0^*(s)$. *Furthermore, the bias induced by the algorithm is bounded by a quantity that converges to zero. In particular,* $V_0^*(s) - E\left[\hat{V}_{0,M_0}(s_0)\right] \leq O\left(\sum_{i=0}^{H-1} \frac{\log M_i}{M_i}\right)$.

Note that in theorem 2.2.1, the strictly non-negative lower bound assumption on $\theta(s)$ trivially holds when the state space of the MDP is finite.

We conclude by noting that the results outlined above are based on the rather conservative weighted estimator $\hat{V}(s)$. Since the bias of this estimator is defined in terms of the MAB regret, and it is well-known that the regret must be logarithmic at best, the bound provided above is the tightest possible. In order to achieve tighter bounds on alternative estimators, it is necessary that the bias of such alternative estimators not be defined in terms of the MAB regret. The REGA algorithm discussed later in this chapter avoids defining the value function estimator in terms of the MAB regret.

Empirical results outlined in [75] compare this estimator to two alternative estimators, which we refer to here as AMS1 and AMS2. AMS1 replaces the original estimator with an alternative estimator based on the best observed Q-function. On the other hand, AMS2 replaces the original estimator with an alternative estimator based on the action most sampled.

$$\text{AMS1 estimator: } \hat{V}_{i,M_i}(s) = \max_{a\in A(s)} \hat{Q}_{i,M_i}(s,a). \tag{2.2}$$

$$\text{AMS2 estimator: } \hat{V}_{i,M_i}(s) = \hat{Q}_{i,M_i}(s,\breve{a}), \text{ where } \breve{a} = \operatorname*{argmax}_{a\in A(s)} M_a^i(s). \tag{2.3}$$

AMS1 was found to outperform AMS empirically, but determining the convergence rates for value function estimators in the form of AMS1 and AMS2 is noted as difficult to obtain and is left as an open problem. Moreover, the quest for an algorithm with provable convergence in a stronger sense than asymptotic unbiasedness is also posed as an open problem. The REGA method that we proposed in this chapter addresses these problems.

2.3 The REGA method

In this section, we introduce the *recursive* ϵ*-greedy algorithm* (REGA). The algorithm is motivated by the UCB-based AMS algorithm discussed in section

2.2.1. We provide a finite-time analysis for REGA, which is stronger than the asymptotic unbiasedness results obtained for AMS in [75].

Consider a single stage $i = 0, 1, \ldots, H-1$ and current state $s \in S$, and let $c_i \geq 1$ be tunable parameters that control the amount of pure exploration at each stage i of the algorithm. Moreover, for a non-negative integer m and a state s, let $\{\epsilon_m^s\}_m$ be the non-increasing sequence given by

$$\epsilon_m^s = \min\left\{1, \frac{c_i|A(s)|}{\sqrt{m}}\right\}, \tag{2.4}$$

where $A(s)$ is the set of available actions at state s, and $|A(s)|$ is the size of $A(s)$. When the state s is clear from the context, we drop the superscript and use the simplified notation ϵ_m instead of ϵ_m^s. The REGA algorithm is outlined in figure 2.3.

We note that the widely-cited ϵ-greedy method of [13] defines ϵ_m^s differently where the denominator $\sqrt{m}$ in equation 2.4 is replaced with m. See section 2.4 for a theoretical analysis using the original definition of ϵ_m^s and section 2.6 for a comparison of empirical results of the two variants. We show in section 2.7 that this change is not desirable for stochastic MABs.

The REGA algorithm treats each action available at the current state as an arm in a MAB setting. As such, the next arm to sample is chosen in accordance with the ϵ-greedy MAB method. Let m be the number of samples drawn at the current stage thus far. At first, all arms are uniformly sampled up to a certain total number of purely exploratory/training simulations. After that, with ties broken arbitrarily, the arm believed to be optimal is sampled with increasing probability $1 - \epsilon_m$. For each simulation and chosen action, the MDP advances stochastically to a future stage s_{i+1} according to the system dynamics $s_{i+1} = f(s_i, a_i, w_i)$ (*c.f.* [72, 116]). The next state s_{i+1} is once again treated as a MAB. We continue to apply the ϵ-greedy method recursively until the maximum horizon H is reached. Via backwards induction, the optimal reward-to-go at each stage is estimated, and when applied to the initial state of the MDP, we obtain an estimation of the optimal value function of the MDP. Similar to AMS, the main advantage of REGA is that the reward-to-go need not be computed for every state of the MDP, as the algorithm only considers the states visited during sampling. As such, REGA is well-suited for MDPs with very large state spaces.

```
**Recursive $\epsilon$-Greedy Algorithm (REGA)**
Input: Stage $i < H$, state $s \in S$, $c_i \geq 1$, and $M_i \geq 1$.
Initialization: $\hat{V}_H(s) = 0$, $\hat{Q}_i(s,a) = 0$ and $M_a^i(s) = 0 \quad \forall\, a \in A(s)$.
Loop (For $m = 1, 2, ..., M_i$):
{
    1. Update the current optimal action $\hat{a}_i$.
         $\hat{a}_i = \operatorname*{argmax}_{a \in A(s)} \hat{Q}_i(s,a)$
    2. Pick the next action to sample
         $\bar{a}_i = \begin{cases} \hat{a}_i, & w.p.\ (1-\epsilon_m) \\ \text{(uniformly) random arm}, & w.p.\ \epsilon_m \end{cases}$
    3. Generate $w_m^i \sim U(0,1)$ then simulate the next state $f(s, \bar{a}_i, w_m^i)$ and
       immediate reward $R(s, \bar{a}_i, w_m^i)$.
    4. Update $\hat{Q}_i(s, \bar{a}_i) =$
         $\frac{M_{\bar{a}_i}^i(s)\hat{Q}_i(s,\bar{a}_i) + \hat{V}_{i+1}(f(s,\bar{a}_i,w_m^i)) + R(s,\bar{a}_i,w_m^i)}{M_{\bar{a}_i}^i(s)+1}.$     (2.5)
    5. Update $M_{\bar{a}_i}^i(s) = M_{\bar{a}_i}^i(s) + 1$.
}
Output: $\hat{V}_i(s) = \operatorname*{argmax}_{a \in A} \hat{Q}_i(s)$.
```
Figure 2.3: The recursive ϵ-greedy algorithm (REGA)

We now provide a theoretical treatment of the convergence properties of REGA. Before introducing the main result, the following lemma is needed to justify some algebraic steps.

Lemma 2.3.1. *Let $c \geq 1$ be a fixed real number, and let $M \geq 1$ and $r >$*

$(c^2+c)^2$ *be fixed integers. Then the following inequalities holds:*

$$\frac{cM}{\sqrt{r+M+1}} < \sum_{i=r+1}^{r+M} -\log\left(1-\frac{c}{\sqrt{i}}\right) < \frac{(c+1)M}{\sqrt{r}} \tag{2.6}$$

$$\sum_{i=r+1}^{r+M} \log^2(1-\frac{c}{\sqrt{i}}) < (c+1)^2 \log\left(1+\frac{M}{r}\right). \tag{2.7}$$

Proof. Let $x > (c^2+c)^2$ and consider the function $b(x) = -\log(1-c/\sqrt{x})$. We have $\lim_{x\to\infty} b(x) = 0$ and $b'(x) < 0$. Therefore, b is positive and decreasing in x, and so the definite integral of b can be bounded from below by the right-hand Riemann sum, and from above by the left-hand Riemann sum, yielding

$$0 < \int_{r+1}^{r+M+1} b(x)dx \le \sum_{i=r+1}^{r+M} b(i) \le \int_{r}^{r+M} b(x)dx. \tag{2.8}$$

To prove the first part, we lower bound the function b. Recall that the identity $\log(1-y) < -y$ holds for $y \in (0,1)$. Since $c/\sqrt{x} \in (0,1)$, we obtain

$$b(x) > \frac{c}{\sqrt{x}}. \tag{2.9}$$

Combining equations 2.8 and 2.9 yields

$$\begin{aligned}\sum_{i=r+1}^{r+M} b(i) \ge \int_{r+1}^{r+M+1} b(x)dx &> \int_{r+1}^{r+M+1} \frac{c}{\sqrt{x}}dx \\ &= 2c\left(\sqrt{r+M+1}-\sqrt{r+1}\right) \\ &= \frac{2cM}{\sqrt{r+M+1}+\sqrt{r+1}} \\ &> \frac{cM}{\sqrt{r+M+1}},\end{aligned}$$

which proves the first part. We now turn our attention to the second part of our claim. For brevity, denote $u = \frac{(c^2+c)(\sqrt{x}-c)}{\sqrt{x}-c^2-c}$. It can be seen via basic algebraic steps that

$$\begin{aligned}
-\log(1-\frac{c}{\sqrt{x}}) &= \log\left(\frac{(u-c)(c+1)}{cu}\right) \\
&= \log(1-\frac{c}{u}) - \log(1-\frac{c}{c^2+c}) = \int_{c^2+c}^{u} \frac{c}{t(t-c)}dt. \\
&\leq \int_{c^2+c}^{u} \frac{c+1}{t^2}dt = \frac{c}{\sqrt{x}-c} < \frac{c+1}{\sqrt{x}}. \qquad (2.10)
\end{aligned}$$

Combining equations 2.8 and 2.10 yields

$$\begin{aligned}
\sum_{i=r+1}^{r+M} b(i) \leq \int_{r}^{r+M} b(x)dx &\leq \int_{r}^{r+M} \frac{c+1}{\sqrt{x}}dx \\
&= 2(c+1)\left(\sqrt{r+M}-\sqrt{r}\right) \\
&= \frac{2(c+1)M}{\sqrt{r+M}+\sqrt{r}} \\
&< \frac{(c+1)M}{\sqrt{r}}.
\end{aligned}$$

This completes the proof of equation 2.6. We now prove equation 2.7. Squaring both sides yields of equation 2.10 yields

$$\log^2(1-\frac{c}{\sqrt{x}}) < \frac{(c+1)^2}{x}.$$

Therefore,

$$\sum_{i=r+1}^{r+M} \log^2(1-\frac{c}{\sqrt{i}}) < \sum_{i=r+1}^{r+M} \frac{(c+1)^2}{i}. \qquad (2.11)$$

But $1/x$ is positive and decreasing for $x > 0$, therefore its definite integral can be bound from below by the function's righthand Riemann sum. Hence,

$$\sum_{i=r+1}^{r+M} \frac{(c+1)^2}{i} \leq \int_{r}^{r+M} \frac{(c+1)^2}{x}dx = (c+1)^2 \log\left(1+\frac{M}{r}\right),$$

which completes the proof. $\square$

Recall that the discrete probability distribution of the sum of independent Bernoulli trails that are not identically distributed is known as the *Poisson binomial distribution.* The following theorem gives a bound on the error of estimating the Poisson binomial distribution CDF using the "standard" Poisson distribution CDF (*c.f.* [123], theorem 5.1).

Theorem 2.3.2. *Let $X_1, ..., X_n$ be independent Bernoulli variables with respective success probabilities $p_1, ..., p_n$. Assume that $0 < p_i < 1$ and let $\lambda_i = -\log(1-p_i)$ for all $1 \leq i \leq n$. Let $Y(\lambda)$ denote a random variable having the Poisson distribution with mean parameter λ. Then for each integer $m \geq 1$ we have*

$$P\left[Y\left(\sum_{i=1}^{n}\lambda_i\right) \leq m\right] \leq P\left[\sum_{i=1}^{n} X_i \leq m\right] \leq P\left[Y\left(\sum_{i=1}^{n}\lambda_i\right) \leq m\right] + \frac{1}{2}\sum_{i=1}^{n}\lambda_i^2.$$

The following result appears in [43] as theorem 3.1 and extends Hoeffding's inequality to the case where the number of random variables averaged is itself a random variable. We refer the reader to the original manuscript for a proof of the result.

Theorem 2.3.3. *Let $\{X_i : i = 1, 2, \ldots\}$ be a sequence of i.i.d. non-negative uniformly bounded random variables, with $0 \leq X_i \leq D$ and $E[X_i] = \mu$ for all i, and let M be a positive integer-valued random variable bounded from above. Then for any real $\epsilon > 0$ and integer $N > 0$ it holds that*

$$P\left[\left|\frac{1}{M}\sum_{i=1}^{M} X_i - \mu\right| \geq \epsilon, M \geq N\right] \leq 2\exp\left[-N\left(\frac{D+\epsilon}{D}\log\frac{D+\epsilon}{D} - \frac{\epsilon}{D}\right)\right].$$

Before analyzing REGA, we first consider a simpler version of the algorithm in which $\hat{V}_{i+1}$ is replaced with V^*_{i+1} in equation 2.5. This removes the recursion from REGA under the assumption that the true reward-to-go at the next stage is known, and thus reduces the problem from a multi-stage problem to a single-stage problem. We refer to this algorithm as the *non-recursive ϵ-greedy algorithm* (NREGA). Later on, we derive results pertaining REGA.

In what follows, we make use of the following additional notation. Define $\Lambda = \max\left\{(c^2+c)^2, (c|A(s)|)^2\right\}$ and $\xi_\epsilon = \frac{R_{\max}H+\epsilon}{R_{\max}H}\log\left(\frac{R_{\max}H+\epsilon}{R_{\max}H}\right) - \frac{\epsilon}{R_{\max}H}$. Note that $\xi_\epsilon > 0$ for $\epsilon > 0$. Moreover, consider a fixed but arbitrary stage i of the algorithm, state s and action a of the MDP. Denote by $a_i(t)$ the arm played at iteration t of the algorithm. Let $\mathbb{1}\{a_i(t) = a\}$ be a random variable indicating whether or not the chosen arm $a_i(t)$ is in fact arm a, and let $\mathbb{1}^U\{a_i(t) = a\}$ denote whether or not the arm was chosen (independently) per the uniformly random rule in step 2 of the REGA algorithm (see figure 2.3). Let $N_a^{i,M}(s) = \sum_{t=1}^{M}\mathbb{1}\{a_i(t) = a\}$ be a random variable denoting the total number of times action a is played during the first M iterations of the algorithm at stage s. Similarly, let $U_a^{i,M}(s) = \sum_{t=1}^{M} \mathbb{1}^U\{a_i(t) = a\}$.

The following result is carved out from the proof of lemma 3.2 in [43]. It was originally stated for the RASA algorithm but also applies to NREGA.

Lemma 2.3.4. *For any positive integer N, it holds for NREGA that*

$$P\left[\left|\hat{Q}_{i,M}(s,a) - Q_i^*(s,a)\right| \geq \epsilon\right] \leq 2\exp[-\xi_\epsilon N] + P\left[N_a^{i,M}(s) \leq N\right].$$

Proof. By NREGA, the estimated reward to go of any action a after M iterations of the algorithm is given by

$$\hat{Q}_{i,M}(s,a) = \frac{1}{N_a^{i,M}(s)} \sum_{j=1}^{N_a^{i,M}(s)} \left(R\left(s,a,w_j\right) + V_{i+1}^*\left(s,a,w_j\right)\right).$$

Since the noise sequence of random variables $\{w_j : j \geq 1\}$ are i.i.d, we apply theorem 2.3.3 to obtain

$$P\left[\left|\hat{Q}_{i,M}(s,a) - Q_i^*(s,a)\right| \geq \epsilon, N_a^{i,M}(s) \geq N\right] \leq 2\exp[-\xi_\epsilon N].$$

Now define the events $\mathcal{A}_M = \left\{\left|\hat{Q}_{i,M}(s,a) - Q_i^*(s,a)\right| \geq \epsilon\right\}$ and $\mathcal{B}_M = \left\{N_a^{i,M}(s) \geq N\right\}$. By the *law of total probability* we have

$$\begin{aligned} P\left[\mathcal{A}_M\right] &= P\left[\mathcal{A}_M \cap \mathcal{B}_M\right] + P\left[\mathcal{A}_M | \mathcal{B}_M^c\right] P\left[\mathcal{B}_M^c\right] \\ &\leq P\left[\mathcal{A}_M \cap \mathcal{B}_M\right] + P\left[\mathcal{B}_M^c\right] \\ &\leq 2\exp[-\xi_\epsilon N] + P\left[N_a^{i,M}(s) \leq N\right]. \end{aligned}$$

This completes the proof.

□

The next lemma gives a finite time bound on the estimation errors of the Q-functions induced by NREGA at the exit of the algorithm as a function of the number of allocated simulations at each stage. The result is analogous to lemma 3.2 in [43].

Lemma 2.3.5. *Let $\epsilon > 0$ and $\alpha \in (1,2)$ be fixed. Consider a single stage of NREGA applied to a MDP using parameters $c \geq 2\sqrt{3}$ and a total number of allocated simulations M large enouch such that $M \geq \Lambda^{1/\alpha}$ and $M^\alpha \geq M + 2$.*

Then for any action a and state s, we have

$$P\left[\left|\hat{Q}_{i,M}(s,a) - Q_i^*(s,a)\right| \geq \epsilon\right] < \frac{(c+1)^2}{2}\log\left(1 + M^{1-\alpha}\right) \qquad (2.12)$$
$$+ \frac{ec}{\sqrt{2}}\left(\frac{c}{\sqrt{2}}\exp\left\{\frac{\sqrt{2}-c}{\sqrt{2}} - \xi_\epsilon\right\}\right)^{M^{1-\alpha/2}}.$$

Proof. Consider a fixed but arbitrary stage i of the algorithm, state s and action a of the MDP. By lemma 2.3.4, for any positive integer N we have

$$P\left[\left|\hat{Q}_{i,M}(s,a) - Q_i^*(s,a)\right| \geq \epsilon\right] \leq 2\exp[-\xi_\epsilon N] + P\left[N_a^{i,M}(s) \leq N\right]. \qquad (2.13)$$

We first bound the term $P\left[N_a^{i,M}(s) \leq N\right]$, and then replace N with an appropriate function of M.

Indeed, $N_a^{i,M}(s) \geq U_a^{i,M}(s)\,\forall a \in A(s)$ and $N_a^{i,M}(s) > U_a^{i,M}(s)$ for some $a \in A(s)$. Therefore, $N_a^{i,M}(s)$ dominates $U_a^{i,M}(s)$ *almost surely*. Hence,

$$P\left[N_a^{i,M}(s) \leq N\right] \leq P\left[U_a^{i,M}(s) \leq N\right] = P\left[\sum_{t=1}^{M} \mathbb{1}^U\{a_i(t) = a\} \leq N\right]. \qquad (2.14)$$

For iteration $t = 1, \ldots, M$ of the algorithm, let ϵ_t be defined as per equation 2.4. Per the algorithm description, the random variables $\mathbb{1}^U\{a_i(t) = a\}$ are independent and have probabilities

$$P\left[\mathbb{1}^U\{a_i(t) = a\} = 1\right] = \frac{\epsilon_t}{|A(s)|} = \begin{cases} \frac{c}{\sqrt{t}}, & t \geq (c|A(s)|)^2 \\ \frac{1}{|A(s)|}, & \text{o/w} \end{cases}.$$

Let integer $r \geq \Lambda$ be a constant "shift" variable, and let $X_1, ..., X_M$ be independent Bernoulli variables with success probabilities $p_t = \frac{c}{\sqrt{r+t}}$. Since $p_t \leq \frac{\epsilon_t}{|A(s)|}\,\forall t = 1, \ldots, M$, then each random variable X_t is (first-order) stochastically dominated by the corresponding random variable $\mathbb{1}^U\{a_i(t) = a\}$ for any fixed t. Hence, it holds that

$$P\left[\sum_{t=1}^{M} \mathbb{1}^U\{a_i(t) = a\} \leq N\right] \leq P\left[\sum_{t=1}^{M} X_t \leq N\right]. \qquad (2.15)$$

Combining equations 2.14 and 2.15 yields

$$P\left[N_a^{i,M}(s) \leq N\right] \leq P\left[\sum_{t=1}^{M} X_t \leq N\right]. \tag{2.16}$$

Let $\lambda_t = -\log(1 - \frac{c}{\sqrt{t}})$. We now applying theorem 2.3.2 to equation 2.16 which yields

$$P\left[N_a^{i,M}(s) \leq N\right] \leq P\left[Y\left(\sum_{t=r+1}^{r+M} \lambda_t\right) \leq N\right] + \frac{1}{2}\sum_{t=r+1}^{r+M} \lambda_t^2. \tag{2.17}$$

It is known that applying Chernoff bounds to a Poisson random variable $Y(\lambda)$ yields

$$P\left(Y \leq y\right) \leq \frac{e^{-\lambda}(e\lambda)^y}{y^y}, \text{ for } y \leq \lambda. \tag{2.18}$$

Consider $N \leq \dfrac{cM}{\sqrt{r+M+1}}$. It follows from lemma 2.3.1 that $N \leq \displaystyle\sum_{t=r+1}^{r+M} \lambda_t$. Hence, we can apply the Chernoff bound from equation 2.18 to equation 2.17 to obtain

$$P\left[N_a^{i,M}(s) \leq N\right] \leq \frac{1}{2}\sum_{t=r+1}^{r+M} \lambda_t^2 + \frac{\exp\left[-\sum_{t=r+1}^{r+M} \lambda_t\right]\left(e\sum_{t=r+1}^{r+M} \lambda_t\right)^N}{N^N}. \tag{2.19}$$

We wish to give an elementary upper bound on the righthand side of 2.19. Since $\frac{e^{-\lambda}(e\lambda)^y}{y^y}$ is decreasing in λ for $y \leq \lambda$, it follows from lemma 2.3.1 that

$$\begin{aligned} P\left[N_a^{i,M}(s) \leq N\right] \leq{}& \frac{1}{2}(c+1)^2 \log\left(1 + \frac{M}{r}\right) \\ &+ \exp\left[-\frac{cM}{\sqrt{r+M+1}}\right]\left(\frac{ecM}{N\sqrt{r+M+1}}\right)^N. \end{aligned} \tag{2.20}$$

Now let $r = \lceil M^\alpha \rceil$ and $N = \lceil M^{1-\alpha/2} \rceil$. For the bound in equation 2.20 to converge to zero as $M \to \infty$, it is necessary that $\alpha \in (1, 2)$. Moreover, the condition $N \leq \dfrac{cM}{\sqrt{r+M+1}}$ is indeed satisfied since $c \geq 2\sqrt{3}$ implies that

$$
\begin{aligned}
M \geq \left(\frac{\sqrt{3}}{c-\sqrt{3}}\right)^{\frac{2}{2-\alpha}} &\Leftrightarrow M^{1-\alpha/2}+1 \leq \frac{cM^{1-\alpha/2}}{\sqrt{3}} \\
&\Rightarrow M^{1-\alpha/2}+1 \leq \frac{cM}{\sqrt{M^\alpha+M+2}} \\
&\Rightarrow \lceil M^{1-\alpha/2} \rceil \leq \frac{cM}{\sqrt{M^\alpha+M+2}} \\
&\Leftrightarrow N \leq \frac{cM}{\sqrt{r+M+1}},
\end{aligned}
$$

where the second equation follows from the restriction $M^\alpha \geq M+2$.

Substituting r and N into equation 2.20 and relaxing the bound yields

$$
\begin{aligned}
P\left[N_a^{i,M}(s) \leq N\right] \leq \frac{1}{2}(c+1)^2 \log\left(1+\frac{M}{M^\alpha}\right)+ \\
\exp\left[-\frac{cM}{\sqrt{M^\alpha+M+2}}\right]\left(\frac{ecM^{\alpha/2}}{\sqrt{M^\alpha+M+1}}\right)^{M^{1-\alpha/2}+1}.
\end{aligned}
\tag{2.21}
$$

By the condition $M^\alpha \geq M+2$, we have

$$
\begin{aligned}
&\exp\left[-\frac{cM}{\sqrt{M^\alpha+M+2}}\right]\left(\frac{ecM^{\alpha/2}}{\sqrt{M^\alpha+M+1}}\right)^{M^{1-\alpha/2}+1} \\
&\leq \exp\left[-\frac{cM}{\sqrt{2M^\alpha}}\right]\left(\frac{ec}{\sqrt{2}}\right)^{M^{1-\alpha/2}+1} \\
&= \exp\left[-\frac{cM^{1-\alpha/2}}{\sqrt{2}}+M^{1-\alpha/2}+1\right]\left(\frac{c}{\sqrt{2}}\right)^{M^{1-\alpha/2}+1} \\
&= \frac{ec}{\sqrt{2}}\left(\frac{c}{\sqrt{2}}\exp\left\{\frac{\sqrt{2}-c}{\sqrt{2}}\right\}\right)^{M^{1-\alpha/2}}.
\end{aligned}
\tag{2.22}
$$

Combining equations 2.21 and 2.22 yields

$$
P\left[N_a^{i,M}(s) \leq N\right] \leq \frac{(c+1)^2}{2}\log\left(1+M^{1-\alpha}\right)+\frac{ec}{\sqrt{2}}\left(\frac{c}{\sqrt{2}}\exp\left\{\frac{\sqrt{2}-c}{\sqrt{2}}\right\}\right)^{M^{1-\alpha/2}}.
\tag{2.23}
$$

Recall that $N = \lceil M^{1-\alpha/2} \rceil$ and that $\xi_\epsilon > 0$ for $\epsilon > 0$. Therefore, equation 2.13 can be relaxed to

$$P\left[\left|\hat{Q}_{i,M}(s,a) - Q_i^*(s,a)\right| \geq \epsilon\right] < \frac{(c+1)^2}{2}\log\left(1 + M^{1-\alpha}\right) + \frac{ec}{\sqrt{2}}\left(\frac{c}{\sqrt{2}}\exp\left\{\frac{\sqrt{2}-c}{\sqrt{2}} - \xi_\epsilon\right\}\right)^{M^{1-\alpha/2}}, \tag{2.24}$$

which completes the proof. □

Corollary 2.3.6. *Suppose the conditions of lemma 2.3.5 hold. Let* $\Phi = \frac{c}{\sqrt{2}}\exp\left\{\frac{\sqrt{2}-c}{\sqrt{2}} - \xi_\epsilon\right\} < 1$. *If* M *is large enough such that* $\Phi^{M^{1-\alpha/2}} \leq M^{1-\alpha}$, *then for any state* s *and action* a *we have*

$$P\left[\left|\hat{Q}_{i,M}(s,a) - Q_i^*(s,a)\right| \geq \epsilon\right] < \frac{(c+1)^2 + \sqrt{2}ec}{2}M^{1-\alpha}.$$

Proof. It is well-known (*c.f.* [142]) that $\log(1+x) \leq x$ for $x > -1$. The result immediately follows by relaxing the bound in lemma 2.3.5. □

Note that the result in corollary 2.3.6 can be made arbitrarily close to $O(1/M)$ in the limit by selecting α arbitrarily close to 2. However, as α is increased towards 2, the minimum value of M for which the finite time bound applies grows exponentially.

For purposes of our theoretical analysis, we now make the following assumption that is identical to that made in [43].

Assumption 2.3.7. We assume that the optimal action at each stage of the algorithm is unique, so that the MDP has a unique optimal policy. That is, $\theta = \min_{s\in S, i=0,\ldots H-1} \theta_i(s) > 0$, where $\theta_i(s) = \min_{a\neq a^*}(V_i^*(s) - Q_i^*(s,a))\,\forall\, s \in S, a \in A(s)$ and $i = 0,\ldots,H-1$.

Note that a small value of θ means increased difficulty in differentiating between the truly optimal action and the best suboptimal action(s). Therefore, θ measures the "size" of the problem.

We are finally ready to make the leap from NREGA to the recursive REGA algorithm. We start by restating lemma 3.3 of [43] in a manner that is useful and applicable to REGA[1]. We refer the reader to the original manuscript for the proof.

[1]Equation 11 of Lemma 3.3 in [43] is unnecessarily loose, where $N_a^i(x)$ is bounded from above by K_{i+1} instead of K_i. Using the lemma as-is will cause our lower bound on the

Lemma 2.3.8. *Consider REGA applied with M_i simulations allocated to each stage $i = 0, \dots, H-1$. Suppose that assumption 2.3.7 holds, and suppose that $\delta_i \in (0,1)$ satisfies $P\left(\left|\hat{Q}_{i,M_i}(s,a) - Q_i^*(s,a)\right| > \frac{\theta}{2^{i+2}}\right) < \delta_i$ for all i and $s \in S$. At the exit step we have*

$$P\left(\left|\hat{V}_{0,M_0}(s_0) - V_0^*(s_0)\right| < \frac{\theta}{2}\right) > (1-\delta_0)\prod_{i=1}^{H-1}(1-\delta_i)^{\Pi_{j=0}^{i-1} M_j}.$$

The following definition is needed in the next theorem:

$$\xi_i = \frac{R_{\max}H + \theta/2^{i+2}}{R_{\max}H}\log\left(\frac{R_{\max}H + \theta/2^{i+2}}{R_{\max}H}\right) - \frac{\theta/2^{i+2}}{R_{\max}H}.$$

We now state our main result pertaining the REGA algorithm, in the form of a finite-time bound on the probability that the approximation error of the MDP value function will exceed half the size of the problem, *i. e.*, $\theta/2$. The bound is given as a function of the number of simulations M_0 allocated at the first stage of the MDP.

Theorem 2.3.9. *Suppose that assumption 2.3.7 holds and that the requirements of corollary 2.3.6 are satisfied for each stage i of the MDP, where ξ_ϵ is replaced with ξ_i. Let $\phi > 1$ be fixed, and suppose that $M_i = \prod_{j=0}^{i-1} M_j^{\phi/(\alpha-1)}$ for all $i = 1, \dots, H-1$. Denote $\Psi = \frac{(c+1)^2}{2} + \frac{ec}{\sqrt{2}}$. Then at the exit step of REGA we have*

$$P\left(\left|\hat{V}_{0,M_0}(s_0) - V_0^*(s_0)\right| < \frac{\theta}{2}\right) > \left(1 - \Psi M_0^{1-\alpha}\right)^{HM_0^{(\alpha-1)/\phi}}.$$

Proof. By corollary 2.3.6, it holds for each stage $i = 0, \dots, H-1$ that

$$P\left(\left|\hat{Q}_{i,M_i}(s,a) - Q_i^*(s,a)\right| > \frac{\theta}{2^{i+2}}\right) < \Psi M_i^{1-\alpha}.$$

probability in theorem 2.3.9 to converge to 0 instead of 1 as the number of simulations at each stage of the MDP is increased to infinity, whereas one would expect the bound to improve as the number of samples is increased. We necessarily use the tighter bound.

By lemma 2.3.8, we have

$$\begin{aligned}
&P\left(\left|\hat{V}_{0,M_0}(s_0) - V_0^*(s_0)\right| < \frac{\theta}{2}\right) \\
&\quad > \left(1 - \Psi M_0^{1-\alpha}\right) \prod_{i=1}^{H-1} \left(1 - \Psi M_i^{1-\alpha}\right)^{\prod_{j=0}^{i-1} M_j} \\
&\quad = \left(1 - \Psi M_0^{1-\alpha}\right) \prod_{i=1}^{H-1} \left(1 - \Psi M_i^{1-\alpha}\right)^{M_i^{(\alpha-1)/\phi}} \\
&\quad > \left(1 - \Psi M_0^{1-\alpha}\right)^{HM_0^{(\alpha-1)/\phi}},
\end{aligned}$$

where the last inequality follows from the fact that $\left(1 - \Psi x^{1-\alpha}\right)^{x^{(\alpha-1)/\phi}}$ is increasing in x. This completes the proof. □

We conclude by noting that the bound in theorem 2.3.9 goes to 1 as $M_0 \to \infty$.

2.4 The OREGA Method

In the previous section, we mentioned that the REGA method uses an ϵ-greedy scheme that differs from the one commonly cited in the literature. In this section, we consider a modification to REGA that is consistent with the heavily-cited ϵ-greedy method of [13], where the denominator $\sqrt{m}$ in equation 2.4 is replaced with m. We refer to this method as OREGA (original REGA).

The analysis of OREGA that we provide herein follows the same steps provided in the previous section. The following result is needed to justify some algebraic steps. It is analogous to lemma 2.3.1 and the proof is similar, so we omit the proof.

Lemma 2.4.1. *Let $c > 1$ be real-valued, and let $r \geq 1$ and $M \geq 1$ be fixed integers. Then the following inequalities hold:*

- *If $r > c + 1$, then*

$$0 < c \log\left(1 + \frac{M}{r+1}\right) \leq \sum_{i=r+1}^{r+M} - \log\left(1 - \frac{c}{i}\right). \tag{2.25}$$

- *If $r \geq c^2 + c$, then*

$$\sum_{i=r+1}^{r+M} \log^2(1 - \frac{c}{i}) \leq (c+1)^2 \frac{M}{r(r+M)}. \tag{2.26}$$

Now define $\bar{\Lambda} = \lceil \max(c^2 + c, c|A(s)|) \rceil$. The following result pertaining the one-stage non-recursive OREGA method (NOREGA) is analogous to lemma 2.3.5.

Lemma 2.4.2. *Let $\epsilon > 0$ be fixed. Consider a single stage of NOREGA applied to a MDP using parameter $c > 1$ and a total number of allocated simulations M large enough such that*

$$M \geq \left(\max \left\{ \exp\left[\exp\left[\frac{\xi_\epsilon (c-1)}{d^2}\right]\right], \exp\left[\frac{\xi_\epsilon c e^{1+\xi_\epsilon}}{d^2}\right], \sqrt{2}\bar{\Lambda} + 1 \right\} \right)^2.$$

Then for any action a and state s, we have

$$P\left[\left|\hat{Q}_{i,M}(s,a) - Q_i^*(s,a)\right| \geq \epsilon\right] < \frac{2(c+1)^2}{\sqrt{M}}.$$

Proof. The proof of lemma 2.3.5 applies until equation 2.16. From there, we follow similar ideas but the derivation is different.

Let $\lambda_t = -\log(1 - \frac{c}{t})$. We now applying Theorem 2.3.2 to equation 2.16 which yields

$$P\left[N_a^{i,M}(s) \leq N\right] \leq P\left[Y\left(\sum_{t=r+1}^{r+M} \lambda_t\right) \leq N\right] + \frac{1}{2}\sum_{t=r+1}^{r+M} \lambda_t^2. \tag{2.27}$$

It is known that applying Chernoff bounds to a Poisson random variable X with mean λ yields

$$P(X \leq x) \leq \frac{e^{-\lambda}(e\lambda)^x}{x^x}, \text{ for } x \leq \lambda. \tag{2.28}$$

For brevity, let $J = 1 + \frac{M}{r+1}$. Now consider $N \leq c \log J$. Since $r > c + 1$, it follows from equation 2.25 that $N \leq \sum_{t=r+1}^{r+M} \lambda_t$. Hence, we can apply the

Chernoff bound from equation 2.28 to equation 2.27 to obtain

$$P\left[N_a^{i,M}(s) \leq N\right] \leq \frac{1}{2}\sum_{t=r+1}^{r+M} \lambda_t^2 + \frac{\exp\left[-\sum_{t=r+1}^{r+M}\lambda_t\right]\left(e\sum_{t=r+1}^{r+M}\lambda_t\right)^N}{N^N}. \tag{2.29}$$

We wish to give an elementary upper bound on the righthand side of 2.29. Since $\frac{e^{-\lambda}(e\lambda)^x}{x^x}$ is decreasing in λ for $x \leq \lambda$, it follows from equation 2.25 that

$$\begin{aligned} \frac{\exp\left[-\sum_{t=r+1}^{r+M}\lambda_t\right]\left(e\sum_{t=r+1}^{r+M}\lambda_t\right)^N}{N^N} &\leq \frac{\exp\left[-c\log J\right](ec\log J)^N}{N^N} \\ &= \left(\frac{ec}{N}\log J\right)^N J^{-c}. \end{aligned} \tag{2.30}$$

Moreover, by equation 2.26 we have

$$\sum_{t=r+1}^{r+M}\lambda_t^2 \leq (c+1)^2 \frac{M}{r(r+M)}. \tag{2.31}$$

Combining equations 2.29, 2.30 and 2.31 we obtain

$$P\left[N_a^{i,M}(s) \leq N\right] \leq \left[\frac{ec}{N}\log J\right]^N J^{-c} + (c+1)^2\frac{M}{2r(r+M)}. \tag{2.32}$$

We now choose an appropriate value for N. In particular, we set

$$N = \left\lfloor \frac{c\log J}{\log\log J} \right\rfloor. \tag{2.33}$$

Note that

$$\begin{aligned} N \leq \frac{c\log J}{\log\log J} &\Leftrightarrow N\log\log J \leq c\log J \\ &\Leftrightarrow (\log J)^N \leq J^c \\ &\Leftrightarrow (\log J)^N J^{-c} \leq 1 \end{aligned} \tag{2.34}$$

and

$$1 \leq \frac{\log J}{\log\log J} \Leftrightarrow \frac{(c-1)\log J}{\log\log J} \leq \frac{c\log J}{\log\log J} - 1 \tag{2.35}$$
$$\Rightarrow \frac{(c-1)\log J}{\log\log J} \leq \left\lfloor \frac{c\log J}{\log\log J} \right\rfloor = N.$$

Therefore we have

$$\begin{aligned} &P\left[\left|\hat{Q}_{i,M}(s,a) - Q_i^*(s,a)\right| \geq \epsilon\right] \\ &< 2e^{-\xi_\epsilon N} + \left(\frac{ec}{N}\right)^N (\log J)^N J^{-c} + (c+1)^2 \frac{M}{2r(r+M)} \\ &\leq 2e^{-\xi_\epsilon N} + \left(\frac{ec}{N}\right)^N + (c+1)^2 \frac{M}{2r(r+M)} \\ &\leq 3\left(\max\left\{e^{-\xi_\epsilon}, \frac{ec}{N}\right\}\right)^N + (c+1)^2 \frac{M}{2r(r+M)} \\ &\leq 3\left(\max\left\{e^{-\xi_\epsilon}, \frac{ec\log\log J}{(c-1)\log J}\right\}\right)^{\frac{(c-1)\log J}{\log\log J}} + (c+1)^2 \frac{M}{2r(r+M)}. \end{aligned} \tag{2.36}$$

where the second inequality follows by equation 2.34 and the last inequality follows by equation 2.35 and the fact that the first term is decreasing in N.

It remains to select an appropriate value for r. The essential requirement for the bound in equation 2.36 to converge to zero is that $r \to \infty$ and $r/M \to 0$ as $M \to \infty$. Ideally, one would optimize over r to arrive at the tightest possible bound in 2.36. However, to simplify the bound, we chose r in the form $\lceil M^\alpha - 2 \rceil$ for some $\alpha \in (0,1)$. Although not necessarily optimal, a choice of $r = \left\lceil \sqrt{M} - 2 \right\rceil$ allows to simplify the final result. Note that

$$J = 1 + \frac{M}{r+1} \geq \frac{M}{r+1} \geq \frac{M}{\left(\sqrt{M}-1\right)+1} = \sqrt{M}. \tag{2.37}$$

Moreover, note that $r \geq \bar{\Lambda} > 2$ since $c > 1$. It hence holds that

$$\frac{M}{2r(r+M)} \leq \frac{1}{2r} \leq \frac{1}{r+2} \leq \frac{1}{(\sqrt{M}-2)+2} = \frac{1}{\sqrt{M}}. \tag{2.38}$$

Since the first and second terms of equation 2.36 are decreasing in J and r, respectively, equations 2.38 and 2.37 can be used to relax equation 2.36 to

$$
\begin{aligned}
&P\left[\left|\hat{Q}_{i,M}(s,a) - Q_i^*(s,a)\right| \geq \epsilon\right] \\
&< 3\left(\max\left\{e^{-\xi_\epsilon}, \frac{ec \log\log\sqrt{M}}{(c-1)\log\sqrt{M}}\right\}\right)^{\frac{(c-1)\log\sqrt{M}}{\log\log\sqrt{M}}} + (c+1)^2 \frac{1}{\sqrt{M}} \\
&\leq 3\left(\exp\left[-\xi_\epsilon(c-1)\right]\right)^{\frac{\log\sqrt{M}}{\log\log\sqrt{M}}} + \frac{(c+1)^2}{\sqrt{M}},
\end{aligned}
\tag{2.39}
$$

where the last equation follows since the restriction $M \geq \left(\exp\left[\xi_\epsilon c e^{1+\xi_\epsilon}\right]\right)^2$ implies that $e^{-\xi_\epsilon} \geq \frac{ec\log\log\sqrt{M}}{(c-1)\log\sqrt{M}}$. Moreover, it follows from the restriction $M \geq (\exp\left[\exp\left[\xi_\epsilon(c-1)\right]\right])^2$ that

$$
\left(\exp\left[-\xi_\epsilon(c-1)\right]\right)^{\frac{\log\sqrt{M}}{\log\log\sqrt{M}}} = M^{\left(\frac{-\xi_\epsilon(c-1)}{2\log\log\sqrt{M}}\right)} \leq \frac{1}{\sqrt{M}}. \tag{2.40}
$$

Combining equations 2.39 and 2.40 and the fact that $(c+1)^2 \geq 3$, it follows that

$$
P\left[\left|\hat{Q}_{i,M}(s,a) - Q_i^*(s,a)\right| \geq \epsilon\right] \leq \frac{2(c+1)^2}{\sqrt{M}}.
$$

Finally, note that the requirement $r \geq \bar{\Lambda}$ is satisfied for $M \geq \left(\bar{\Lambda}+1\right)^2$. This completes the proof. $\square$

Finally, we give the analogues of theorem 2.3.9 pertaining the multi-stage recursive OREGA algorithm. Define

$$
\beta_i = \left(\max\left\{\exp\left[\exp\left[\xi_i(c-1)\right]\right], \exp\left[\xi_i c e^{1+\xi_i}\right], \sqrt{2}\bar{\Lambda}+1\right\}\right)^2,
$$

then the following result holds. We omit the proof as it is identical to that of theorem 2.3.9.

Theorem 2.4.3. *Suppose that assumption 2.3.7 holds and that the requirements of lemma 2.4.2 are satisfied. Let $\phi > 2$ be fixed. Suppose that $M_0 \geq \beta_0$ and that $M_i = \prod_{j=0}^{i-1} M_j^\phi$ for all $i = 1, \ldots, H-1$. Then at the exit step of OREGA we have*

$$
P\left(\left|\hat{V}_{0,M_0}(s_0) - V_0^*(s_0)\right| < \frac{\theta}{2}\right) > \left(1 - \frac{2(c+1)^2}{\sqrt{M_0}}\right)^{HM_0^{1/\phi}}.
$$

We note that the bound in theorem 2.4.3 approaches 1 as M_0 is increased towards infinity. We conclude by noting that the bound on REGA provided in theorem 2.3.9 is better than the bound on OREGA provided in theorem 2.4.3. As we will discuss in section 2.6, REGA was found to empirically outperform OREGA when applied to our benchmark problems. Nonetheless, it is unclear from our analysis whether or not the provided bounds are tight, and so no theoretical conclusions comparing the two algorithms can be drawn solely based on our results.

2.5 Practical Limitations

Value iteration and policy iteration are classical exact methods for solving MDPs. One iteration of these methods has runtime complexity $O(|S|^2|A|)$ and $O(|S|^2|A| + |S|^3)$, respectively, which are nonlinear in the size of the state space $|S|$. On the other hand, the (O)REGA algorithm has runtime complexity $O\left(\left(\max_{i=0,\dots,H-1}\{M_i\}\right)^H\right)$, which is independent of the size of the state space [2]. Due to the backwards induction process, the runtime complexities of (O)REGA, RASA and AMS grow exponentially as the sampling horizon H is increased. Thus, these algorithms are particularly attractive for problems with a large state space $|S|$, small action space $|A|$ and small horizon H.

We now give two suggestions to dealing with MDPs with relatively large horizons.

2.5.1 Rolling Horizon Control (RHC)

As [75] points out, one solution to the exponential runtime complexity issue pertaining a large horizon H is to use *rolling horizon control* (RHC), also known in the literature as *receding/moving horizon control.* RHC provides an approximate solution to infinite horizon control problems in an online manner. The basic idea is to consider a fixed horizon length to determine the best policy/action at the current stage, ignoring stages far out into the future in hopes that they have little effect on the decision being made at the current stage. If the length of this “sliding” lookahead window is chosen to be large enough, then the error induced by this approach can be made small.

[2]Although the runtime complexities are also independent of the size of the action space, the accuracies of the algorithms are certainly dependent on the size of the action space. As such, (O)REGA, RASA and AMS are generally not well-suited for MDPs with large action spaces. Furthermore, unlike (O)REGA, AMS has to sample each action at least once.

To that effect, let π_{rh} denote the rolling H-horizon control policy for some fixed finite lookahead length H. The RHC framework proceeds by defining a sequence of smaller finite-horizon problems with non-stationary optimal policies $\{\pi_0^*, \pi_1^*, ..., \pi_{H-1}^*\}$, then returns a stationary optimal policy for the infinite problem $\pi_{rh} = \pi_0^*$. For the discounted reward criterion, the value of the policy induced by rolling horizon control converges to the true (infinite horizon) optimal value uniformly in the initial state. As such, the policy is asymptotically optimal and the rate of convergence is geometric in the length of the rolling horizon and is characterized by the discount factor γ. We emphasize that the RHC policy is determined based on the sequence of finite horizon problem. Yet, its value function $V^{\pi_{rh}}(s)$ is calculated against the original infinite horizon problem.

As usual, for any initial state $s \in S$, let $V^*(s)$ denote the true optimal value of the MDP. For an MDP with positive rewards, the following bound appears in [73]:

$$0 \leq V^*(s) - V^{\pi_{rh}}(s) \leq \frac{R_{\max}}{1-\gamma}\gamma^H, \quad s \in S.$$

2.5.2 Decreased Simulation Allocation into the Horizon

Another approach to dealing with the exponential runtime complexity of the AMS and (O)REGA methods is to reduce the number of simulations allocated for each stage as the horizon increases, especially in the case of discounted future rewards. As such, if M_i denotes the number of simulations allocated to stage $i = 0, ..., H-1$, we consider a non-increasing sequence $\{M_i\}$ of simulation allocations. The intuition here is that if the rewards obtained at future stages are discounted, then they can potentially be of less significance. Note though that our theoretical results outlined earlier require that the number of samples be increased in future states to guarantee the provided convergence rates. However, we believe these requirements are too strict, and our simulations in the next section demonstrate that using a fixed number of samples per stage can yield "good" results.

To illustrate the effect of a non-increasing sequence $\{M_i\}$, consider the two MDP benchmark problems *chain* and *loop* that appear in [56][3].

chain As shown in figure 2.4, this MDP consists of five states, each with two actions a and b. State 1 is the initial state. In many settings, the optimal policy for this domain is to take action a all the time. This optimal policy holds for the infinite horizon problem with discount factor $\gamma = 0.99$,

[3]Figures 2.4 and 2.5 are courtesy of [56].

as well as for the (non-discounted) finite horizon problem with horizon $H \geq 6$. The challenge is that learning algorithms can get trapped at the initial state always taking action b, thus yielding a series of smaller rewards. To achieve a stochastic setting, the actions are flipped with "slip" probabilities 0.2.

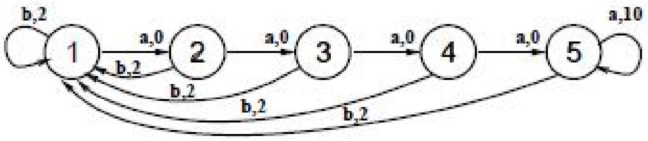


Figure 2.4: The *chain* MDP benchmark problem

loop Depicted in figure 2.5, this MDP consists of two loops. State 0 is the initial state, and actions are deterministic in this setting. A learning algorithm can end up following a myopic solution consisting of always picking action a at state 0 before backtracking from state 8 which has a larger reward but is too far into the future. The optimal policy for this problem is to pick action b everywhere.

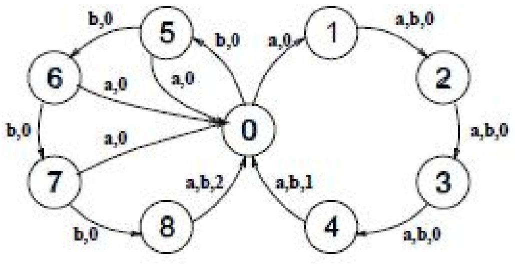


Figure 2.5: The *loop* MDP benchmark problem

We empirically compare three intuitive simulation allocation sequences $\{M_i\}$ against our benchmark problems to gain some insight into their effect on the REGA algorithm. The sequences all share the property that $M_0 \geq M_1 \geq ... \geq M_{H-1} \geq 1$. Two of the allocation schemas are geometric in some decay constant $\alpha > 0$, and have the form $M_i = \lceil \alpha^i M_0 \rceil$: One scheme samples each stage proportionally to the amount of discounting applied at that stage, *i. e.*, $\alpha = \gamma$, thus achieving $M_i = \lceil \gamma^i M_0 \rceil$ with the intuition being to allocate more samples towards stages with least discounted reward. The other scheme counteracts the curse of dimensionality via exponential decay using $\alpha = 1/2$ to obtain $M_i = \lceil M_0/2^i \rceil$. Such exponential decay will typically

Max Horizon	**Geometric** $M_i = \lceil (1/2)^i M_0 \rceil$		**Geometric** $M_i = \lceil \gamma^i M_0 \rceil$		**Linear** $M_i = \left\lceil M_0 - i\frac{M_0-1}{H-1} \right\rceil$		**Fixed** $M_i = M_0$	
	$\hat{V}$	**Runtime (sec)**	$\hat{V}$	**Runtime (sec)**	$\hat{V}$	**Runtime (sec)**	$\hat{V}$	**Runtime (sec)**
1	2.00	0.018	2.00	0.001	2.00	0.001	2.00	0.001
2	3.80	0.002	3.80	0.007	3.80	0.001	3.80	0.005
3	1.62	0.016	5.42	0.087	2.71	0.004	5.42	0.116
4	6.88	0.071	6.88	1.981	6.88	0.061	6.88	3.529
5	8.19	0.183	8.19	43.679	8.16	0.962	*DNF*	*DNF*
8	18.39	0.541	*DNF*	*DNF*	*DNF*	*DNF*	*DNF*	*DNF*
10	22.98	0.741	*DNF*	*DNF*	*DNF*	*DNF*	*DNF*	*DNF*
25	40.72	2.448	*DNF*	*DNF*	*DNF*	*DNF*	*DNF*	*DNF*

Table 2.1: Performance of REGA against the *chain* benchmark problem using various sample allocation decay strategies with $M_0 = 32$ and 1 minute simulation timeouts

result in vast improvement in the runtime performance of REGA. The third allocation scheme follows linear decay of the form $M_i = \lceil M_0 - i\alpha \rceil$. A choice of $\alpha = \frac{M_0-1}{H-1}$ yields $M_{H-1} = 1$.

We run simulations for the three decaying simulation allocation sequences as well as the constant allocation sequence where $M_i = M_0$. We set the discount factor to $\gamma = 0.9$ and the number of samples at stage zero to $M_0 = 32$. Each scenario was allowed to run for 1 minute before declaring that the run did not finish (denoted by *DNF*).Tables 2.1 and 2.2 summarize the results for the *chain* and *loop* benchmark problems, respectively. We observe that the geometric decay strategy with $\alpha = 1/2$ outperforms the other strategies. In particular, we note that out of all four simulation allocation strategies, the geometric decay strategy with $\alpha = 1/2$ is the only one useful in practical problems. The other allocation strategies do not run in reasonable time even for basic small problems when the horizon of the problem is large. Moreover, in cases were the runs with other allocation strategies did in fact complete within the imposed deadline on the simulation, the geometric decay strategy with $\alpha = 1/2$ returned a value function that is on par, yet at a significantly smaller fraction of the runtime cost.

2.6 Empirical Results

In this section, we provide emperical results that compare our (O)REGA algorithm against other algorithms. But first, we introduce the *SysAdmin* benchmark problem.

Max Horizon	**Geometric** $M_i = \lceil (1/2)^i M_0 \rceil$		**Geometric** $M_i = \lceil \gamma^i M_0 \rceil$		**Linear** $M_i = \left\lceil M_0 - i\frac{M_0-1}{H-1} \right\rceil$		**Fixed** $M_i = M_0$	
	$\hat{V}$	**Runtime (sec)**	$\hat{V}$	**Runtime (sec)**	$\hat{V}$	**Runtime (sec)**	$\hat{V}$	**Runtime (sec)**
1	0.00	0.120	0.00	0.001	0.00	0.001	0.00	0.001
2	0.00	0.002	0.00	0.006	0.00	0.001	0.00	0.005
3	0.00	0.016	0.00	0.082	0.00	0.006	0.00	0.106
4	0.00	0.071	0.00	1.940	0.00	0.059	0.00	3.355
5	1.31	0.173	1.31	38.727	1.31	0.869	*DNF*	*DNF*
8	1.31	0.508	*DNF*	*DNF*	*DNF*	*DNF*	*DNF*	*DNF*
10	2.08	0.778	*DNF*	*DNF*	*DNF*	*DNF*	*DNF*	*DNF*
25	2.95	2.325	*DNF*	*DNF*	*DNF*	*DNF*	*DNF*	*DNF*

Table 2.2: Performance of REGA against the *loop* benchmark problem using various sample allocation decay strategies with $M_0 = 32$ and 1 minute simulation timeouts

2.6.1 The SysAdmin Problem

In what follows, we demonstrate the performance of (O)REGA when applied to the well-known *SysAdmin* discrete-time problem (*c.f.* [66]). In the *SysAdmin* problem, a system administrator manages B computing machines that are connected via a known network configuration, so that each machine $b = 1, \ldots, B$ is connected to a subset of "neighbor" machines $N(b)$. Let I_b^h denote the state of machine b at horizon $h = 0, \ldots, H-1$, where $I_b^h = 1$ indicates that the machine is in working state, and $I_b^h = 0$ indicates that the machine is in faulted state. The state of the system at time h can be captured as a binary vector $< I_1^h, I_2^h, \ldots, I_B^h >$, and so the size of the state space is $2^{|B|}$, which is exponential in the number of machines. At each time step, the system administrator collects rewards $r(b)$ associated with each machine b that is in working state. The objective is to maximize $E\left[\sum_{h=0}^{H-1}\sum_{b=1}^{B} r(b)\, I_b^h\right]$, the expected total non-discounted reward over the entire horizon of the problem, where the expectation is taken with respect to machine states.

At each time step, the system administrator has to decide to either reboot a particular machine, or not reboot any machine at all. Only one machine can be rebooted at each time step. Hence, the action space consists of $|B|+1$ actions. We denote the action taken at each horizon h by a binary vector $< A_1^h, A_2^h, \ldots, A_B^h >$, where $A_b^h = 1$ if machine b is rebooted, and $A_b^h = 0$ otherwise. Since at each time step at most one machine can be rebooted, we have $0 \leq \sum_{b=1}^{B} A_b^h \leq 1$.

Machine states are controlled via the following rules: If any machine in $N(b)$ is in faulted state, then b will also fail with high probability p_1. On the other hand, if none of the machines in $N(b)$ are in faulted state, then b can still fail with a much smaller probability p_2. When a machine is rebooted, it will fail to start up in working state with a very small probability p_3. The transition probabilities for each machine can be formally stated as follows:

1. A faulted machine that is not rebooted will remain in faulted state.

$$P\left(I_b^{h+1} = 0 \,|\, I_b^h = 0\,,\ A_b^h = 0\right) = 1.$$

2. A machine in working state that is not rebooted will become in faulted state with different probability depending on the state of neighbor machines.

$$P\left(I_b^{h+1} = 0 \,|\, I_b^h = 1\,,\ A_b^h = 0\right) = \begin{cases} p_1, & \text{if } \exists\, \beta \in N(b) \text{ s.t. } I_\beta^h = 0 \\ p_2, & \text{o/w} \end{cases}$$

$$P\left(I_b^{h+1} = 1 \,|\, I_b^h = 1\,,\ A_b^h = 0\right) = \begin{cases} 1 - p_1, & \text{if } \exists\, \beta \in N(b) \text{ s.t. } I_\beta^h = 0 \\ 1 - p_2, & \text{o/w} \end{cases}$$

3. Rebooting a machine can result in it becoming in faulted state (regardless of its current state).

$$P\left(I_b^{h+1} = 0 \,|\, I_b^h \in \{0,1\}\,,\ A_b^h = 1\right) = p_3.$$

$$P\left(I_b^{h+1} = 1 \,|\, I_b^h \in \{0,1\}\,,\ A_b^h = 1\right) = 1 - p_3.$$

Typically, in the literature, the *SysAdmin* problem is modeled as a finite horizon discrete-time MDP with total (non-discounted) reward criteria. Due to the size of the state space, it is often cited in articles related to so-called *factored MDPs* (*c.f.* [66]).

2.6.2 Results

We consider two network topologies: A *ring* configuration and a *star* configuration. In the ring configuration, each machine $b \in B$ is connected to two neighbor machines $b-1$ and $b+1$ (arithmetic taken modulo $|B|$). In the star

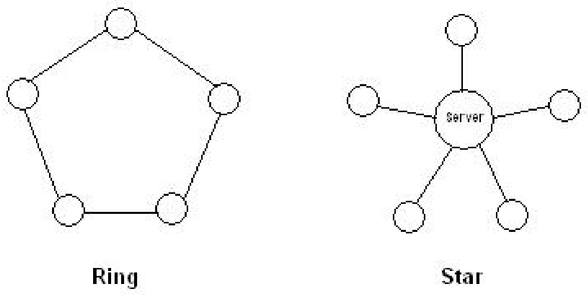


Figure 2.6: Example network topologies

configuration, there is one central server, and all other machines are connected only to the central server. The two topologies are depicted in figures 2.6.

In addition to REGA, RASA and OREGA, we also consider the trivial *purely greedy strategy* (PGS) where each action is sampled exactly once, after which each iteration of the algorithm plays the machine with highest observed average reward. We also simulate AMS and its two variants AMS1 and AMS2 previously mentioned in equations 2.2 and 2.3. We simulated all algorithms against the cycle and star network topologies of 10 machines with an initial state of all machines being online. The problems have state spaces with 2^{10} states and action spaces with 11 actions. We set the horizon of the problem to $H = 3$. The number of simulations was set to a constant across all stages, *i. e.*, $M_1 = M_2 = M_3$. The reward of each machine b being online was set to b, instead of being fixed across all machines, in order to eliminate any simplifications introduced by symmetry where trivial strategies can perform well. The machine failure rates were set to $p_1 = 0.7, p_2 = 0.1$ and $p_3 = 0.01$. For REGA and OREGA, we set the parameters c_i to $c_1 = c_2 = c_3 = 6$, and for RASA we set the tunable parameter $\mu_i = 1 - 2^{-1/M_i}$ as was chosen in the simulations described by the original paper [43][4]. Each simulation was repeated 30 times in order to calculate the average reward and standard error for each setting. AMS1 performed best, followed by REGA and RASA which were found to have similar performance, and as expected both algorithms outperformed the trivial PGS policy. Interestingly OREGA did not perform as well as REGA and RASA. Tables 2.3 and 2.4 summarizes the results of our simulations. The same results are plotted in figures 2.7 and 2.8. For both networks, the optimal value is $V^* = 149.93$, which was obtained via value iteration.

Recall the parameters c_i in REGA and OREGA control the amount of

[4]The performance of the ϵ-greedy method (and thus REGA and OREGA) is generally sensitive to the choice of values c_i. Our choice is not necessarily the best, but was found to perform well. We experiment later in this chapter with the effect of varying c_i.

M_i	**REGA**	**RASA**	**OREGA**	**AMS**	**AMS1**	**AMS2**	**PGS**
35	160.8(0.99)	158.7(0.85)	161.0(0.92)	118.2(1.03)	159.2(1.18)	142.3(1.68)	139.8(0.77)
50	157.9(0.82)	157.0(0.87)	155.9(0.81)	118.4(0.86)	156.1(0.89)	141.5(1.34)	136.7(0.76)
75	151.0(0.81)	150.5(0.84)	149.6(1.03)	119.7(0.56)	151.6(0.99)	143.0(1.28)	135.1(0.43)
100	148.1(0.88)	147.1(0.65)	143.7(0.58)	120.5(0.81)	147.5(1.48)	143.7(0.81)	132.7(0.51)
125	144.6(0.43)	145.0(0.52)	140.2(0.62)	121.7(0.47)	148.7(1.48)	143.6(1.04)	132.3(0.42)
150	144.5(0.52)	144.5(0.73)	138.7(0.49)	122.8(0.54)	149.7(2.22)	139.6(0.83)	131.4(0.33)
175	142.6(0.50)	141.8(0.46)	137.0(0.36)	122.7(0.53)	148.0(2.91)	139.8(0.89)	131.2(0.33)
200	142.1(0.41)	141.9(0.56)	136.3(0.31)	122.0(0.43)	147.0(1.80)	140.1(0.59)	131.8(0.35)
225	140.7(0.42)	141.5(0.57)	135.5(0.29)	122.3(0.23)	148.5(2.43)	137.4(0.67)	130.3(0.27)
250	140.7(0.39)	140.8(0.48)	135.0(0.27)	122.4(0.38)	146.9(2.91)	139.1(0.75)	131.2(0.39)

Table 2.3: Performance of REGA, RASA, OREGA, AMS and PGS against a *cycle* network configuration of 10 machines with machine failure probabilities $p_1 = 0.7, p_2 = 0.1, p_3 = 0.01$. The optimal value is $V^* = 149.93$.

M_i	**REGA**	**RASA**	**OREGA**	**AMS**	**AMS1**	**AMS2**	**PGS**
35	162.7(0.58)	159.5(0.93)	163.6(0.34)	118.9(1.11)	157.4(1.66)	140.6(2.17)	145.2(0.76)
50	158.7(0.72)	158.7(0.69)	156.3(0.78)	118.9(0.99)	153.5(1.16)	143.5(2.24)	141.4(0.64)
75	153.6(0.79)	152.4(0.73)	152.9(0.65)	120.8(0.89)	155.1(1.59)	145.7(1.63)	139.2(0.46)
100	149.6(0.52)	149.9(0.60)	146.7(0.58)	122.6(0.74)	150.5(1.39)	145.2(0.84)	138.4(0.47)
125	149.4(0.61)	147.3(0.57)	143.5(0.50)	122.8(1.02)	152.8(2.70)	144.6(1.09)	137.8(0.40)
150	147.8(0.44)	148.0(0.47)	142.4(0.32)	124.4(0.90)	149.7(1.16)	144.0(0.99)	138.0(0.37)
175	146.1(0.48)	146.3(0.41)	141.7(0.39)	123.5(0.63)	153.0(2.43)	142.5(0.60)	137.9(0.46)
200	145.8(0.51)	145.9(0.50)	141.3(0.29)	123.5(0.55)	150.4(1.94)	143.7(0.54)	136.4(0.40)
225	145.4(0.40)	145.0(0.49)	140.5(0.27)	124.0(0.61)	150.9(1.96)	143.1(0.74)	137.0(0.27)
250	145.1(0.37)	144.7(0.48)	140.3(0.26)	123.4(0.31)	151.7(2.69)	144.3(0.68)	136.5(0.37)

Table 2.4: Performance of REGA, RASA, OREGA, AMS and PGS against a *star* network configuration of 10 machines with machine failure probabilities $p_1 = 0.7, p_2 = 0.1, p_3 = 0.01$. The optimal value is $V^* = 149.93$.

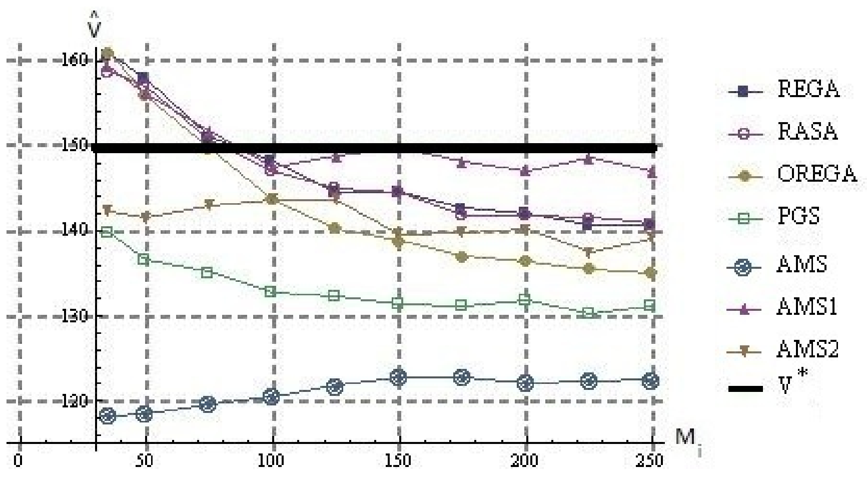


Figure 2.7: Performance of REGA, RASA, OREGA, AMS and PGS against a *cycle* network configuration of 10 machines with machine failure probabilities $p_1 = 0.7, p_2 = 0.1, p_3 = 0.01$.

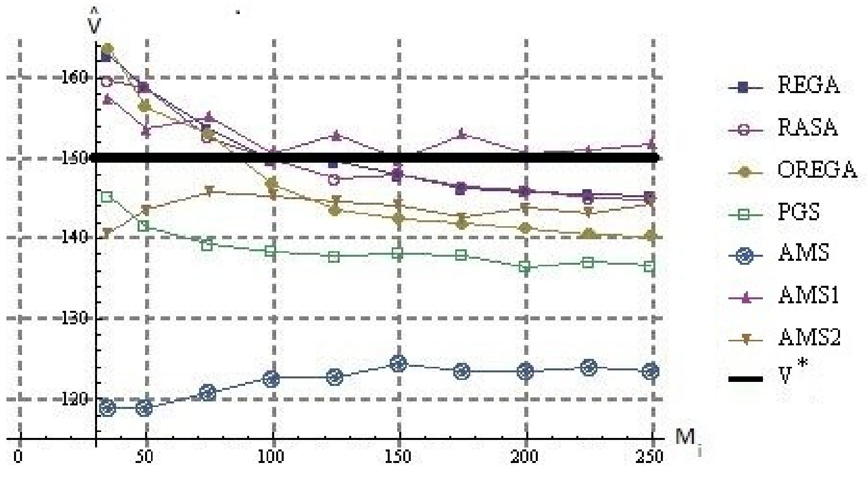


Figure 2.8: Performance of REGA, RASA, OREGA, AMS and PGS against a *star* network configuration of 10 machines with machine failure probabilities $p_1 = 0.7, p_2 = 0.1, p_3 = 0.01$.

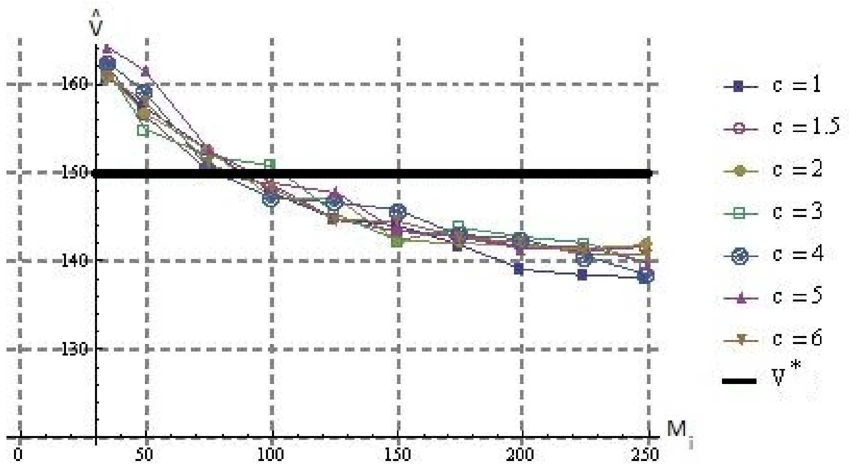


Figure 2.9: Effect of the exploration control parameters c_i on REGA when applied to a *cycle* network configuration of 10 machines with machine failure probabilities $p_1 = 0.7, p_2 = 0.1, p_3 = 0.01$.

exploration via the definition of ϵ_m^s in equation 2.4. In [13], the authors discuss how the observed empirical performance of the ϵ-greedy MAB (for a traditional single-stage stochastic MAB) algorithm is generally sensitive to the parameters c. Simulations for the effect of c_i on the cycle and star benchmark problems are summarized in figures 2.9 - 2.12. We found in our experiments that REGA is generally not very sensitive to the values c_i. On the other hand, OREGA is more sensitive for experiments with smaller simulation allocations per stage M_i, but not as sensitive to c_i for larger M_i at each stage i. This can be seen from the spacing between the plots for each value of c_i. When further spaced out, that is indication of higher sensitivity to the value of c_i, and when "bunched" together, that is indication of low sensitivity to c_i.

2.7 A Note on the Generalized Epsilon-Greedy Method and Regret of MABs

We discussed earlier how a changed to the exploration-exploitation model proposed by Auer in [13] provides improved empirical results when estimating the value function of an MDP. That is, REGA outperformed OREGA in our experiments. It is thus natural to consider whether or not the same change can yield improved results when the objective is to minimize regret for the classical stochastic MAB. It turns out that such modification is actually not desirable, as the resultant MAB algorithm can be shown to no longer attain logarithmic

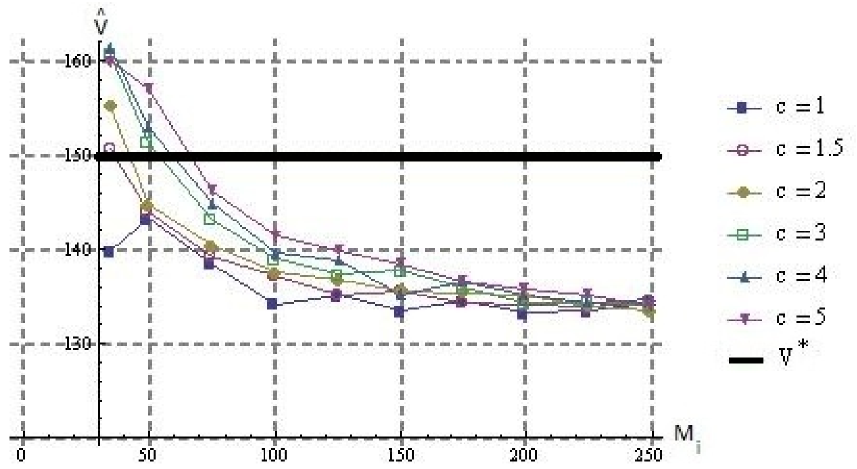


Figure 2.10: Effect of the exploration control parameters c_i on OREGA when applied to a *cycle* network configuration of 10 machines with machine failure probabilities $p_1 = 0.7, p_2 = 0.1, p_3 = 0.01$.

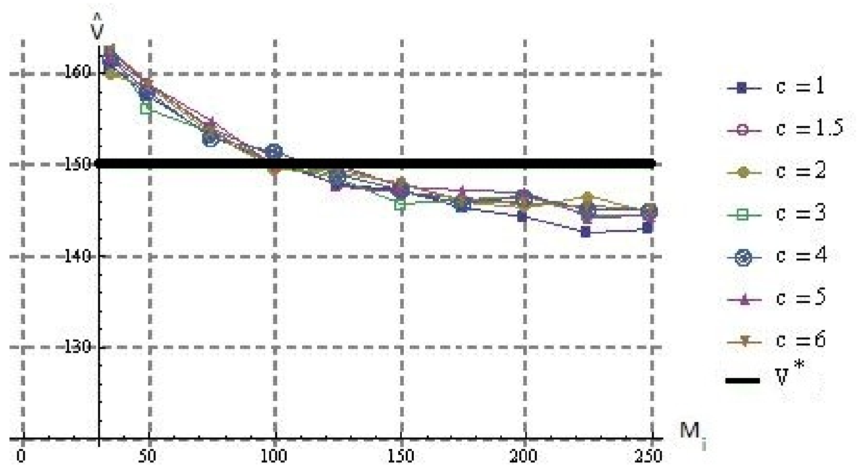


Figure 2.11: Effect of the exploration control parameters c_i on REGA when applied to a *star* network configuration of 10 machines with machine failure probabilities $p_1 = 0.7, p_2 = 0.1, p_3 = 0.01$.

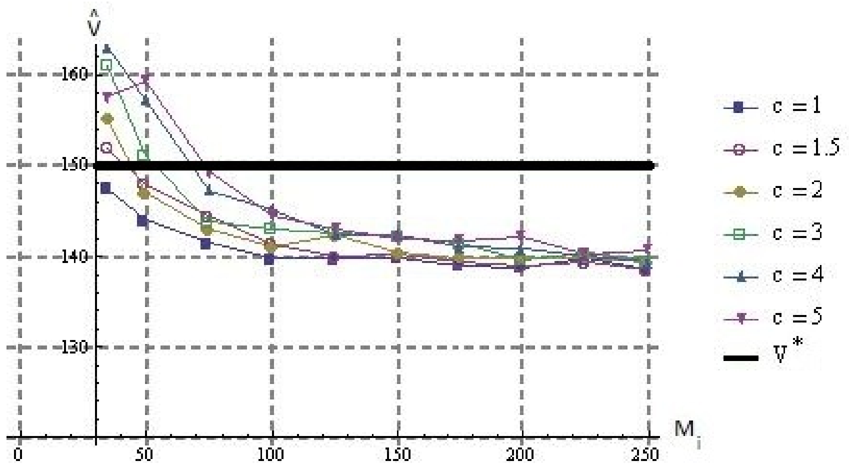


Figure 2.12: Effect of the exploration control parameters c_i on OREGA when applied to a *star* network configuration of 10 machines with machine failure probabilities $p_1 = 0.7, p_2 = 0.1, p_3 = 0.01$.

regret as does Auer's original ϵ-greedy method. We include empirical results that demonstrate how the regret can rapidly increase when deviating from the original exploration-exploitation model proposed by Auer in [13], but first we provide the theoretical analysis.

Consider a generalization to Auer's ϵ-greedy method outlined in figure 1.1, where the real-valued parameter $\alpha > 0$ is introduced so that

$$\epsilon_n^\alpha = \min\left\{1, \frac{cK}{d^2 n^\alpha}\right\}. \tag{2.41}$$

The following result asserts that for $\alpha \in [0, 1)$, it is impossible for the generalized ϵ-greedy method to achieve logarithmic regret. Although we are unable to prove it, we conjecture that the same suboptimality persists for $\alpha > 1$.

Theorem 2.7.1. *Let $K > 1$. For all reward distributions $\nu_1, ..., \nu_K$ with support in $[0, 1]$, suppose that the generalized ϵ-greedy algorithm from figure 1.1 (but using equation 2.41) is run with $\alpha \in [0, 1)$ and $0 < d \leq \min_{i:\mu_i<\mu^*} \Delta_i$. Then the probability that a suboptimal arm j is chosen after any number $n \geq (cK/d^2)^{1/\alpha}$ of plays is strictly larger than $c/(d^2 n)$. Thus, the generalized ϵ-greedy algorithm cannot achieve logarithmic regret.*

Proof. Let I_n be the arm sampled at iteration n, and consider any suboptimal arm j. For $n \geq (cK/d^2)^{1/\alpha}$, it holds that $\epsilon_n^\alpha = cK/(d^2 n^\alpha)$. Per the

MAB ID	1	2	3	4	5	6	7	8	9	10
1	0.9	0.6								
2	0.55	0.45								
3	0.9	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6
4	0.55	0.45	0.45	0.45	0.45	0.45	0.45	0.45	0.45	0.45

Table 2.5: Various MAB problems (courtesy of [13])

"randomized" exploration step outlined in figure 1.1, we have

$$P\left[I_n = j\right] \geq \frac{\epsilon_n^\alpha}{K} = \frac{c}{d^2 n^\alpha} > \frac{c}{d^2 n}.$$

The result follows by aggregating over n. □

Recall that in the closely-related theorem 1.4.1, Auer showed that the original ϵ-greedy method attains instantaneous regret $P\left[I_n = j\right]$ of order $c/(d^2 n) + o(1/n)$. Hence, Auer's algorithm is indeed superior. In fact, Auer's algorithm attains logarithmic regret which is the best one can achieve. Furthermore, recall that REGA uses a sequence $\{\epsilon_n^\alpha\}$ with $\alpha = 1/2$ which allows for more exploration that benefitted the case of estimating the value function in the MDP setting, but such choice clearly causes large regret when considering a "pure" single-stage stochastic MAB problem.

In the remainder of this section, we re-run some of the experiments carried out in [13] for $\alpha \in \{1/2, 1, 2\}$ using various values for the tuning parameter c. Table 2.5 outlines some MAB problems that appear in [13]. Each row indicates a separate MAB problem, and each column represents an arm of the MAB. Each arm i returns a binary reward in $\{0, 1\}$ following a Bernoulli distribution with success probability μ_i. The cells of the table show the values μ_i. Problems 1 and 2 each have two arms whereas problems 3 and 4 each have 10 arms. Problems 1 and 3 are easier than problems 2 and 4 since for the latter $\mu^* - \mu_i$ are smaller and the reward distributions of the optimal machines have larger variances. As in [13], each simulation was performed 100 times for 10^5 iterations. The results for each of the four MAB problems are depicted in figures 2.13 - 2.16. We observe in the graphs that $\epsilon_n^{1/2}$ performs better for smaller values of c, *i. e.*, $c = 0.15$ whereas ϵ_n^2 performs better for larger values of c, *i. e.*, $c = 6$. Moreover, a well-tuned ϵ_n^2 outperforms a well-tuned $\epsilon_n^{1/2}$. The original ϵ-greedy method of Auer (*i. e.*, ϵ_n^1) is generally best-tuned using small $c \sim 0.15$. All three variants of the algorithm, when well-tuned, perform reasonably well for the simpler MAB problems 1 and 3. The distinction in performance is more pronounced for the relatively difficult problems 2 and 4.

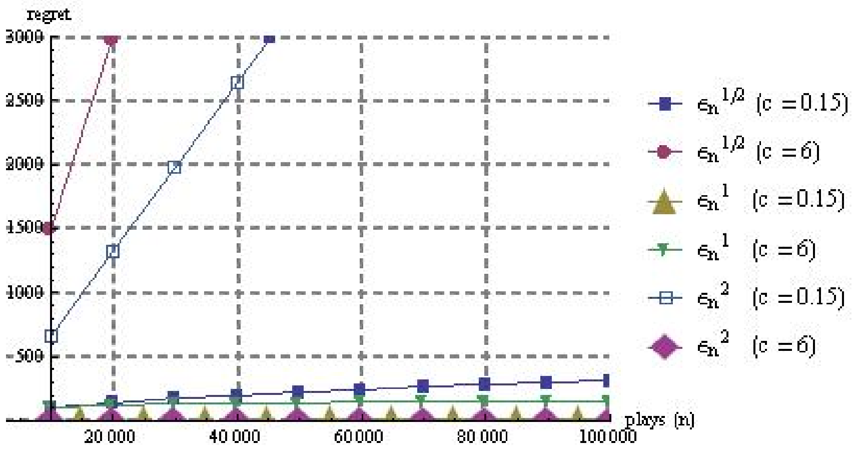


Figure 2.13: A comparison of various generalized ϵ-greedy algorithm for MAB problem #1

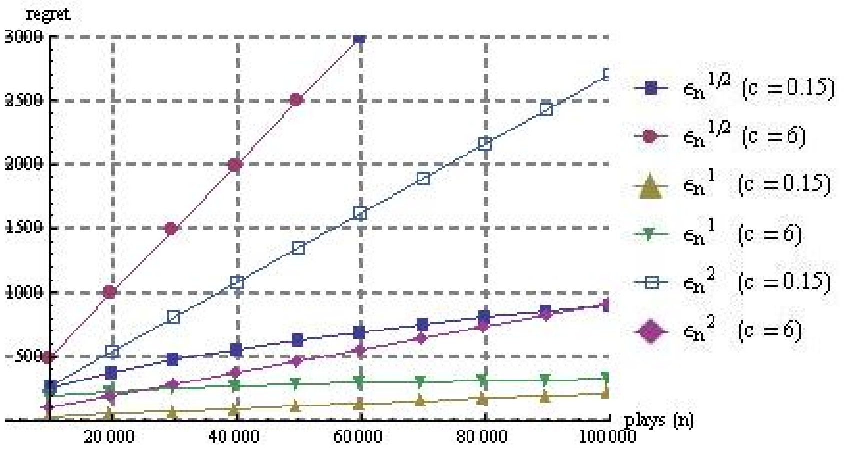


Figure 2.14: A comparison of various generalized ϵ-greedy algorithm for MAB problem #2

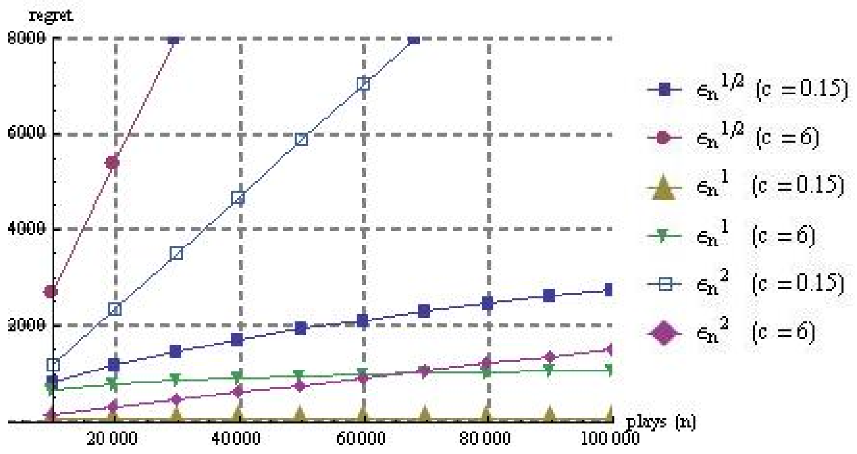


Figure 2.15: A comparison of various generalized ϵ-greedy algorithm for MAB problem #3

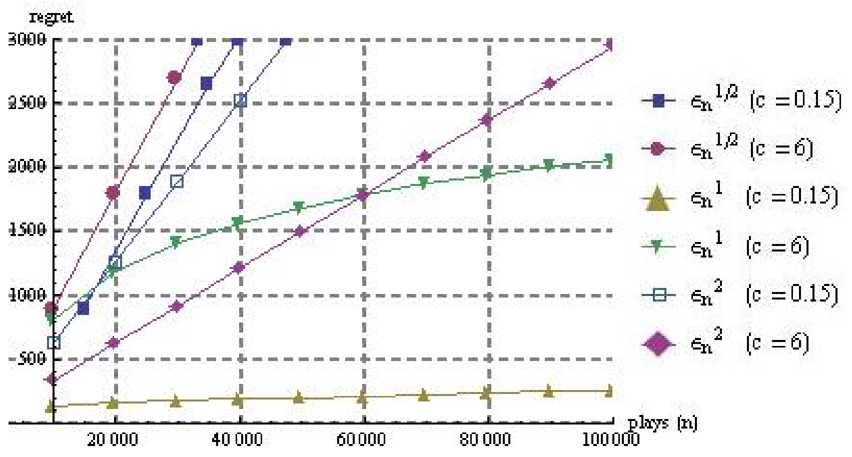


Figure 2.16: A comparison of various generalized ϵ-greedy algorithm for MAB problem #4

2.8 Concluding Remarks

We introduced REGA and its variant OREGA, recursive methods based on the well-known ϵ-greedy multi-armed bandit algorithm useful for numerically solving MDPs with (possibly large) state spaces but small action spaces. We provided a finite-time analysis of REGA and OREGA, and showed that the bounds on the errors induced by the algorithms approach zero as the number of sample points at each stage of the MDP is increased. The algorithms were tested empirically against the *SysAdmin* benchmark problem with up to 2^{10} states and compared to other closely related algorithms that appear in the literature. In our experiments, we found that a modification to the exploration/exploitation trade-off model of the ϵ-greedy method commonly cited in the literature yields improved results in practice against multi-stage MDP problems (*i. e.*, REGA *vs.* OREGA). As discussed in section 2.7, while this modification aids in obtaining better results for the adaptive sampling MDP framework, we proved that the same change will produce inferior results if applied to the classical stochastic MAB problem with objective of minimizing regret.

There remain many open problems and areas for future work. It would be interesting to construct a counterpart algorithm that is useful for MDPs with large action space and small state space. It would be of interest to carry out a formal theoretical analysis comparing, in probability, the relative performance of REGA, RASA and AMS. An open problem is to find a MAB-based algorithm that can explicitly spell out a near-optimal policy rather than only the optimal value of the MDP (*c.f.* a relevant discussion in section 2.5 of [86]). To the best of our knowledge, work along this path has not been conducted to date. Finally, AMS1 and AMS2 are known to converge to the truly optimal value of the MDP *w.p.1* (*c.f.* [75], section 3.4.1), but finding bounds on the rate of convergence of the two algorithms remains an open problem.

Chapter 3

Scheduling in the e-Discovery Domain

3.1 Introduction

In this chapter we introduce a topic of extreme significance to the legal industry, pertaining to the ability of sifting through large volumes of data in order to identify the "needle in the haystack" electronic documents related to a lawsuit or investigation. Looking at the task from a parallel computing and scheduling perspective, we highlight the main properties and challenges pertaining to the problem at hand, and propose an approach to abstract the problem into mathematical form. We hypothesize that the statistical properties of the subtasks at hand can be exploited to produce competitive schedules. We discuss an approach to the problem based on related work from the field of scheduling theory and provide simulation results that demonstrate the performance of our approach. We also provide a novel approach that weds a side multi-armed bandit to list-based scheduling. Necessarily, we propose the first MAB algorithm that accounts for both sleeping bandits and bandits with history.

When a legal investigation takes place, the involved entities are required to provide certain evidence and/or documentation as ordered by the court. For larger organizations and corporations, this often means plowing through many systems and large amounts of data in search of records related to the case. Failure to provide the mandated evidence in a timely manner may result in imposed penalties against the offending entity. Since Moore's Law has it that the amount of data in the world doubles every 20-24 months, it becomes imperative that the discovery process be automated.

Electronic Discovery, or e-Discovery for short, refers to the process of identifying, collecting, and searching electronic data with the objective of using it as evidence in a civil or criminal legal case. Locating the relevant data is an interesting problem in its own right, but is not one that we cover in this work.

Once located, the data must be harvested in a forensically-sound manner that maintains the integrity of the file contents, as well as any system-level meta data such as system dates. Proper collection techniques are covered under the field of forensics. A wide array of commercial off-the-shelf products are in existence today.

The data collected will, in general, be highly non-normalized. For example, most data sets are composed of many different file types, some files may be encrypted or password protected, a large amount of duplicated files can be present, and irrelevant system files may have not been filtered out in order to simplify the data collection process. The data need not be harvested in a single pass, as it is common to provide rolling deliveries of collected data for examination. Furthermore, data will generally be nested in layers, where a file can contain one or more attachments and embedded objects, all of which need to be captured and analyzed in a recursive manner since they can have attachments of their own.

In order to accommodate queries with different search criteria, commonly referred to in the industry as "search terms", a data processor is used to normalize and index the data into a searchable database. To shed some light on what we mean by data normalization, consider a data set containing two files: A Microsoft Word document, and a Unix e-mail message. The Word document has an "author" and a "creation date", and the e-mail message has a "sender" and a "date sent" property. The data processor will map the author of the Word document and the sender of the e-mail message into the *CREATOR* property, and the two dates are mapped into the *CREATION_DATE* property. Going forward, we refer to the component responsible for this data "massaging" as the *Data Processor*, or the *Processor* for short, which we treat as a black box. The data processor is a software program that receives supported file types as input, extracts and normalizes their contents, and then optionally outputs the results to a searchable database or filesystem. It must also discover and handle the attachments found within the files that it receives.

Indeed, multiple data processors can work in parallel to crawl the entire data set in an efficient manner. However, decomposing the data set for parallel processing is not a trivial task. In fact, the scheduling problem with arbitrary task processing times is known to be NP-hard [95]. It is hence natural to search for near-optimal scheduling policies instead. The metric of interest that we consider is the *makespan* of the schedule produced by the scheduling algorithm, that is, the total time it takes to process the entire data set at hand. We casually refer to this problem as the *e-Discovery scheduling* problem. The rest of the chapter is organized as follows: Section 3.2 outlines a formal model of the e-Discovery problem. Section 3.3 discussed a relevant POMDP approach that appears in the literature. Section 3.4 outlines some related results from the field of scheduling theory. Section 3.5 describes an online real-time algorithm based on ideas from scheduling theory. Section 3.6 provides a novel approach that incorporates a side multi-armed bandit in conjunction with the previously discussed ideas from scheduling theory. Simulation results are provided in section 3.7. Concluding remarks and directions for future work are given in section 3.8.

3.2 A Formal Model

In this section, we aim at formalizing a model that captures the e-Discovery scheduling problem. The data to be processed consists of a collection of *top level files* that exist on the file system. Each top level file can (potentially) have some attachments, each of which can have attachments of their own. We will refer to the attachments as *attached files*.

In graph theory, an out-tree is a rooted tree, *i. e.*, it has one root and each node has exactly one predecessor node in the tree. The root node naturally does not have a predecessor. An out-forest is a collection of out-trees. Figure 3.1 depicts an out-forest with three out-trees. Each node in a tree has a *level* that signifies its depth within the tree, and we adopt the notion that the level of a node is 1-based, that is, the root has level 1, the immediate child nodes of the root have level 2, *etc.* The depth of each tree is the maximum level of all (leaf) nodes in the tree.

We visualize the data set to be processed in e-Discovery as an out-forest, where each top level file is a root of an out-tree, and the attachments are descendant nodes. A directed edge exists from the root to each of its immediate attachment nodes at level 2. In turn, any attachment node at level $l \geq 2$ can have attachments of its own at level $l + 1$, and a directed edge exists from each "parent" to "child" node. The leaf nodes are files that do not have any attachments. In general, a file at level $l \geq 1$ will have zero or more children at level $l + 1$. In what follows, we denote by $\mathcal{A}(f)$ the set of attachments of file f, and by integer $N(f) = |\mathcal{A}(f)| \geq 0$ we denote the number of successors (*i. e.*, attachments) of file f.

In practice, the files encountered in e-Discovery can be of any file type, but the number of different file types is finite, albeit large, and several file types can have similar statistical properties when considering properties of interest to the modeler. We suppose that each file encountered during processing can

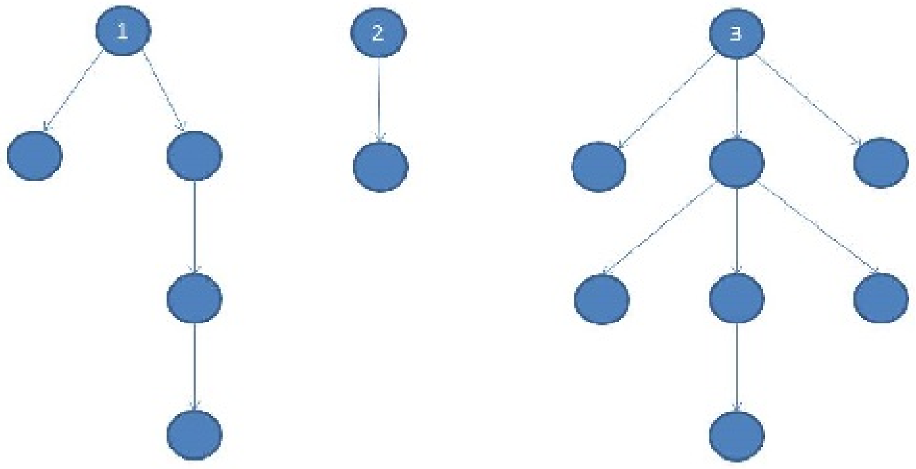


Figure 3.1: An out-forest with three out-trees.

be classified as having one of file types in $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2, \ldots\}$.

The entire data set is processed by a finite set of identical processing machines $\mathcal{M} = \{\mathcal{M}_1, \mathcal{M}_2, \ldots\}$, with the number of machines being significantly smaller than the total number of nodes in the out-forest (top level plus attachments). While processing machines can work on different files in parallel, we make a practical assumption that each machine can only process one file at a time. Within the context of the e-Discovery scheduling problem, the structure of the data tree is unknown, and file migrations involve transfers over the network, and are thus quite costly. Moreover, all processing machines are uniform in terms of their processing power, and so we avoid preemptive algorithms even though non-preemptive scheduling algorithms are known for producing schedules of lower quality, and the "good" schedules are generally too complex for implementation in practice [23]. Since we do not allow preemptions, once a machine begins processing a file it cannot be interrupted and must complete processing the file. However, the parent file and its attachments need not be processed on the same machine. Yet, we do allow reassigning to a different machine $\acute{m}$ a task previously mapped to another machine m if the latter has not yet begun working on the task. Finally, a file already processed is not permitted to be reprocessed, and so each file is processed exactly once.

At time zero, all top level files exist on some "source" storage device that is accessible to all processing machines [1], and data has not been copied yet

[1]The source storage device can be modeled as a "special" processing machine with infinite processing costs so that it is not chosen by any (reasonable) scheduling policy. The cost of copying from this machine is set appropriately per the actual cost of copying from the storage device. This allows for easier simulations and modeling of the problem.

to any of these machine. Moreover, at time zero, only the top level files are known and the attachment nodes are not known to exist yet. As progress is made, attachments are uncovered and can be processed by either the same machine that processed their parent or on different machines. The edges of each out-tree denote precedence constraints so that a parent must be processed before its attachments are uncovered and later processed. The objective is to minimize the makespan. Due to precedence constraints, the last file to be processed is necessarily a leaf node.

Before a machine can process a file, whether top level or an attachment, the machine must obtain a local copy of the file. There is always a copy cost associated with processing top level files. Ideally, the file is copied directly from the client's source storage device rather than from another processing machine, since that signifies the first time the file is copied. As for attachments, there is an (optional) copy cost depending on whether or not the attachment is processed on the same machine where its parent was processed. The general rule of thumb adopted here is that when a parent is processed on machine $m \in \mathcal{M}$, its attachments are extracted locally to the hard drive of machine m. If later on machine m processes one of the attachments and finds it to exist on its local hard drive, then a copy cost is not incurred for processing the attachment. However, if machine m no longer has a copy of the attachment locally (perhaps there was need to free space on the machine in order to accommodate other tasks), or a machine other than m processes the attachment, then a copy cost will be incurred.

We adopt a notion where copy costs are measured in terms of the time it takes to copy the file from source to destination machine, and that copy costs are proportional to the size of the files being copied. For any network, file transfer times are in general random due to external factors such as copy activities initiated by other users of the network. Suppose this randomness is captured by a (usually) small bounded random variable $\omega \geq 1$ drawn from an unknown distribution. Two costs are incurred when copying a file having size s bytes over the network: A latency b to initiate the copy request, and a streaming cost linear in s. For a network with speed of T bytes/second, the cost to copy a set of k files $\{a_1, a_2, \ldots, a_k\}$ having file sizes $s_i = s(a_i)\, \forall\, i = 1, \ldots, k$ (in bytes) is given by the function [2]

$$C(a_1, a_2, \ldots, a_k) = \omega \left(k\, b + \frac{1}{T} \sum_{i=1}^{k} s(a_i) \right). \qquad (3.1)$$

[2]For simplicity, in our experiments we set the network latency $l = 0$ and network state random multiplier factor $\omega = 1$. The file sizes of the nodes that we sample are however randomized and that captures the randomness of the problem at hand.

We now discuss processing costs. Consider an arbitrary file a with file type $ft \in \mathcal{F}$, size $s \geq 0$ bytes, and let $l \geq 1$ denote the level of the file in its out-tree. When convenient, we denote such file by the tuple $a =< ft, s, l >$. Upfront before the file is processed, its file type, file size and level in the tree are all known[3], but the same information pertaining its attachments as well as the number of attachments are unknown. A random variable $X(a)$ drawn from an unknown distribution represents the cost (per byte) of processing the file and extracting its attachments as "stand-alone" files to the local disk of the processing machine. This is when the attachments become known and added as successors to the out-tree of file a, thus becoming eligible to be processed. Recall that the set $\mathcal{A}(a)$ denotes the attachments of file a. The cost of fully handling all nodes of an out-tree with root a is denoted by $H(a)$ and can be computed recursively using

$$H(a) = s(a)\ X(a) + \sum_{n \in \mathcal{A}(a)} H(n). \tag{3.2}$$

Although the files processed in a job can be of arbitrary flavor, in practice data from the same legal matter can be generally assumed to follow the *principle of locality*, so that future data tends to be similar to historic ones previously observed. We consider four families of empirical distributions that we track over time based on observations collected throughout the lifetime of the legal case. For each of the following four families, the file type and level of the node in the out-tree define an instance of the family. That is, a different distribution is tracked for each file type and node level combination. Let $L = 10$ be the maximum attachment depth observed in most situations in practice, then the four distribution families are as follows:

Definition 3.2.1. Consider any node in a data tree. Let ft denote a file type and $l \in \{1, \ldots, L\}$ denote the level of the node, then define the following families of distributions for each combination of ft and l:

- **Probability distribution of file type ft appearing at a level l:** Denote this probability by $f(ft, l) \in [0, 1]$ such that $\sum_{\forall ft \in \mathcal{F}} f(ft, l) = 1$ for each fixed $1 \leq l \leq L$.

- **Processing cost per byte for a node at level l of file type ft:** Denote this cost by $x(ft, l)$. This cost does not include the cost of

[3]We make a simplifying assumption that the file type is known up front, whereas in reality that is not necessarily the case. One can certainly make use of the file extension or a lightweight file check command, but that can return an incorrect file type on occasion.

processing the attachments of the file, and can be used to estimate the value of X that appears in equation 3.2.

- **Number of attachments of file type ft at level l:** Denote the number of attachments by $n(ft, l)$, which can take on nonnegative integer values.
- **File size for file type ft at level l:** Denoted by $s(ft, l)$, the size of any file can take on nonnegative real values if tracked in higher units such as kilobyte, or nonnegative integers if tracked in bytes.

Since historical data for each file type and level combination need not be always available, some interpolation may be necessary in practice. Likely to be most useful is the observation that for a fixed file type ft, $n(ft, l)$ and $s(ft, l)$ are generally non-increasing in the level l. The intuition here is two fold: Theoretically, if that were not the case, then we would not expect an out-tree to be of finite size. Furthermore, a ZIP file or email archive found on the file system usually has file size and attachment count larger than a ZIP file found attached to an email document, *etc.* On the other hand, another reasonable assumption that is useful is that $x(ft, l)$ is dependent on the file type ft but independent of the level l, as the processing software need not care whether or not a file is an attachment *vs.* top level.

We conclude this section by noting that equation 3.2 can only be used when the entire data tree is known. However, since the data tree is not known a priori, an estimate on the number and type of attachments of each file can be used to compute an estimate of $H(a)$. In particular, given a node a of file type ft, size s and level l, the processing cost $X(a)$ can be computed by sampling from the distribution $x(ft, l)$ and multiplying the result by the file size s (recall x is the processing cost per byte), whereas the cost $\sum_{n \in \mathcal{A}(a)} H(n)$ of processing all attachments can be estimated by first sampling from $n(ft, l)$ to estimate the attachment count, then sampling from $f(ft, l)$ to estimate the file type of each attachment, and finally sampling $s(ft, l)$ to estimate the file size of the attachment. This can be done in a recursive manner all the way to the maximum level L. For any random variable Y, let $\widetilde{Y}$ denote a realization of Y, then the procedure can be captured by the recursion

$$\tilde{H}(ft, s, l) = s\,\tilde{x}(ft, l) + \tilde{n}(ft, l) \sum_{\forall j \in \mathcal{F}} \tilde{f}(j, l+1) \tilde{H}\left(j, \tilde{s}(j, l+1), l+1\right). \quad (3.3)$$

3.3 A POMDP Approach

Since in e-Discovery the data tree nodes are revealed in an online manner as the process evolves, any consideration of a model-based solution that incorporates the data tree into the state space of an MDP will naturally result in a POMDP model due to uncertainty. In this section, we discuss a POMDP approach that we attempted unsuccessfully.

There does not exist a dominating policy amongst the class of online scheduling policies, which makes it necessary to have knowledge of a prior distribution over task arrival patterns if one is to have any hope of determining an optimal policy [41]. For a finite horizon setting, the theory of POMDPs guarantees existence of an optimal scheduling policy for a fixed arrivals distribution, but calculating such policy is intractable in general. [41] uses stochastic planning techniques to address the scheduling problem where future tasks are unknown, and points out that prior work based on sampling techniques (*c.f.* [88, 89, 102]) are not practical due to the time required to carry out the necessary computations.

We now highlight the main results of [41], since the paper relates closely to an approach that we considered for addressing the e-Discovery scheduling problem. We do note that their work does not consider precedence constraints, which is at the core of the e-Discovery problem. In [41], tasks arriving in the future are assumed to be categorized into different classes, with each class conforming to a *hidden Markov model* (HMM). The entire scheduling problem is formally modeled as a POMDP. To that end, suppose future arriving tasks are categorized into m classes. For each $i \in \{1, ..., m\}$, let an HMM be defined by a tuple $< Q_i, T_i, \Lambda_i, \Pi_i >$, where Q is a finite set of task-generation states, T gives the probability distribution for each next state in Q, Λ maps each state in Q to probabilities of task arrivals, and Π is a distribution over Q representing the uncertainty of the initial state of the HMM. Let d be a fixed deadline, measured as a time offset since the arrival of a task. The entire scheduling problem is defined as a POMDP with state space $Q = Q_1 \times ... \times Q_m \times \{0, 1\}^{m \times d}$, where the last factor captures the ready tasks that have not been processed yet, and is called the *buffer* of each state. The POMDP action space is $\{1, ..., m\}$, meaning that a task of type i is selected for processing. State transitions are analogous to the transition probabilities of the HMMs and the state buffers. The observation space is given by $\{0, 1\}^m$, where each of the m dimensions is a binary indicator representing the observed arrival of a task from that class. The observation probabilities $p(o|s)$ are also binary $\in \{0, 1\}$.

Given the POMDP formalization, section 3 of [41] gives three basic online scheduling algorithms, whereas section 4 of their work gives a polynomial-time offline scheduling algorithm for the discounted minimum loss problem, where losses are measured in terms of missed deadlines, under a simplifying assumption that the entire list of processing tasks is known a priori. In their empirical results, it was found that sampling can reduce the weighted loss by $20-35\%$. The authors point out that an expectation maximization algorithm can be used to infer an HMM when a distribution describing future arrivals is unknown or subject to change over time [117].

We note that the e-Discovery problem can certainly be modeled as a POMDP, but we are unable to provide a model where the state space is compact enough to allow for practical usage of the model in real-time scheduling. During our investigation, we considered a few POMDP models that track the states of the data tree and the processing machines. As expected, these models were found to provide competitive schedules for very small problems involving only a handful of machines and small data trees. However, when the number of machines or number of nodes in the data tree were increased, the curse of dimensionality resulted in expensive computations due to the massive size of the state space required to capture the dynamics of the problem. Furthermore, a POMDP approach requires strong assumptions of knowledge of the involved transition distributions between the states of the underlying Markov model. Similarly, modeling the problem as a non-linear program and relaxing the constraints using a technique from [106], we found that the optimal solution was once again expensive to compute, making it impractical for real-time scheduling applications. We have thus abandoned our attempts at applying POMDPs and non-linear programs to arrive at competitive schedules for the e-Discovery setting. Our lack of success using this approach is not to be taken as indication that these approaches cannot be applied effectively to the e-Discovery problem in online fashion, and that continues to remain an open problem to consider in future work. In the remainder of this chapter, we discuss related results from the field of scheduling theory, outline a real-time online scheduling algorithm for the e-Discovery problem, and provide empirical results all based on a combination of popular scheduling approaches that appear in the literature. Our empirical results are based on large data sets.

3.4 An Overview from Scheduling Theory

In this section, we provide some relevant background from scheduling theory, some of which is used in the next section to provide a real-time scheduling algorithm for the e-Discovery problem. We note that in e-Discovery the nodes of the data tree are not known a priori, and so we are unable to use scheduling algorithms that rely on identifying and exploiting knowledge of the critical path. Recalling that an out-tree is a special case of a directed acyclic graph

(DAG) is useful in making the connection between the e-Discovery problem and some of the results that appear in the scheduling theory literature.

In *precedence-constrained* scheduling, a task cannot be scheduled until its predecessors have been processed. Tasks that can be scheduled are commonly known as *ready tasks*. A common solution to scheduling problems is *list scheduling*, where the set of ready tasks are sorted in a list/queue by some order of priority, and then the scheduler repeatedly assigns out the highest priority task to the best candidate resource/processor that can complete the task. Some of the common prioritization schemes include ordering by highest level first, longest processing time, and longest path [153]. A well-known method in the field of *project management* is the *Critical Path method* [91].All list-based scheduling using p identical processors are a $(2-1/p)$-approximation to the optimal makespan [64, 65].

Firing-squad scheduling (FS) addresses the scheduling problem from a different perspective, where a free processor always selects a task at random from a set of *enabled tasks*, which is a subset of all ready tasks. In essence, this can be viewed as a prioritization scheme where ready tasks are classified as either enabled or disabled, and all enabled tasks have equal priority. [24] studies a special case of the DAG scheduling problem with uniform-length tasks on *asynchronous processors*, where asynchrony here refers to processors of different speeds. In particular, the paper compares bounds for the makespan of two algorithms used to determine the set of enabled tasks: The $\mathbb{ALL}$ algorithm, where all ready tasks are enabled, and $\mathbb{LEVEL}$, where only tasks at the lowest level of the DAG are enabled. It is shown that asymptotically, the $\mathbb{LEVEL}$ algorithm is on-par or better than the $\mathbb{ALL}$ algorithm. To that end, let $\log^*(n)$ denote the *iterated logarithm* of n, that is, the number of times the logarithm function must be iteratively applied before the result is less than or equal to 1. Let W and D denote the total number of tasks and the longest path in the DAG, respectively, and let π_{ave} be the average speed of the p processors during execution of the DAG. [24] derived the following bounds for the makespan T_p:

- For algorithm $\mathbb{ALL}$:

$$T_p = \begin{cases} \Theta\left(\frac{D}{p\pi_{ave}}\right), & \frac{W}{D} \leq \frac{p}{\log p}, \\ \Theta\left((\log p)^{\alpha}\frac{W}{p\pi_{ave}} + (\log p)^{1-\alpha}\frac{D}{\pi_{ave}}\right), & \frac{W}{D} = p(\log p)^{1-2\alpha},\ \alpha \in [0,1], \\ \Theta\left(\frac{W}{p\pi_{ave}}\right), & \frac{W}{D} \geq p\log p. \end{cases}$$

- For algorithm $\mathbb{LEVEL}$:

$$T_p = \Theta\left(\frac{W}{p\pi_{ave}} + [\log^* p - \log^*(pD/W)]\,\frac{D}{\pi_{ave}}\right)$$

In *greedy scheduling*, processors are not allowed to stay idle if they can be assigned ready tasks. In the special case of homogeneous processors where all processors are identical, greedily scheduling the DAG on p processors without redundancies produces a schedule with makespan $T_p \leq \frac{W}{p\pi_{ave}} + \frac{D}{p\pi_{ave}}$ [36, 65], which is within factor 2 of optimal since $\frac{W}{p\pi_{ave}}$ and $\frac{D}{p\pi_{ave}}$ are both lower bounds on the best possible makespan [65]. However, for the heterogeneous case, the last term is not a bound on the optimal makespan, and so the same argument does not hold [24]. For the common case in parallel computing, it holds that $\frac{W}{D} > p$, and so a makespan dominated by $\Theta\left(\frac{W}{p}\right)$ is nearly optimal, for both cases of homogeneous and heterogeneous processors [24]. For the homogeneous setting, all greedy schedules have comparable makespans within a factor of 2 of each other [23]. Lemma 7 of [24] demonstrates that the number of time steps during which there are more than p enabled tasks is at most $O(\alpha W/p)$ with probability at least $1 - 2^{-\Theta(\alpha p + \alpha W/p)}$. More results on parallel processing in a heterogeneous setup can be found within publications from the field of *asynchronous parallel computing* (*c.f.* [17, 18, 39, 53, 90]). [23] points out that while asynchronous parallel computing assumes extreme fluctuation in processor speeds, scheduling theory considers fixed processor speeds, prior knowledge necessary for certain computations, and the scheduling authority is assumed to have global knowledge of the entire state of the system. As such, [23] provides a scheduler that bridges the gap between asynchronous parallel computing and scheduling theory.

As discussed later in this chapter, within the context of the e-Discovery scheduling problem it is reasonable to do as much as possible on a machine to which the involved files have already been copied until other machines become free and can pick up a portion of the remaining tasks. We consider a *work-stealing* (WS) approach where an under-utilized or idle processor “steals” some of the remaining tasks, typically half, from a busy processor. Earlier research on work-stealing can be found in [39, 69]. [31] showed that with p processors, the makespan of the scheduler is bounded by $O(T_1/p + T_\infty)$, where T_n is a lower bound on the execution time of the multi-threaded computation using n processors, so that T_∞ is the length of the critical path. Since T_1/p and T_∞ are both lower bounds on the makespan, this bound is within a constant factor of optimal. This scheduler is implemented as CILK, an ANSI C-based programming language for multi-threaded parallel programs. We refer the reader to [32, 120, 152] for more details on CILK. [7] generalizes WS to a non-blocking implementation with running time $O(T_1/p_A + T_\infty p/p_A)$, where p_A is the average number of processors that work in parallel to fully execute the schedule. The constant hidden in the big-Oh notation is very close to one. The authors argue that this bound is within constant factor of optimal, and achieves a linear speedup, *i. e.*, $O(T_1/p_A)$, if p_A is relatively small compared to the *relative parallelism* T_1/T_∞ of the data tree. [136] gives bounds on the number of steals within a schedule. Other important papers on WS include [131] which tackles the problem via a queueing theoretic approach, [105] which is based on a differential equations approach, and [1, 132, 133] which in some sense are biased towards locality and processor-cache affinity with tolerance for necessarily-induced load imbalance. The resultant technique is called *affinity scheduling*. [108] gives a good bibliography on the main results pertaining to WS, from which we collected some of the information outlined above.

In general, the scheduling problem with arbitrary task processing times is known to be NP-hard [95], thus making it necessary to consider heuristic techniques. Selecting the best heuristic for a particular problem from amongst those that appear in the literature is not easy due to differences in assumptions in the original studies for each heuristic [35]. At a high level, scheduling heuristics can be classified into two types: offline (static) schedules *vs.* online (dynamic/reactive) schedules. Offline scheduling requires knowledge of the problem in advance and is more efficient in the sense that heavy calculations can be computed upfront. However, such schedules cannot react to unexpected anomalies encountered at runtime, whereas online algorithms can. As [23] points out, early papers in scheduling theory approximate the schedule makespan using $O(\sqrt{p})$-approximation algorithms (*c.f.* [81, 82]), and more recent papers propose $O(\log p)$-approximation algorithms (c.f [45, 46, 50, 51]). [35] implements and compares 11 offline heuristics found in the literature, representing various flavors of iterative, non-iterative, greedy, and biologically-inspired techniques. Other techniques mentioned in the paper (see section 3.3 of [35]) include the Mapping Heuristic (MH) algorithm [58], linear programming [54], the Dynamic Level Scheduling (DLS) algorithm [126], the Levelized Min Time (LMT) algorithm [79], the Cluster-M technique [59], recursive bisection [127], neural networks [47], the k-percent best (KPB) and Sufferage heuristics [100], and the Heterogeneous Earliest-Finish-Time (HEFT) and Critical-Path-on-a-Processor (CROP) techniques [141]. Due to the uncertainty in the data being processed, offline schedulers are not ideal for the e-Discovery setting. Nonetheless, due to simplicity of implementation, we believe they are widely used in the e-Discovery industry.

Several studies have considered online schedules for the heterogeneous setting. Examples include [99], which introduced the *hybrid remapper*, a non-preemptive list-based hybrid scheduling algorithm that starts with an offline scheduling algorithm as a baseline, and then incorporates the most up-to-date state of the system at runtime. Other references on scheduling in the heterogenous setting include [7, 48, 70, 80, 94].

Some of the more recent work from scheduling theory is tailored towards advances in the IT industry. [78] gives a very good overview of *cloud computing*, as well as a simulation for dynamically deciding when to add worker machines to scale vertically in an on-demand manner, thus meeting workload requirements without letting many machines sit idle. In essence, a common theme in cloud computing is for the number of collaborating processors to be dynamically adjusted at run-time based on the amount of pending workload. On the other hand, [68] studied multiple DAGs scheduled in an *internet-computing* framework, where the performance of the scheduler is measured by the average number of tasks ready for processing at any time. A static max-min heuristic would perform well per this definition. Similarly, [99] describes two variants of the hybrid remapper method based on the intuition that executing the subtasks with larger "rank" as early as possible can reduce the expected makespan of the resultant schedule. Practical experience with the e-Discovery setting has shown that processing larger files as early as possible yields reasonable schedules and allows for discovering larger portions of the data tree early on. We revisit this heuristic in our simulations and describe our findings later on in this chapter.

3.5 A Scheduling Algorithm for e-Discovery

In this section, we combine ideas from scheduling theory discussed in section 3.4, which allow us to obtain real-time online schedules for the e-Discovery problem. In particular, we make use of list scheduling and work stealing, along with the min/max path heuristics and firing-squad scheduling heuristics.

Consider a set of identical processing machines $\mathcal{M}$ tasked with fully processing a data set TL consisting of top level files. At time zero, only the top level files are known. Each processing machine maintains a ready-queue consisting of out-trees ready to be processed, so that the ready queue represents an out-forest. Furthermore, for each machine we maintain a discrete-time event queue where the machine knows the type of events in the queue but not the time the events occur due to uncertainty with respect to the performance of the system and structure of the data being processed. The event-queue can hold events of one of three types:

StealFiles This event occurs when the ready-queue of a processing machine becomes empty. A work stealing event occurs where the machine steals unprocessed files from another machine, chosen based on the heaviest estimated remaining workload. Files begin to be copied immediately and a ***FileCopyCompleted*** event (which we define momentarily) is pushed onto the event queue, to be completed in the future per the copy costs of equation 3.1. We adopt the notion that if multiple machines steal files at the same exact moment, then they all steal from the machine with the heaviest load. In particular, let W_m denote the total workload in the ready-queue of each machine $m \in \mathcal{M}$. If j machines steal work at any moment from a machine $z = \underset{m \in \mathcal{M}}{\operatorname{argmax}} W_m$, then the objective is for each machine to end up with a workload of roughly $W_z/(j+1)$, with an equal portion of that kept on the source machine z. Therefore, if only one machine steals work, then it aims at stealing half the remaining workload of the victim machine z. A special case of work-stealing occurs at the beginning of the processing job at time zero, where all processing machines $\mathcal{M}$ steal roughly an equal portion of the TL top level files that are to be processed from the source storage device. Arguably, the most inefficient limitation of our proposed model is a simplifying assumption that work stealing and file processing cannot occur in parallel on different threads. Furthermore, each machine can be engaged in at most one ***StealFiles*** event at any time.

FileCopyCompleted This event occurs in the future as a result of initiating a ***StealFiles*** event at some point in the past. Its time of occurrence is controlled by equation 3.1.

FileProcessingCompleted This event occurs when a processing machine completes processing a single node a and extracting all its attachments to the machine's local file system. If the machine begins processing the file at time t_1, then this event occurs at time $t1 + s(a)\ X(a)$, where $s(a)$ and $X(a)$ are as given in equation 3.2.

The sequence of events on any given processing machine is as follows: At time zero, the machine's ready-queue is empty and so a ***StealFiles*** event occurs where the machine begins to copy an (ideally equal) portion of files from the TL top level files that currently only exist on a client's external storage device. Portions are divided by giving each file a weight according to some stealing heuristic $\mathcal{H}^s$. In practice, as we discuss in our simulation results, a heuristic that performs well gives each top level file a weight proportional to the total copy cost plus processing costs of the file and all its descendants. That is, each file a is given a weight proportional to an estimate of $H(a) + C(a)$.

By convention, we sort the files according to their associated weights then round-robin through the list and divide the files amongst the machines that are to share these files. That is, the first file in the sorted list is assigned to the first machine, the second file to the second machine, *etc.* When the last machine is served its first file, the first machine is served its second file, and so on. The intuition here is to give each machine some "heavy" tasks as well as some "light" tasks in hopes of minimizing the need to re-balancing the workload at a future time step which will incur additional work-stealing costs. A ***StealFiles*** event is always followed by a ***FileCopyCompleted*** event. Until the ***FileCopyCompleted*** event occurs, the processing machine remains busy purely copying files without any files being processed on that machine, as we assume that all copied files arrive at once. A file that is currently being processed cannot be stolen by another machine since we do not allow preemptions of started tasks. Furthermore, if a machine has one file remaining to be processed but processing has not begun yet, then that file cannot be stolen as it would introduce an unnecessary (and inefficient) copy cost, only to be processed on an identical machine. Finally, while a file is flagged as in the process of being stolen (*i. e.*, a copy request is in progress), it cannot be stolen by any other machine nor processed by the machine from which the file is stolen. This ensures that a file is never processed twice which is a requirement that we impose in our setting.

The occurrence of a ***FileCopyCompleted*** event signifies that the ready-queue is no longer empty. The machine now can and will immediately begin to process all files in the ready-queue per the scheduling policy. We restrict our attention to list schedules using heuristics that we will discuss later. The heuristic used to select the next file to process, denoted $\mathcal{H}^p$, need not be the same as the heuristic $\mathcal{H}^s$ used for work-stealing. The processing events are tracked in the event-queue via a series of ***FileProcessingCompleted*** events. Eventually, the ready-queue becomes empty and a ***StealFiles*** event needs to occur so that the machine is assigned more work from another machine. This sequence of events continues to occur until all files, including their attachments, have been processed. The objective is to minimize the makespan, which is the time the last ***FileProcessingCompleted*** event occurs across the event queues of all the processing machines. The sequence of events is depicted in figure 3.2.

Each machine has a ready-queue of files that are eligible to be processed. Since the ready files have not been processed yet, the attachments that they contain, if any, are unknown. Once a ready file a with level l is processed, the processing cost $s(a)\ X(a)$ in incurred (in time units) and the immediate attachments at level $l+1$ become known, although the lower-level attachments

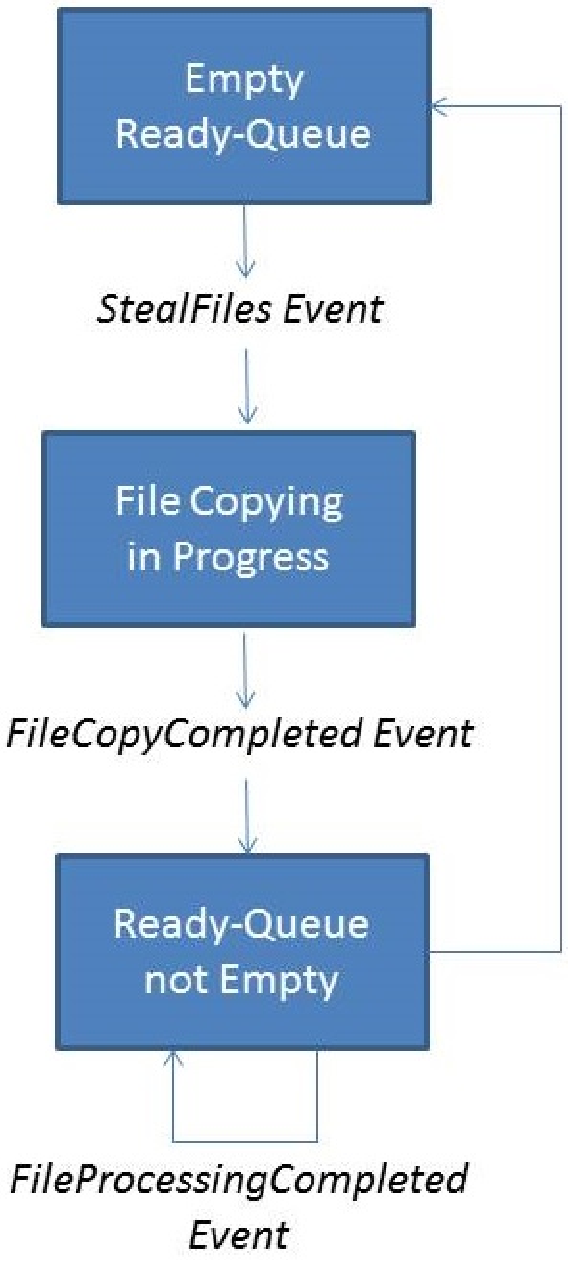


Figure 3.2: The sequence of discrete-time events that can occur on each processing machine during the e-Discovery workflow.

at level $l + 2$ (and beyond) are not known yet, since only one level of the out-tree is discovered at a time. As soon as the attachments of a processed file are known, we remove the root node (*i. e.*, processed file) from the out-tree, thus creating smaller out-trees which we add to the ready-queue, one for each discovered attachments. Following this notion, the root of an out-tree in the ready-queue always corresponds to a file that is ready for processing. Given this implementation, we are able to liberally speak of ready files and ready out-trees interchangeably. Figure 3.3 demonstrates the evolution of the ready queue as nodes 1, 2 and 3 are processed, in order. The solid circles denote files that have been uncovered, whereas the hollow nodes denote files attachments that are unknown at that time.

Under expectations that the *principle of locality* holds, we assume that historical data provide a good starting point for estimating statistical properties of data arrivals in the future. As such, at time zero the scheduling policy starts with the distributions outlined in definition 3.2.1 initialized based on historical data. As the scheduling policy progresses over time, sample statistics observed are used to update these distributions. We track one set of global statistics that are shared by all processing machines rather than a separate set of statistics per machine. The algorithm that we propose for e-Discovery scheduling is outlined in figure 3.4, where we use the following added notation: $\mathbb{1}\{g\}$ is the indicator function so that $\mathbb{1}\{g\} = 1$ if condition g holds, and $\mathbb{1}\{g\} = 0$ otherwise. RQ_m denotes the ready-queue of machine $m \in \mathcal{M}$. For an ordered list J, $\sigma(j)$ denotes the j^{th} element where $j = 0, 1, 2, \ldots$ is zero-based. We emphasize that in our algorithm we do not account for copy costs when determining the victim machine from which a steal is to occur. However, once a victim machine is determined, balancing the amount of work to steal takes into account the copy costs, but only for the stealing machines and not for the victim machine (recall a portion of works remains assigned to the victim machine). That is, with five identical files and only one stealing machine, the stealing machine picks up one or two files and the victim machine keeps the rest since the latter do not incur copy costs.

We conclude this section with a brief discussion of the work-stealing heuristic $\mathcal{H}^s$ and the $\mathcal{H}^p$ heuristic used to select the next file to process. Given an out-forest A consisting of out-trees a_i ready to be processed, consider the maximum estimated total tree process time $\max\limits_{a_i \in A} \tilde{H}(a_i)$. For $\mathcal{H}^s$, we believe that $\max\limits_{a_i \in A} \tilde{H}(a_i) + C(a_i)$ is the most logical choices since the aim is to balance the workload on all available processing machines in order to minimize the makespan. To see this, suppose that $H(a_i)$ can be computed precisely in a full-information setting, and suppose that we have two processing machines and total workload W to be split between the two machines. Suppose

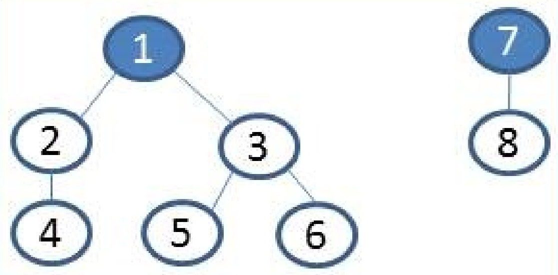
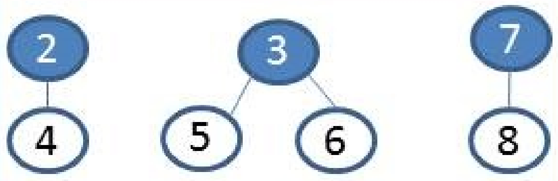
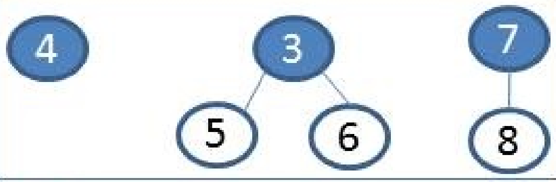
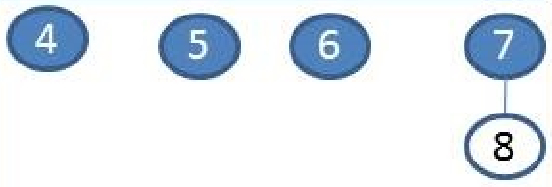
Time & Description	Ready-Queue
• Initial state at time t. • File 1 processing begins.	
• At time t + s(1) X(1). • File 1 processing completes. • Nodes 2 and 3 become ready. • File 2 processing begins.	
• At time t + s(1) X(1) + s(2) X(2). • File 2 processing completes. • Node 4 becomes ready. • File 3 processing begins.	
• At time t + s(1) X(1) • + s(2) X(2) + s(3) X(3). • File 3 processing completes. • Nodes 5 and 6 become ready. • Node 7 begins processing.	

Figure 3.3: The evolution of a ready queue for a processing machine as nodes 1, 2 and 3 are processed, in order. Solid nodes denote files that have been uncovered, whereas hollow nodes denote attachment files that are unknown at that time.

```
The e-Discovery Scheduling Algorithm
Initialization: $M \geq 2$ processing machines. Max node level $L$. $\forall$ file types $ft \in \mathcal{F}$ and level $1 \leq l \leq L$, initialize $f(ft,l)$, $x(ft,l)$, $n(ft,l)$ and $s(ft,l)$ outlined in definition 3.2.1. Specify heuristics $\mathcal{H}^s$ and $\mathcal{H}^p$. $RQ_m$ are empty $\forall$ machines $m$.
Loop (until all files are processed)
 Parallel.Foreach machine $1 \leq m \leq M$
   1. Work-Stealing step:
      (a) Determine machine $m_0 = \operatorname{argmax}_{i \in \mathcal{M}} \sum_{a \in RQ_i} \mathcal{H}^s(a)$ having heaviest workload (or use client hard drive at start of algorithm).
      (b) Assume machines $m_1, \ldots, m_y$ arrive here simultaneously ($y \geq 1$). Reorder $a \in RQ_{m_0}$ per $\mathcal{H}^s(a) + C(a)$ in non-increasing order.
      (c) Split the load: $\forall z = 0, \ldots y$, assign one file $\sigma(z) \in RQ_{m_0}$ to each machine $m_z$ and update $SW_z = \mathcal{H}^s(\sigma(z)) + \mathbb{1}\{z \neq m_0\}C(\sigma(z))$.
      (d) $\forall a \in RQ_{m_0}$ that remain, repeatedly assign the next file $a$ to machine $z^* = \min SW_z$ and update $SW_{z^*} = SW_{z^*} + \mathcal{H}^s(a) + \mathbb{1}\{z^* \neq m_0\}C(a)$.
      (e) Block until file copying completes, incurring copy costs per equation 3.1.
      (f) If stole some files, goto "Process Step", else Sleep one time-step then goto "Work-Stealing step".
   2. Process Step: While ready queue not empty
      (a) Select next file $f_m \in RQ_m$ to process per $\mathcal{H}^p(f_m)$.
      (b) Process $f_m$ having file type $ft$ and level $l$. Update $x(ft,l)$ per incurred cost per byte $X(f_m)$.
      (c) Remove $f_m$ from $RQ_m$. Add discovered attachments $\mathcal{A}(ft_m)$ to $RQ_m$. Update $n(ft,l)$, $f(ft,l)$ and $s(ft,l)$.
      end
   3. Update "local" makespan of this machine $LM_m$ = current time.
   4. Goto "Work-Stealing step".
 end
end
Return: Makespan $\max_{m \in \{1,\ldots,M\}} LM_m$.
```

Figure 3.4: The e-Discovery scheduling algorithm

that the first machine is allotted workload $W_1 = W/2 + \delta$, where $\delta \geq 0$, and the second machine is allotted the remaining workload $W_2 = W - W_1$. The makespan is given by $\max(W_1, W_2)$ and is minimized when $\delta = 0$, *i. e.*, when the two machines are given an equal workload. As for $\mathcal{H}^p$, we know from prior practical experience with the e-Discovery problem that the $\max\limits_{a_i \in A} \tilde{H}(a_i)$ heuristic produces reasonable schedules. Copy costs are ignored in the case of $\mathcal{H}^p$ since the ready files are made available locally on the machine before they are considered ready for processing[4].

Since one of the main challenges in the e-Discovery setting is due to uncertainty in the data tree sub-nodes that are yet to be uncovered as the process evolves, it's reasonable to consider for $\mathcal{H}^p$ a heuristic $\max\limits_{a_i \in A} \tilde{N}(a_i)$ that schedules next the node that yields the most information about the yet unknown child nodes (attachments). Recall that $N(a)$ denotes the number of immediate children of node a, not to be confused with the total number of sub-nodes in the data tree under a. Using the total number of sub-nodes instead of the immediate child-node count can result in expensive computation of the heuristic, which in turn can degrade the performance of the scheduling algorithm thus leaving it unusable for real-time scheduling problems. We thus do not consider the latter in our study.

We conclude by noting that these heuristics can be combined where one heuristic is used for initial sorting and then another heuristic is used to break ties. Heuristics can also be combined in a weighted manner depending on the problem setting. We simulate the performance of several heuristics in section 3.7.

3.6 List Scheduling With Side Bandits

In this section, we demonstrate a novel approach where bandits can be used in lieu of heuristics to improve the quality of online list-based scheduling policies. The general idea is that in scheduling problems with incomplete information, when the heuristic of choice is itself a random variable due to uncertainly, the quality of a list-based scheduling policy can be potentially improved over time so that later iterations of the schedule can take advantage of earlier ob-

[4]For work-stealing, $\min\limits_{a_i \in A} \tilde{H}(a_i) + C(a_i)$ and $\max\limits_{a_i \in A} \tilde{H}(a_i) + C(a_i)$ produce identical steal schedules but with the files assigned to the stealing machines in opposite order. Thus, the makespan is not affected if either is used. However, selecting the next file for processing is not symmetric with respect to these two heuristics since the processing schedule can be interrupted with a steal event from another machine. See our simulation results in section 3.7.

servations. When the heuristic can be computed in deterministic fashion, the choice is to schedule next the action that optimizes the heuristic. However, when only an estimate of the heuristic is available, an opportunity presents itself to consider a trade-off between selecting an action that optimizes the heuristic (exploitation) *vs.* selecting another action to explore its effect on the heuristic (exploration). This concept sounds all too familiar and can be achieved by incorporating a side bandit model that is used by the scheduling policy in order to determine the next action to take. We refer to this approach as ***list scheduling with side bandits***[5]. There is a wide array of publications in the scheduling domain where bandits are used, but to the best of our knowledge our work is the first to replace a heuristic with a side bandit model to improve the quality of list-based scheduling policies over time.

Let A_t denote the finite set of actions available to the scheduling policy at time t. Suppose that each time action $a_i \in A_t$ is played, it returns a heuristic $\mathcal{H}_t(a_i)$. Furthermore, suppose that for each fixed a_i, $\mathcal{H}_t(a_i)$ are i.i.d random variables having distribution $Y_t(a_i)$, where $t \geq 0$. Furthermore, with respect to actions a_i, $Y_t(a_i)$ are also independent. That is, samples drawn from Y are independent with respect to both t and a_i. For simplicity, we restrict our discussion to the setting in which the heuristics are stationary with respect to the time t, and thus drop the suffix and denote $Y(a_i) = Y_t(a_i) \, \forall \, t \geq 0$.

Without a side bandit, at each iteration of the algorithm, a conventional list-based scheduling policy in an incomplete-information setting would list any available action a per the sample mean of its respective heuristics $\overline{Y}(a)$. Such policy is demonstrated in figure 3.5. Aiming to minimize the regret due to randomness in the heuristic associated with a particular action, we integrate a multi-armed bandit into the algorithm. Since not all actions are available to the scheduling policy at decision-making times, we adopt a *sleeping bandit* algorithm [6]. In particular, we consider the AUER algorithm of [92]. Since historical information can also be available to the scheduling policy from previous runs, we also incorporate results pertaining to bandits with history. In particular, we chose the HUCB1 algorithm of [124]. The sleeping and history bandits are fused together, and the resultant list-based scheduling algorithm with side bandit is outlined in figure 3.6[7].

[5]For the case of complete information, *best experts* are a natural choice instead of multi-arm bandits; *c.f.* [92] which also provides a theoretical treatment of the subject.

[6]Recall that in a sleeping bandit not all arms are available at each iteration of the algorithm. See section 1.4.3.

[7]Both the AUER and HUCB1 algorithms are adapted versions of the UCB1 method of [13]. Combining the two algorithms as done in figure 3.6 results in a generalized MAB algorithm which we believe is the first to appear in the literature for the setting of sleeping bandits with history. Since the underlying algorithms are each known to achieve logarithmic

```
**A Standard List-Based Scheduling Policy**
for each time step $t = 0, 1, \ldots$ do
    1. Determine the set of available actions $A_t$.
    2. Schedule action $a_t = \operatorname*{argmax}_{a \in A_t} \overline{Y}(a)$ that optimizes the heuristic $\overline{Y}$.
end
```

Figure 3.5: A standard list-based scheduling policy

We now turn our discussion to the e-Discovery scheduling problem. Several properties of our setting make the inclusion of a side multi-armed bandit particularly attractive. More specifically, all nodes of the data tree are to be eventually visited by the scheduling policy, and we do not allow files to be processed more than once[8]. Thus, incorporating a side bandit into the scheduling policy simply re-orders the way in which the nodes are processed. Moreover, the setting of synchronous (*i. e.*, identical) processors allows for one central side multi-armed bandit that can share all observations obtained from all processing machines. In the case of asynchronous (*i. e.*, non-identical) processing machines, each machine would have to maintain its own side multi-armed bandit which is less cooperative and is subject to higher estimation errors due to reduced sample counts for each bandit arm. Additionally, tracking multiple side bandits will increase the overhead of the scheduling policy in terms of memory and CPU usage. We are fortunate to avoid such issues in our setting.

In practice, not all heuristics fit the side bandit model well. For example, consider the e-Discovery data tree model, and suppose that the total subtree processing heuristic, $\max H(a)$, is used. Per equation 3.2, $H(a)$ must be computed recursively, and thus a relatively-expensive bandit similar in some sense to the REGA method will be necessary unless a myopic or rolling-horizon approach is taken to estimate $H(a)$ (*c.f.* sections 2.3 and 2.5 for a relevant discussion). This becomes particularly problematic if the data tree contains very deep attachments since the side bandit will generally be too expensive to compute in real-time due to the curse of dimensionality associated with the backwards induction process.

On the other hand, we discussed in section 3.5 a simple and interesting heuristic based on the number of immediate attachments $N(a)$ which fits the

regret, we conjecture that the resultant combination can also be shown to have logarithmic regret. We leave that for future work.

[8]The opposite of scheduling each task once is known in the scheduling theory literature as *eager scheduling*, where a task is allowed to be processed multiple times in parallel by several processors.

```
**A List-Based Scheduling Policy with a Side Bandit**
**Input**: Current time $t$. List of available actions $A_t$. For each action
$a^i \in \bigcup_{t=0,1,\ldots} A_t$, historical averages $z_h^i$ and number of simulations $n_h^i$ are
provided. New observation averages $z^i = 0$ and counts $n^i = 0$ are initialized
to zero.

for each time step $t = 0, 1, \ldots$ do
    1. Determine the set of available actions $A_t$.
    2. if $\exists\, a^j \in A_t$ s.t. $n^j = 0$ then choose $a_t = a^j$
       else choose action $a_t = \operatorname{argmax}_{a^j \in A_t} \left( \frac{n^i z^i + n_h^i z_h^i}{n^i + n_h^i} + \sqrt{\frac{8 \log(t + n_h^i)}{n^i + n_h^i}} \right)$
       end
    3. Schedule action $a_t$ and observe heuristic $r^{a_t} \sim Y(a_t)$.
    4. Update $z^{a_t} = z^{a_t} + r^{a_t}$.
    5. Update $n^{a_t} = n^{a_t} + 1$.
end

for each available action $a^i \in \bigcup_{t=0,1,\ldots} A_t$, update histories for future use:
    1. $z_h^i = \frac{n^i z^i + n_h^i z_h^i}{n^i + n_h^i}$.
    2. Update $n_h^i = n_h^i + n^i$.
end
```

Figure 3.6: A list-based scheduling policy with a side bandit

side bandit approach well. In particular, the side multi-armed bandit is used to select the next action (*i. e.*, file) $a \in A_t$, and the file is then processed and the actual attachments count $N(a)$ is discovered and represents the side bandit's one-step reward associated with processing files from the same class as a. Computation of the bandit reward in this case is obtained, essentially for free, as a byproduct of file a being processed and its content discovered, which allows for such heuristic to be used in real-time scheduling problems without much added computational overhead. The max $N(a)$ heuristic is particularly interesting in our setting where the data tree is discovered in an online fashion, and can be interpreted as being the “added value” of learning more about the data tree. We thus only consider $\max N(a)$ and not $\min N(a)$. We refer to the bandit version of $\max N(a)$ as the *side bandit attachments count* (SBAC) heuristic. We discuss the empirical performance of SBAC and the other heuristics in section 3.7.

3.7 Empirical Results

We simulate the e-Discovery scheduling algorithm outlined in figure 3.4 against two randomly-generated benchmark problems with 122537 and 5824813 files in total, including attachment. We refer to these as problems *eDisc1* and *eDisc2*, respectively, with problem *eDisc2* being the larger of the two. We classify each file in the datasets as being one of four types: Email archive, Zip file, regular file, and files that cannot contain attachments. All but the last type can contain zero or more attachments. Email archives can contain immediate attachments that are email messages which we include in the regular file category, or contact information items which cannot contain any attachments, but email archives cannot contain Zip files as immediate attachments. Zip files can contain all other types, including other zip files. All file types can appear as top level or attachments within the data tree. *eDisc1* and *eDisc2* have maximum node depths six and seven, respectively. Tables 3.1 and 3.2 outline the file type counts for each level of the out-tree of *eDisc1* and *eDisc2*, respectively. The total file size of the two problems is 5.4 GB top level (17.1 GB total) and 162 GB top level (458 GB total), respectively[9].

As samples are collected throughout our simulation, we update the distributions outlined in definition 3.2.1. Rather than track actual families of distributions, we track point-based averages for each file type and tree node level encountered at runtime. The statistics are initialized per table 3.3 to simulate historical information collected from previous processing jobs. We assume that the statistics were computed based on 100 historical sample points for each combination of file type and tree node level. When meaningful and consistent with general trends encountered in real-world data, certain statistics are set to be decreasing in the level of the node. In particular, deeper attachments are smaller and contain less attachments on average. We adopt the notation $Y^+ := \max(0, Y)$ for easier readability.

We simulated processing of the benchmark problems using 5, 50 and 100

[9] *eDisc2* is not uncommon in the field, but is certainly a “respectable” amount of data to handle. Many data sets of size similar to *eDisc1* are encountered on a daily basis in the field.

Level	Email Archive	Zip File	Regular File	No Attachments File
1	0	12	0	0
2	0	0	9822	4725
3	7	4	61483	2557
4	0	0	28608	4172
5	0	0	7084	3643
6	0	0	410	10

Table 3.1: File type counts of e-Discovery benchmark problem *eDisc1* at each level of the out-forest (122537 files in total).

Level	Email Archive	Zip File	Regular File	No Attachments File
1	21	32	392326	176564
2	179	177	1266812	556829
3	59	69	1865764	720240
4	0	0	540440	100218
5	0	0	101473	70755
6	0	0	15860	14336
7	0	0	713	1946

Table 3.2: File type counts of e-Discovery benchmark problem *eDisc2* at each level of the out-forest (5824813 files in total).

Level	Stats	Email Archive	Zip File	Regular File	No Attachments File
1	$f(ft,l)$	0.35	0.35	0.20	0.10
	$x(ft,l)$	2×10^{-6}	2×10^{-6}	2×10^{-6}	2×10^{-6}
	$n(ft,l)$	2000	500	6	0
	$s(ft,l)$	3GB	200 MB	200 KB	1 KB
2	$f(ft,l)$	0.01	0.01	0.68	0.30
	$x(ft,l)$	2×10^{-6}	2×10^{-6}	2×10^{-6}	2×10^{-6}
	$n(ft,l)$	95	300	5	0
	$s(ft,l)$	100MB	9 MB	200 KB	1 KB
≥ 3	$f(ft,l)$	10^{-6}	10^{-6}	$0.999998 \times 2.5/l$	$0.999998 \times (l-2.5)/l$
	$x(ft,l)$	2×10^{-6}	2×10^{-6}	2×10^{-6}	2×10^{-6}
	$n(ft,l)$	$(100-5l)^+$	$(20-l)^+$	$(6-l)^+$	0
	$s(ft,l)$	100MB	$(10-l)^+$ MB	200 KB	1 KB

Table 3.3: Point-based (mean) initial statistics for the e-Discovery benchmark problem, assuming 100 sample points from historical processing jobs.

processing machines. For each simulation, two file transfer speeds T are considered: 10 MB/s and 100 MB/s[10]. Treating the statistics outlined in definition 3.2.1 as point-based (averages), we sample each statistic with current average $\overline{u}$ from a uniform distribution $U(0, 2\,\overline{u})$ when in need of a sample point for a statistic.

For work stealing, the workload of each machine $m \in \mathcal{M}$ is estimated via the total work of all files available in its ready queue RQ_m (plus copy costs), so that the heuristic $\mathcal{H}^s$ is given by $\mathcal{H}^s = \sum\limits_{a \in RQ_m} \tilde{H}(a) + C(a)$, where C and $\tilde{H}$ are given in equations 3.1 and 3.3, respectively. We refer to this as a work-stealing setting with incomplete information due to the lack of full information of the actual costs $H(a)$. For sake of completeness, we redo all simulations and consider work-stealing under the complete information setting where $\tilde{H}(a)$ is replaced with $H(a)$ and $\mathcal{H}^s$ is given by $\mathcal{H}^s(a) = \sum\limits_{a \in RQ_m} H(a) + C(a)$. This provides a more controlled setting to compare the performance of the various scheduling heuristics $\mathcal{H}^p$ used.

We consider several heuristics $\mathcal{H}^p$ for selecting the next file a to process from each machine's ready queue RQ_m: Minimum estimated total subtree processing cost $\min\limits_{a \in RQ_m} \tilde{H}(a)$, maximum estimated total subtree process-

[10]For sake of comparison, a gigabit ethernet connection has speeds of up to 125 MB/s.

ing cost $\max_{a \in RQ_m} \tilde{H}(a)$, maximum estimated number of immediate attachments $\max_{a \in RQ_m} \tilde{N}(a)$, as well as the two firing-squad (FS) scheduling methods $\mathbb{ALL}$ and $\mathbb{LEVEL}$ described in section 3.4.

We also consider what we call the *side bandit attachments count* (SBAC) extension of the $\max \tilde{N}(a)$ heuristic, as outlined in figure 3.6. With four file types $ft \in \mathcal{F}$ and up to ten levels $l \in \{1, \ldots, L = 10\}$ in each data tree, we define $|\mathcal{F}| \times L = 40$ bandit arms where each arm is a filetype and node level pair (ft, l).

Moreover, although unrealistic, for sake of comparison we also simulate the case where $\mathcal{H}^p$ selects the next file to process from each machine's ready-queue based on the (true) maximum total subtree processing cost $\max H(a)$ and maximum number of immediate attachments $\max N(a)$ under the assumption of complete information. When reporting our simulation results, we list these at the bottom of tables 3.4 - 3.7. Furthermore, we ignore these unrealistic settings in our discussions of how the various heuristics compare to one another based on the simulation results.

Each simulation setting was run 10 times and the results for the *eDisc1* and *eDisc2* benchmark problems with complete and incomplete work-stealing settings are outlined in tables 3.4 - 3.7. Each cell in the tables contains an entry in the form of "$X(Y) < Z >$", where X is the resultant average makespan (in seconds), Y is the standard error of the sample population, and Z is the average total file size moved around due to steals (in megabyte) from the beginning to the end of the entire schedule.

Plots of total work-steals over time (*i. e.*, aggregated file sizes stolen up to current time) for the *eDisc2* problem in the complete and incomplete work-stealing settings are given in figures 3.7 and 3.8, respectively. "Good" schedules would have a single steal at time zero in order to split up the total work across all machines, which in turn would keep machines busy till the very end where one would expect to see a second (small) spike of steals towards the end of the schedule as machines begin to complete the tasks initially assigned to them.

For the larger of the two problems, *eDisc2*, it is easier to spot relative differences in the performance of the algorithms. As such, in tables 3.6 and 3.7 we highlight for each simulation setting (column) the heuristic (row) that performs best, with the performance being measured by both the makespan and total work-steals. The two metrics are observed independently, and so for any column one heuristic can have the best average makespan while another heuristic can have the best (least) amount of average generated network traffic due to work-stealing.

The results of our simulations suggest that faster file copy speeds T and a larger number of machines M both independently reduce the makespan of the produced schedule. We also surprisingly find that increasing T or M will in general independently reduce the total bytes transferred over the network, *i. e.*, less steals, which can be important in busy or non-dedicated networks. An interesting observation is that the standard error of the makespan is decreased as the network speed T is increased. A similar correlation between M and the sample standard error cannot be verified from our experiments. In general, a lower variance is desirable, as it increases our confidence in the "predictability" of the performance of the scheduling policy. This is particularly important in the e-Discovery setting where one has a "single shot" at processing the data set and there is little to no benefit from processing the data more than once.

A slow network can easily become the bottleneck in the performance of the scheduling policy. This is clear when observing in table 3.7 that none of the heuristics show much improvement in the makespan of *eDisc2* for a slow network of 10 MB/s when doubling the number of machine from 50 to 100 machines. The decreased amount of steals, however, continues to hold due to the increase in the number of machines.

We observe that for the smaller *eDisc1* problem, the performance of all heuristics is roughly similar in terms of makespan. However surprisingly, the $\min \tilde{H}(a)$ heuristic results in the least amount of network traffic (*i. e.*, steals). As for the much larger *eDisc2* problem, the FS $\mathbb{LEVEL}$, $\max \tilde{H}(a)$ and SBAC algorithms result in the best makespans, in general. SBAC performed best overall compared to the other algorithms with respect to total work-steals. As expected, SBAC's incorporation of a side bandit resulted in significant improvements over its $\max \tilde{N}(a)$ counterpart, both in terms of the makespan as well as total work-steals. Somewhat expected due to the theoretical guarantees that appear in [24], firing-squad $\mathbb{LEVEL}$ generally outperformed $\mathbb{ALL}$, even in a finite-time (non-asymptotic) sense. In terms of worst performance, $\max \tilde{N}(a)$ and $\min \tilde{H}(a)$ generated the most amount of network traffic. The two algorithms produced in several settings the worst average makespan, especially for the setting with a small number of machines $M = 5$.

In terms of runtime performance of the three heuristics that performed best overall, $\max \tilde{H}(a)$ is rather expensive to estimate for lower-level (shallower) nodes due to the recursion involved in its estimation (*c.f.* equation 3.3), and the cost of computing this heuristic becomes cheaper as the level of the root of the subtree increases, thus resulting in a shallower subtree to estimate. On the other hand, SBAC and FS $\mathbb{LEVEL}$ have computational cost that is independent of the nodes' level in the data tree. While SBAC accounts for anomalies outside of expected performance based on historical data, FS $\mathbb{LEVEL}$ may not handle anomalies as well.

$\mathcal{H}^p$	$M = 5$		$M = 50$		$M = 100$	
	$T = 10$	$T = 100$	$T = 10$	$T = 100$	$T = 10$	$T = 100$
$\min \tilde{H}(a)$	69940(34.60) $< 19707 >$	63733(81.60) $< 17752 >$	48018(128.34) $< 6908 >$	41244(55.15) $< 6800 >$	47980(94.86) $< 6037 >$	40260(35.46) $< 5898 >$
$\max \tilde{H}(a)$	69844(31.52) $< 22267 >$	63407(3.03) $< 24599 >$	47705(90.68) $< 6882 >$	41255(101.21) $< 6693 >$	47968(74.59) $< 5975 >$	40246(12.76) $< 5875 >$
$\max \tilde{N}(a)$	70044(65.02) $< 28943 >$	63603(64.36) $< 23700 >$	47944(80.53) $< 6969 >$	41137(29.16) $< 6861 >$	47798(55.35) $< 5992 >$	40232(7.81) $< 5885 >$
FS $\mathbb{ALL}$	69889(37.36) $< 26876 >$	63555(47.16) $< 20893 >$	47895(78.21) $< 6847 >$	41177(71.11) $< 6724 >$	47878(59.22) $< 5987 >$	40240(9.62) $< 5879 >$
FS $\mathbb{LEVEL}$	69956(39.03) $< 25547 >$	63548(19.97) $< 20028 >$	47928(102.25) $< 6916 >$	41110(53.33) $< 6775 >$	47964(118.33) $< 6017 >$	40240(7.71) $< 5882 >$
SBAC	69939(56.42) $< 22777 >$	63573(35.45) $< 23605 >$	48006(102.24) $< 6819 >$	41157(52.93) $< 6720 >$	48058(123.01) $< 6008 >$	40225(9.01) $< 5876 >$
$\max H(a)$	69940(33.02) $< 26909 >$	63412(4.20) $< 26282 >$	47387(84.46) $< 6889 >$	40928(46.85) $< 6766 >$	47462(112.07) $< 5968 >$	40201(8.12) $< 5880 >$
$\max N(a)$	70153(71.71) $< 27994 >$	63657(89.47) $< 27969 >$	47884(61.64) $< 6936 >$	41205(39.51) $< 6858 >$	47934(94.06) $< 5993 >$	40227(11.97) $< 5886 >$

Table 3.4: Simulation results (makespan) for e-Discovery benchmark problem *eDisc1* using 5, 50 and 100 processing machines, network file copy speeds of 10 MB/s and 100 MB/s using different heuristics. Work-stealing as a complete information setting.

$\mathcal{H}^p$	$M = 5$		$M = 50$		$M = 100$	
	$T = 10$	$T = 100$	$T = 10$	$T = 100$	$T = 10$	$T = 100$
$\min \tilde{H}(a)$	70016(58.46) $< 19176 >$	63643(58.53) $< 17951 >$	47253(71.02) $< 7104 >$	41550(108.06) $< 6883 >$	47014(173.18) $< 6011 >$	40143(40.07) $< 5911 >$
$\max \tilde{H}(a)$	69809(35.91) $< 26717 >$	63394(4.57) $< 27188 >$	46979(170.51) $< 7058 >$	41241(95.61) $< 6884 >$	46763(100.87) $< 5966 >$	40131(16.40) $< 5880 >$
$\max \tilde{N}(a)$	70195(103.21) $< 25317 >$	63639(51.18) $< 20215 >$	47298(122.17) $< 7273 >$	41395(72.04) $< 6935 >$	46960(156.68) $< 5991 >$	40098(10.75) $< 5898 >$
FS $\mathbb{ALL}$	69859(43.84) $< 20699 >$	63484(14.67) $< 22806 >$	46825(112.52) $< 7295 >$	41288(75.71) $< 6893 >$	47021(184.76) $< 5998 >$	40095(7.96) $< 5900 >$
FS $\mathbb{LEVEL}$	69927(56.19) $< 26575 >$	63504(26.07) $< 24878 >$	46887(104.79) $< 7123 >$	41209(86.51) $< 6880 >$	46869(192.51) $< 5990 >$	40103(13.02) $< 5888 >$
SBAC	69858(42.26) $< 21016 >$	63622(36.06) $< 25982 >$	46911(79.38) $< 7121 >$	41255(82.02) $< 6864 >$	46878(203.68) $< 5989 >$	40098(6.38) $< 5889 >$
$\max H(a)$	69973(50.30) $< 34852 >$	63421(5.01) $< 28785 >$	47028(172.21) $< 7230 >$	41061(55.06) $< 6944 >$	47015(183.45) $< 6001 >$	40099(9.78) $< 5883 >$
$\max N(a)$	70305(131.20) $< 31471 >$	63539(35.51) $< 27494 >$	47304(219.12) $< 7271 >$	41317(60.61) $< 7025 >$	46938(237.82) $< 5989 >$	40116(14.41) $< 5881 >$

Table 3.5: Simulation results (makespan) for e-Discovery benchmark problem *eDisc1* using 5, 50 and 100 processing machines, network file copy speeds of 10 MB/s and 100 MB/s using different heuristics. Work-stealing as an incomplete information setting.

$\mathcal{H}^p$	$M = 5$		$M = 50$		$M = 100$	
	$T = 10$	$T = 100$	$T = 10$	$T = 100$	$T = 10$	$T = 100$
$\min \tilde{H}(a)$	1586610(19158.50) $< 1617748 >$	1429050(15629.93) $< 1254885 >$	933541(10.83) $< 328752 >$	837725(2124.06) $< 383138 >$	933530(31.04) $< 218942 >$	808658(2.87) $< 276805 >$
$\max \tilde{H}(a)$	1309219(387.76) $< 1320855 >$	1186642(73.16) $< 765645 >$	933619(3.17) $< 243677 >$	819857(6.70) $< 238148 >$	933626(6.53) $< 243728 >$	808630(0.08) $< 209073 >$
$\max \tilde{N}(a)$	1578726(29281.41) $< 1831140 >$	1335604(16839.01) $< 1486984 >$	933848(138.13) $< 577829 >$	824845(1070.96) $< 339310 >$	933435(3.07) $< 217661 >$	808631(11.87) $< 239765 >$
FS $\mathbb{ALL}$	1381453(14547.16) $< 1273745 >$	1258205(21223.29) $< 1400627 >$	933919(123.70) $< 586305 >$	826289(1367.80) $< 315003 >$	933577(17.59) $< 274505 >$	809823(1014.13) $< 228177 >$
FS $\mathbb{LEVEL}$	1308338(677.81) $< 1098767 >$	1186882(55.08) $< 710527 >$	933599(95.87) $< 357301 >$	820270(54.83) $< 266978 >$	933491(14.52) $< 235617 >$	808566(62.74) $< 213422 >$
SBAC	1308326(483.40) $< 941394 >$	1186640(63.21) $< 641976 >$	933744(107.77) $< 477772 >$	822457(405.19) $< 248700 >$	933508(1.37) $< 214087 >$	808572(0.90) $< 222840 >$
$\max H(a)$	1311017(338.32) $< 1170564 >$	1187006(47.28) $< 866701 >$	941617(90.42) $< 465412 >$	821772(39.18) $< 250184 >$	941442(2.32) $< 213291 >$	811917(2.82) $< 336759 >$
$\max N(a)$	1311920(489.54) $< 1309449 >$	1187026(76.99) $< 911760 >$	933580(2.61) $< 262419 >$	820477(7.16) $< 254491 >$	933581(0.65) $< 267691 >$	808630(1.94) $< 212941 >$

Table 3.6: Simulation results (makespan) for e-Discovery benchmark problem *eDisc2* using 5, 50 and 100 processing machines, network file copy speeds of 10 MB/s and 100 MB/s using different heuristics. Work-stealing as a complete information setting.

$\mathcal{H}^p$	$M = 5$		$M = 50$		$M = 100$	
	$T = 10$	$T = 100$	$T = 10$	$T = 100$	$T = 10$	$T = 100$
$\min \tilde{H}(a)$	1591558(23130.42) $< 1912378 >$	1457965(18381.40) $< 1467201 >$	921457(345.66) $< 486454 >$	830593(1611.26) $< 299566 >$	919330(32.17) $< 269394 >$	809448(660.05) $< 239831 >$
$\max \tilde{H}(a)$	1309764(314.72) $< 1371639 >$	1186747(56.88) $< 957493 >$	917858(97.64) $< 364297 >$	820789(98.60) $< 286530 >$	917794(32.70) $< 254055 >$	808308(0.33) $< 227272 >$
$\max \tilde{N}(a)$	1556359(31188.37) $< 1777136 >$	1302993(11292.50) $< 1537074 >$	919321(190.79) $< 488516 >$	827441(2620.49) $< 406010 >$	919291(10.89) $< 329354 >$	809631(1217.73) $< 354936 >$
FS $\mathbb{ALL}$	1447698(34396.18) $< 1540748 >$	1288630(24091.06) $< 1392716 >$	920314(753.96) $< 293874 >$	825128(887.31) $< 334931 >$	919295(18.61) $< 266338 >$	808854(318.46) $< 230226 >$
FS $\mathbb{LEVEL}$	1308241(371.00) $< 1190348 >$	1186815(37.77) $< 844518 >$	919387(102.16) $< 319739 >$	821372(587.88) $< 282617 >$	919276(2.48) $< 213279 >$	808311(1.56) $< 214293 >$
SBAC	1307477(445.70) $< 1120472 >$	1186755(69.73) $< 831141 >$	919392(89.55) $< 411518 >$	821358(316.47) $< 275944 >$	919310(22.90) $< 263390 >$	808308(0.32) $< 210429 >$
$\max H(a)$	1310408(345.11) $< 1323191 >$	1186856(59.42) $< 1171030 >$	919404(87.79) $< 402768 >$	820148(217.34) $< 300019 >$	919257(3.90) $< 272079 >$	808316(3.50) $< 263978 >$
$\max N(a)$	1312078(1107.19) $< 1565099 >$	1187194(85.54) $< 1150904 >$	919352(633.69) $< 527073 >$	828574(356.38) $< 302909 >$	918123(33.82) $< 304138 >$	808905(0.25) $< 213965 >$

Table 3.7: Simulation results (makespan) for e-Discovery benchmark problem *eDisc2* using 5, 50 and 100 processing machines, network file copy speeds of 10 MB/s and 100 MB/s using different heuristics. Work-stealing as an incomplete information setting.

We conjecture that a scheduling policy that mixes SBAC and $\mathbb{LEVEL}$ for "shallower" nodes, and $\max \tilde{H}(a)$ and SBAC for nodes deeper in the data tree would perform well in the field and is lightweight enough so that such policy can be used for real-time online scheduling problems. Further studies are needed to confirm or reject our proposal for a mixing policy of this type.

We conclude with a note on the greediness of our algorithm, which is based on Graham's list-scheduling algorithm. The latter is indeed a greedy algorithm in the sense that machines are not allowed to remain idle if there is work to be done. Certainly, in our proposed algorithm an idle machine steals work from another machine immediately upon becoming idle. Due to copy costs, precedence constraints, and the online fashion in which the data tree is revealed, such greedy policy is not necessarily optimal. Consider for example a scenario having two files currently assigned to machine $\mathcal{M}_1$, whereas machine $\mathcal{M}_2$ is idle. Both files have copy times of 1 second and processing times of 0.25 seconds. If machine $\mathcal{M}_2$ steals at least one file, the makespan of the schedule is > 1. On the other hand, if machine $\mathcal{M}_2$ is left idling, then the makespan is 0.5. In a similar argument, one can easily construct examples of unbalanced data trees where waiting until a file's attachments are uncovered can result in a better work-stealing balance across the machines involved in the steal, thus improving on the makespan compared to when the steal occurs greedily in presence of higher uncertainly. [143] covers interesting examples where greedy schedules are suboptimal in stochastic scheduling settings with and without precedence constraints. We strongly encourage the interested reader to consult their work.

3.8 Concluding Remarks

In this chapter, we introduced the e-Discovery scheduling problem, which is effectively a *job-shop* scheduling problem with precedence constrains and setup times [11]. We surveyed related results from scheduling theory and combined some of these ideas to produce a distribution-free online algorithm for scheduling in an e-Discovery setting. Our algorithm does not require any knowledge of the underlying mathematical models embedded within the data set that is to be processed. Our algorithm also boasts very low computational overhead, which makes it usable for real-time scheduling problems as well as "production" environments. We found that three heuristics perform particularly well

[11]In scheduling theory, the term "setup times" is commonly used, whereas in the multi-armed bandit domain, the term "switching costs" is commonly used instead. The two carry the same meaning, which is a cost associated with migrating a task (or ready-file in the e-Discovery setting) from one machine to another.

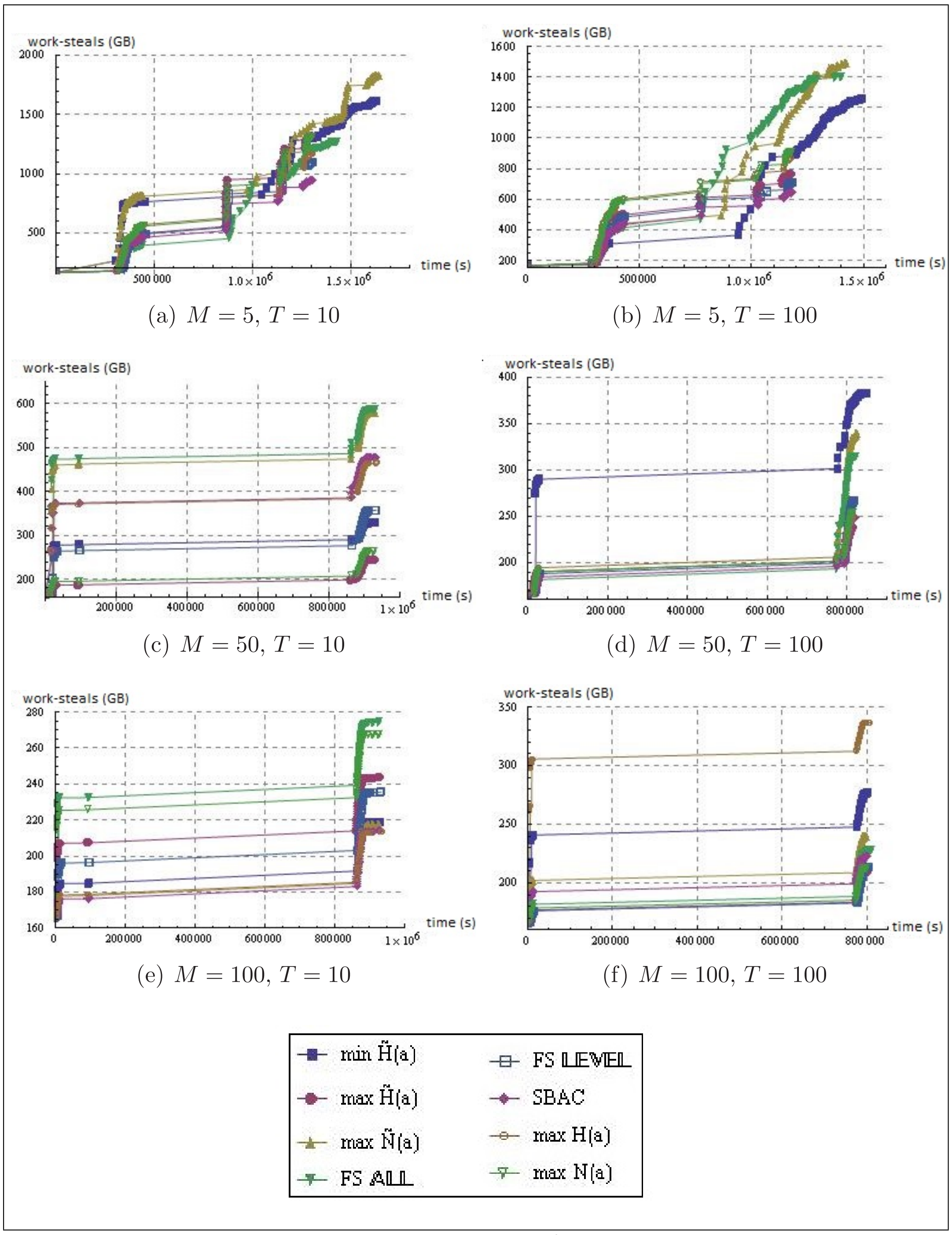
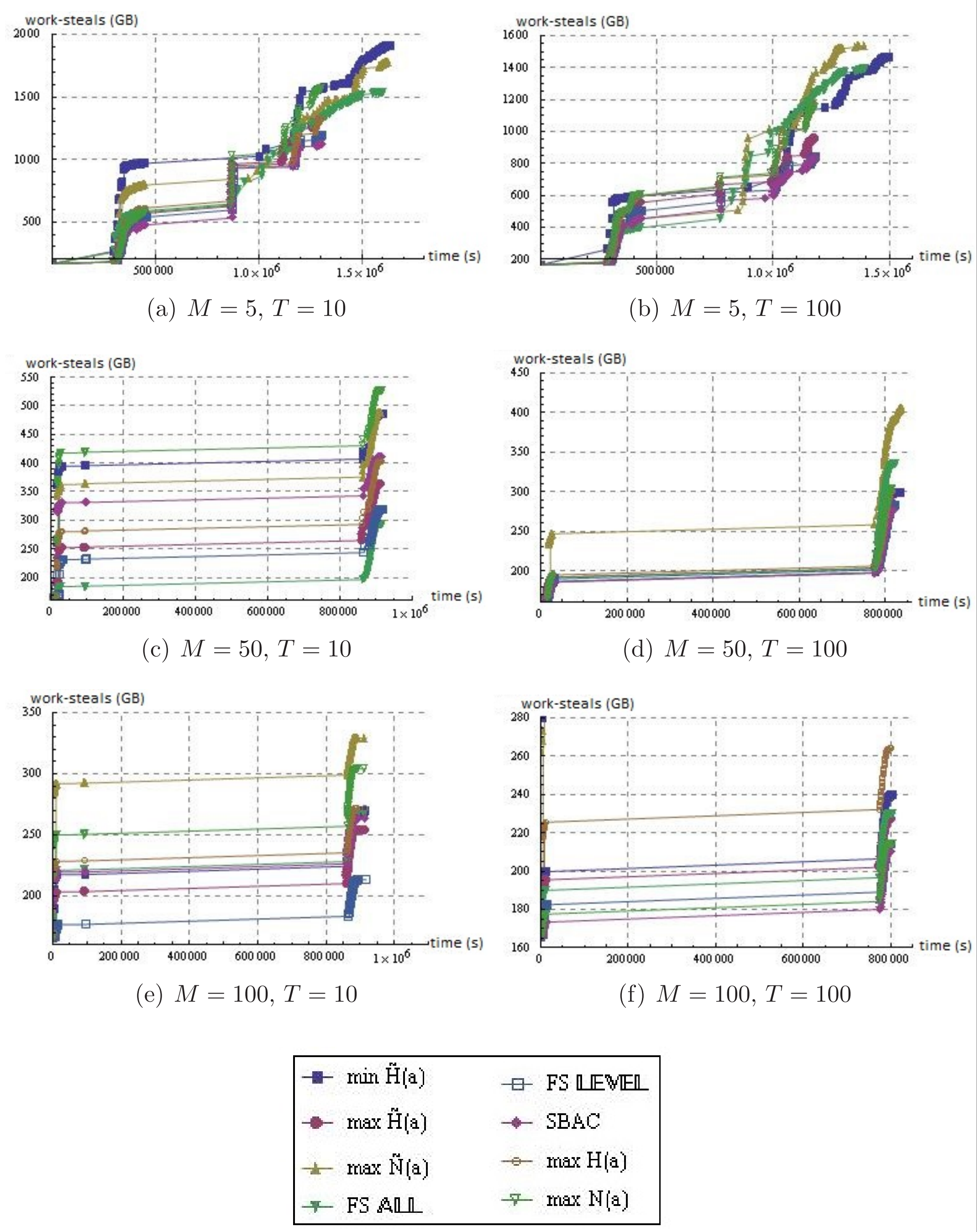


Figure 3.7: Total work-stealing over time for the *eDisc2* problem under a complete information setting of work-stealing.



(a) $M = 5$, $T = 10$ (b) $M = 5$, $T = 100$

(c) $M = 50$, $T = 10$ (d) $M = 50$, $T = 100$

(e) $M = 100$, $T = 10$ (f) $M = 100$, $T = 100$

Figure 3.8: Total work-stealing over time for the *eDisc2* problem under an incomplete information setting of work-stealing.

in the e-Discovery setting.

We demonstrated how incorporating a bandit model into list-scheduling can result in improved performance, both in the sense of makespan and total generated network traffic, compared to the "non-bandit" counterpart heuristic, and the results are encouraging. To achieve this, we provided a novel algorithm that we believe to be the first to account for both sleeping bandits and bandit history in a single algorithm. We conjecture that the algorithm achieves logarithmic regret, but leave that for future work.

In our simulations, we made a simplification that assumes file copying (setup/switching) costs are deterministic in the size of the file/task being transferred across machines. It would be interesting in future work to consider randomness in the copying costs due to network load. In a related sense, it would be interesting to consider a problem with multiple objectives of reducing the makespan as well as minimizing the number of concurrent file copy requests, which can in practice degrade the performance of a network. Achieving such Pareto improvements without modifications to the computer and network infrastructure is of particular interest to practitioners. Another direction of interest is to allow file processing and work-stealing to occur in parallel, which is very practical since the two operations use different resources on a processing machine (processing is CPU-intensive whereas file copying is IO-intensive). Finally, the performance of our algorithm was demonstrated empirically, and it is desirable to conduct a formal analysis to obtain theoretical guarantees of average and worse-case performance. It is already known that the problem is NP-completed even without switching costs.

Bibliography

[1] U. A. Acar and G. E. Blelloch. *The data locality of work stealing.* Theory of Computing Systems, 1-12, 2000.

[2] R. Agrawal, M. V. Hedge, and D. Teneketzis. *Asymptotically efficient adaptive allocation rules for the multiarmed bandit problem with switching cost.* IEEE Transactions on Automatic Control, 33(10):899 - 906, 1988.

[3] R. Agrawal, M. V. Hedge, and D. Teneketzis. *Multi-Armed Bandit Problems with Multiple Plays and Switching Cost.* Stochastics and Stochastic Reports, 29:437-459, 1990.

[4] S. Agrawal and N. Goyal. *Analysis of Thompson sampling for the multi-armed bandit problem.* In Proceedings of the 25th Annual Conference on Learning Theory (COLT), JMLR Workshop and Conference Proceedings Vol. 23, 2012.

[5] I. Ahmad, Y. Kwok, and M. Wu. *Performance comparison of algorithms for static scheduling of DAGs to multiprocessors.* Second Australasian Conference on Parallel and Real-time Systems, 185-192, 1995.

[6] S. Albers. *Better bounds for online scheduling.* SIAM Journal on Computing, 29(2):459473, 1999.

[7] N. S. Arora, R. D. Blumofe, and C. G. Plaxton. *Thread scheduling for multiprogrammed multiprocessors.* Proceedings of the Tenth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA), Puerto Vallarta, 119-129, 1998.

[8] N. S. Arora, R. D. Blumofe, and C. G. Plaxton. *Thread scheduling for multiprogrammed multiprocessors.* Theory of Computing Systems, 34(2):115-144, 2001.

[9] R. Arora, O. Dekel, and A. Tewari. *Online bandit learning against an adaptive adversary: from regret to policy regret.* In Proceedings of the 29th International Conference on Machine Learning (ICML), 2012.

[10] K. J. Åström. *Optimal control of Markov decision processes with incomplete state estimation.* Journal of Mathematical Analysis and Applications, 10:174-205, 1965.

[11] J.-Y. Audibert and S. Bubeck. *Minimax policies for adversarial and stochastic bandits.* In Proceedings of the 22nd Annual Conference on Learning Theory (COLT), 2009.

[12] J.-Y. Audibert and S. Bubeck. *Regret bounds and minimax policies under partial monitoring.* Journal of Machine Learning Research, 11:2635-2686, 2010.

[13] P. Auer, N. Cesa-Bianchi and P. Fisher. *Finite-time analysis of the multiarmed bandit problem.* Machine Learning, 47:235-256, 2002.

[14] P. Auer, N. Cesa-Bianchi, Y. Freund and R. Schapire. *The nonstochastic multiarmed bandit problem.* SIAM Journal on Computing, 32(1):48-77, 2003.

[15] P. Auer and R. Ortner. *UCB revisited: Improved regret bounds for the stochastic multi-armed bandit problem.* Periodica Mathematica Hungarica, 61:55-65, 2010.

[16] M. Aufenanger, and P. van Lck, *Simulation-based adaption of scheduling knowledge*, Proceedings of the 2010 Winter Simulation Conference, 2010.

[17] Y. Aumann, K. Palem, Z. Kedem, and M. O. Rabin. *Highly efficient asynchronous execution of large grained parallel programs.* Procs. of the 34th Annual Symposium on the Foundations of Computer Science (FOCS), 271-280, 1993.

[18] Y. Aumann, M. A. Bender, and L. Zhang. *Efficient execution of nondeterministic parallel programs on asynchronous systems.* Information and Computation, 139(1):1-16, 1997.

[19] R. Azoulay-Schwartz, S. Kraus and J. Wilkenfeld. *Exploitation vs. Exploration: Choosing a Supplier in an Environment of Incomplete Information*, Decision Support Systems, 38(1):1-18, 2004.

[20] A. Banõs. *On pseudo games.* The Annals of Mathematical Statistics, 39:1932-1945, 1968.

[21] R. Bellman. *Dynamic programming.* Princeton University Press, 1957.

[22] M. A. Bender, and M. O. Rabin. *Scheduling cilk multithreaded parallel programs on processors of different speeds.* Proceedings of the 12th Annual Symposium on Parallel Algorithms and Architectures, 13-21, 2000.

[23] M. A. Bender, and M. O. Rabin. *Online scheduling of parallel programs on heterogeneous systems with applicatoins to CILK.* Theory of Computing Systems Special Issue on SPAA, 35(3):289-304, 2002.

[24] M. A. Bender, and C. A. Phillips. *Scheduling DAGs on asynchronous processors.* 19th ACM Symp. on Parallel Algorithms and Architectures, 35-45, 2007.

[25] L. Benkherouf and J.A. Bather. *Oil exploration: Sequential decisions in the face of uncertainty.* Journal of Applied Probability, 28:529-543, 1988.

[26] L. Benkherouf, K.D. Glazebrook and R.W. Owen. *Gittins indices and oil exploration.* Journal of Royal Statistical Society, Serial B, 54:229-241, 1992.

[27] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-dymanic programming.* Belmont, MA Athena Scientific, 1996.

[28] D. P. Bertsekas and D. A. Castanon. *Rollout algorithms for stochastic scheduling problems.* Journal of Heuristics 5(1):89-108, 1999.

[29] D. P. Bertsekas. *Dynamic programming and optimal control.* Athena Scientic, Belmont, MA, 2001. 2nd Edition.

[30] J. Bidot, T. Vidal, P. Laborie, and J. C. Beck. *A theoretic and practical framework for scheduling in a stochastic environment.* Journal of Scheduling, 12(3):315-344, 2009.

[31] R. D. Blumofe, and C. E. Leiserson. *Scheduling multithreaded computations by work stealing.* 35th Annual Symposium on Foundations of Computer Science (FOCS 1994), 1994.

[32] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou. *Cilk: An efficient multithreaded runtime system.* Journal of Parallel and Distributed Computing, 1995.

[33] R. D. Blumofe and D. Papadopoulos. *The performance of work stealing in multiprogrammed environments (extended abstract).* Proceedings of the 1998 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, Poster Session, 266-267, 1998.

[34] R. D. Blumofe, and C. E. Leiserson. *Scheduling multithreaded computations by work stealing*, Journal of the ACM (JACM), 46(5):720-748, 1999.

[35] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund. *A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems.* Journal of Parallel and Distributed Computing. 61(6):810-837, 2001.

[36] R. P. Brent. *The parallel evaluation of general arithmetic expressions.* Journal of the ACM, 21(2):201-206, 1974.

[37] S. Bubeck and N Cesa-Bianchi. *Regret analysis of stochastic and non-stochastic multi-armed bandit problems.* Foundations and Trends in Machine Learning, 5(1):1-122, 2012.

[38] S. Bubeck and A. Slivkins. *The best of both worlds: stochastic and adversarial bandits.* In Proceedings of the 25th Annual Conference on Learning Theory (COLT), JMLR Workshop and Conference Proceedings, vol. 23, 2012.

[39] F. Burton, and M. Sleep. *Executing functional programs on a virtual tree of processors.* Proc. of the 1981 conference on Functional programming languages and computer architecture (FPCA), 187-194, 1981.

[40] A. R. Cassandra, M. L. Littman, and N. L. Zhang. *Incremental pruning: A simple, fast, exact algorithm for partially observable Markov decision processes.* In Proceedings of Uncertainty in Artificial Intelligence, 54-61, 1997.

[41] H. S. Chang, R. Givan, and E. K. P. Chong. *On-line scheduling via sampling.* Artificial Intelligence Planning and Scheduling (AIPS), 62-71, 2000.

[42] H.S. Chang and S. I. Marcus. *Approximate receding horizon approach for Markov decision processes: Average reward case.* Journal of Mathematical Analysis and Applications, vol. 286, 2003.

[43] H.S. Chang, M.C. Fu, J. Hu, and S.I. Marcus. *Recursive learning automata approach to Markov decision processes.* IEEE Transactions on Automatic Control, 52(7):1249-1355, 2007.

[44] H. S. Chang, M. C. Fu, J. Q. Hu and S. I. Marcus. *Simulation-based algorithms for Markov decision processes.* Springer. 2007.

[45] C. Chekuri and M. A. Bender. *An efficient approximation algorithm for minimizing makespan on uniformly related machines.* Proc. of the Sixth Conference on Integer Programming and Combinatorial Optimization (IPCO), Lecture Notes in Computer Science, 1412:383-393, 1998.

[46] C. Chekuri, and M. A. Bender. *An efficient approximation algorithm for minimizing makespan on uniformly related machines.* Journal of Algorithms, 41:212-224, 2001.

[47] R. M. Chen, and Y. M. Huang. *Multiconstraint task scheduling in multiprocessor systems by neural networks*, 10th IEEE Conference on Tools with Artificial Intelligence, 288-294, 1998.

[48] H. Chen. *Distributed dynamic scheduling for composite tasks on grid computing system*, Masters Thesis, Department of Electrical & Computer Engineering, University of Manitoba, Winnipeg, Manitoba, Canada, 2001.

[49] H. T. Cheng. *Algorithms for partially observable Markov decision processes.* PhD thesis, University of British Columbia, 1988.

[50] F. A. Chudak, and D. B. Shmoys. *Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds (extended abstract).* Proc. of the Eighth Annual ACM SIAM Symposium on Discrete Algorithms (SODA), 581-590, Jan. 1997.

[51] F. A. Chudak and D. B. Shmoys. *Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds.* Journal of Algorithms, 30(2):323-343, 1999.

[52] V. Cicirello and S. F. Smith. *The max k-armed bandit: A new model of exploration applied to search heuristic selection.* In Proceedings of the Twentieth National Conference on Artificial Intelligence, 13551361, 2005.

[53] R. Cole, and O. Zajicek. *The expected advantage of asynchrony.* SPAA '90 Proceedings of the second annual ACM symposium on Parallel algorithms and architectures, 1990.

[54] T. H. Cormen, C. E. Leiserson, R. L. Rivest. *Introduction to algorithms*, MIT Press, Cambridge, MA, 1992.

[55] G. Cordasco, and A. L. Rosenberg. *On scheduling DAGs to maximize area.* Parallel and Distributed Processing, 2009:1-12, 2009.

[56] R. Dearden, N. Friedman, and S. Russell. Bayesian Q learning. AAAI, 15:761-768, 1998.

[57] E. B. Dynkin. *Controlled random sequences.* Theory of probability and its applications, 10(1): 1-14, 1965.

[58] H. El-Rewini and T. G. Lewis. *Scheduling parallel program tasks onto arbitrary target machines.* Journal of Parallel Distrib. Comput. 9(2):138-153, 1990.

[59] M. Eshaghian. *Heterogeneous computing*, Edited, Artech House, Norwood, MA, 1996.

[60] D. Fernandez-Baca. *Allocating modules to processors in a distributed system.* IEEE Trans. Software Engrg, 15(11):1427-1436, 1989.

[61] D. P. Foster, R. Vohra. *Regret in the Online Decision Problem.* Games and Economic Behavior, 29:7-35, 1999.

[62] T. Furmston, and D. Barber. *Lagrange dual decomposition for finite horizon Markov decision processes.* Proceedings of the 2011 European conference on Machine learning and knowledge discovery in databases, vol. Part I. pp. 487-502. Springer-Verlag. 2011.

[63] A. Garivier and O. Cappé. *The KL-UCB algorithm for bounded stochastic bandits and beyond.* In Proceedings of the 24th Annual Conference on Learning Theory (COLT), JMLRWorkshop and Conference Proceedings, vol. 19, 2011.

[64] R. L. Graham. *Bounds for certain multiprocessing anomalies.* The Bell System Technical Journal, 45:1563-1581, 1966.

[65] R. L. Graham. *Bounds on multiprocessing timing anomalies.* SIAM Journal on Applied Mathematics, 17(2):416-429, 1969.

[66] C. Guestrin, D. Koller, R. Parr, and S Venkataraman. *Efficient Solution Algorithms for Factored MDPs.* Journal of Artificial Intelligence Research, 19(1):399-468, 2003.

[67] E. Haddad. *Dynamic load distribution optimization in heterogeneous multiple processor systems.* Technical Report TR-93-02, Computer Science, Virginia Polytechnic Institute and State University, 1993.

[68] R. Hall, A. L. Rosenberg, and A. Venkataramani. *A comparison of DAG-scheduling strategies for internet-based computing.* International Parallel and Distributed Processing Symposium/International Parallel Processing Symposium - IPDPS(IPPS), 1-9, 2007.

[69] R. Halstead. *MULTILISP: A language for concurrent symbolic computation.* ACM Transactions on Programming Languages and Systems (TOPLAS), 7(4):501-538, 1985.

[70] B. Hamidzadeh, D. J. Lilja, and Y. Atif. *Dynamic scheduling techniques for heterogeneous computing systems.* Concurrency: Practice and Experience, 7(7):633-652, 1995.

[71] J. Hannan. *Approximation to Bayes risk in repeated play.* Contributions to the theory of games, 3:97-139, 1957.

[72] O. Hernandez-Lerma. *Adaptive Marokov Control Processes.* New York. Springer-Verlag. 1989.

[73] O. Hernandez-Lerma and J.B. Lasserre. *Error bounds for rolling horizon policies in discrete-time Markov control processes.* IEEE Transactions on Automatic Control, vol. 35:1118-1124, 1990.

[74] J. Honda and A. Takemura. *An asymptotically optimal bandit algorithm for bounded supportmodels.* In Proceedings of the 23rd Annual Conference on Learning Theory (COLT), 2010.

[75] J. Q. Hu. *Randomized search methods for solving Markov decision processes and global optimization.* PhD thesis, Electrical Engineering, University of Maryland (College Park, Md.), 2006.

[76] J. Q. Hu and H. S. Chang. *An aproximate stochastic annealing algorithm for finite horizon Markov decision processes.* In Proceedings of the 49th IEEE Conference on Decision and Control (CDC), 5338-5343, 2010.

[77] O. H. Ibarra and C. E. Kim. *Heuristic algorithms for scheduling independent tasks on nonidentical processors*, Journal of the ACM, 24(2):280-289, 1977.

[78] J. Idziorek. *Discrete event simulation model for analysis of horizontal scaling in the cloud computing model.* Proceedings of the 2010 Winter Simulation Conference, 3004-3014, 2010.

[79] M. Iverson, F. Özgüner, and G. Follen. *Parallelizing existing application in a distributed heterogeneous environment.* 4th IEEE Heterogeneous Computing Workshop (HCW '95), 93-100, 1995.

[80] M. Iverson, and F. Özgüner. *Dynamic, competitive scheduling of multiple DAGs in a distributed heterogeneous environment.* Heterogeneous Computing Workshop, 1998.

[81] J. M. Jaffe. *An analysis of preemptive multiprocessor job scheduling.* Mathematics of Operations Research, 5(3):415-421, 1980.

[82] J. M. Jaffe. *Efficient scheduling of tasks without full use of processor resources.* Theoretical Computer Science, 12:1-17, Aug. 1980.

[83] K. Jamieson, M. Malloy, S. Bubeck and R. Nowak. *lil' UCB: An Optimal Exploration Algorithm for Multi-Armed Bandits.* In Proceedings of the 27th Annual Conference on Learning Theory (COLT), JMLR Workshop and Conference Proceedings, vol. 35, 2014.

[84] T. Jun. *A survey on the bandit problem with switching costs.* De Economist, 152(4):513-541. 2004.

[85] L. P. Kaelbling, M. L. Littman and A. R. Cassandra. *Planning and acting in partially observable stochastic domains.* Artificial Intelligence, 101(1-2):99-134, 1998.

[86] S. Kakade. *On the Sample Complexity of Reinforcement Learning.* PhD thesis, University College London, 2003.

[87] E. Kaufmann, N. Korda, and R. Munos. *Thompson sampling: An asymptotically optimal finite-time analysis.* In Proceedings of the 23rd International Conference on Algorithmic Learning Theory (ALT), 2012.

[88] M. Kearns, Y. Mansour, and A.Y. Ng. *Approximate planning in large POMDPs via reusable tragectories.* Advances in Neural Information Processing Systems, 1999.

[89] M. Kearns, Y. Mansour and A. Y. Ng. *A sparse sampling algorithm for near-optimal planning in large Markov decision processes.* Machine Learning, 49:193-208, 2002.

[90] Z. M. Kedem, K. V. Palem, M. O. Rabin, and A. Raghunathan. *Efficient program transformation for resilient parallel computation via randomization.* Proc. of the 24th Annual ACM Symposium on the Theory of Computing (STOC), 306-317, 1992.

[91] J. Kelley, Jr. *Critical path planning and scheduling: Mathematical basis.* Operations Research, 9(3):296-320, 1961.

[92] R. Kleinberg, A. Niculescu-Mizil, and Y. Sharma. *Regret bounds for sleeping experts and bandits.* Machine Learning, 80(2-3):245-272, 2010.

[93] T. Lai and H. Robbins. *Asymptotically efficient adaptive allocation rules.* Advances in Applied Mathematics, 6:4-22, 1985.

[94] C. Leangsuksun, J. Potter, and S. Scott. *Dynamic task mapping algorithms for a distributed heterogeneous computing environment.* 4th Heterogeneous Computing Workshop (HCW 95), 30-34, 1995.

[95] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker. *Complexity of machine scheduling problems.* Annals of Discrete Mathematics, 1:343-362, 1977.

[96] C. K. Li and W. K. Wong. *Extension of stochastic dominance theory to random variables.* RAIRO - Operations Research, 33(4):509-524, 1999.

[97] O.-A. Maillard, R. Munos, and G. Stoltz. *A finite-time analysis of multi-armed bandits problems with Kullback-Leibler divergences.* In Proceedings of the 24th Annual Conference on Learning Theory (COLT), JMLR Workshop and Conference Proceedings, vol. 19, 2011.

[98] A. Mahajan and D. Teneketzis. *Multi-armed bandit problems.* In Foundations and Applications of Sensor Management. 121 151. Springer. 2008.

[99] M. Maheswaran and H. J. Siegel. *A dynamic matching and scheduling algorithm for heterogeneous computing systems.* In Seventh Heterogeneous Computing Workshop, 57-69, 1998.

[100] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund. *Dynamic mapping of a class of independent tasks onto heterogeneous computing systems.* Journal of Parallel Distributed Computing, 59(2):107-121, 1999.

[101] M. S. Maxwell, S. G. Henderson, and H. Topaloglu. *Identifying effective policies in approximate dynamic programming: beyond regression.* Proceedings of the 2010 Winter Simulation Conference, 2010.

[102] D. A. McAllester and S. Singh. *Approximate Planning for Factored POMDPs using Belief State Simplification.* In Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, 409-416, 1999.

[103] B. P. McCall and J. J. McCall. *A Sequential study of migration and job search.* Journal of Labor Economics, 5:452476, 1987.

[104] A. McLennan. *Price dispersion and incomplete learning in the long run.* Journal of Economic Dynamics and Control, 7:331-347, 1984.

[105] M. Mitzenmacher. *Analyses of load stealing models based on differential equations.* Proceedings of the 10th Annual ACM Symposium on Parallel Algorithms and Architectures, 212-221, 1998.

[106] R. H. Möhring, A. S. Schulz, and M. Uetz. *Approximation in stochastic scheduling: The power of LP-based priority policies.* Journal of the ACM (JACM), 46(6):924-942, 1999.

[107] G. E. Monahan. *A survey of partially observable Markov decision processes: Theory, models, and algorithms.* Management Science, 28(1):1-16, 1982.

[108] D. B. Neill. *Work stealing: An annotated bibliography*, Annotated Bibliography, http://tinyurl.com/orcyxdd.

[109] J. Neufeld, A. György, D. Schuurmans, and Cs. Szepesvári. *Adaptive Monte Carlo via Bandit Allocation.* In Proceedings of the 31st International Conference on Machine Learning (ICML '2014), 1944-1952, 2014.

[110] R. Ortner, D. Ryabko, P. Auer, and R. Munos. *Regret bounds for restless Markov bandits.* arXiv preprint arXiv:1209.2693, 2012.

[111] Sandeep Pandey, Deepayan Chakrabarti, and Deepak Agarwal. *Multi-armed bandit problems with dependent arms.* In Proceedings of the 24th international conference on Machine learning (ICML '07), 721-728, 2007.

[112] V. Perchet and P. Rigollet. *The multi-armed bandit problem with covariates.* Arxiv preprint arXiv:1110.6084, 2011.

[113] J. Pineau, G. Gordon and S. Thrun. *Point-based value iteration: An anytime algorithm for pomdps.* In Proceedings of the International Joint Conference on Artificial Intelligence, 1025-1032, 2003.

[114] J. M. Porta, N. Vlassis, M. T. J. Spaan, and P. Poupart. *Point-based value iteration for continuous POMDPs.* Journal of Machine Learning Research, 7:2329-2367, 2006.

[115] D. Pucci de Farias and N. Megiddo. *Combining expert advice in reactive environments.* Journal of the ACM, 53(5):762-799, 2006.

[116] M. L. Puterman. *Markov decision processes: Discrete stochastic dynamic programming.* New York. Wiley. 1994.

[117] L. R. Rabiner. *A tutorial on hidden Markov models and selected applications in speech recognition.* Proceedings of the IEEE, 77(2):257-285, 1989.

[118] H. Robbins. *Some aspects of the sequential design of experiments.* Bulletin of the American Mathematical Society, 58:527-535, 1952.

[119] M. Rothschild. *A Two-armed bandit theory of market pricing.* Journal of Economic Theory, 9:185-202, 1974.

[120] S. Ryu. *Scheduling multithreaded computations by work stealing.* Presentation. http://plrg.kaist.ac.kr/_media/home/lectures/cs720_2010-1/lecture14.pdf, 2010.

[121] G. Santharam and P. S. Sastry. *A reinforcement learning neural network for adaptive control of Markov chains.* IEEE Transactions on Systems, Man, and Cybernetics, Part A Systems and Humans, 27(5):588-600, 1997.

[122] K. H. Schlag. *How to Minimize Maximum Regret under Repeated Decision-Making.* Discussion paper, Florence: European University Institute, 2003.

[123] R. J. Serfling. *Some elementary results on Poisson appoximation in a sequence of Bernoulli trails.* SIAM Rev, 20(3):567-579, 1978.

[124] P. K. Shivaswamy and T. Joachims. *Multi-armed bandit problems with history.* Journal of Machine Learning Research - Proceedings Track, 1046-1054, 2012.

[125] H. J. Siegel, J. K. Antonio, R. C. Metzger, M. Tan, and Y. A. Li. *Heterogeneous computing.* Parallel and Distributed Computing Handbook, A. Y. Zomaya, ed., McGraw-Hill, New York, NY, 725-761, 1996.

[126] G. C. Sih, and E. A. Lee. *A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures.* IEEE Transactions on Parallel Distributed Systems, 4(2):175-186, 1993.

[127] H. D. Simon, and S. H. Teng. *How good is recursive bisection?.* SIAM Journal Scientific Computing, 18(5):1436-1445, 1997.

[128] A. Skoogh and J. Michaloski. *Towards continuously updated simulation models: combining automated raw data collection and automated data processing*, Proceedings of the 2010 Winter Simulation Conference, 2010.

[129] E. J. Sondik. *The optimal control of partially observable Markov processes.* PhD thesis, Stanford University, 1971.

[130] M. T. J. Spaan and N. Vlassis. *Perseus: Randomized point-based value iteration for POMDPs.* Journal of Artificial Intelligence Research, 24:195-220, 2005.

[131] M. S. Squillante and R. D. Nelson. *Analysis of task migration in shared-memory multiprocessor scheduling.* Proceedings of the 1991 ACM SIGMETRICS conference on Measurement and modeling of computer systems, 143-155, 1991.

[132] M. S. Squillante, and E. D. Lazowska. *Using processor-cache affinity information in shared-memory multiprocessor scheduling*, IEEE Transactions on Parallel and Distributed Systems, 131-143, 1993.

[133] M. S. Squillante, C. H. Xia, D. D. Yao, and L. Zhang. *Threshold-based priority policies for parallel-server systems with affinity scheduling*, Proc. American Control Conference, 2992-2999, 2001.

[134] M. Streeter and S. F. Smith. *A simple distribution-free approach to the max k-armed bandit problem.* In Proceedings of the 12th international conference on principles and practice of constraint programming. Lecture Notes in Computer Science 4204, 560-574. Springer, Berlin. 2006.

[135] R. Sutton and A. Barto. *Reinforcement learning, an introduction.* Cambridge: MIT Press/Bradford Books. 1998.
Szepesvari2010 Cs. Szepesvri. *Reinforcement Learning Algorithms for MDPs.* Wiley Encyclopedia of Operations Research, Wiley. 2010.

[136] M. Tchiboukdjian, N. Gast, D. Trystram, J. Roch, and J. Bernard. *A tighter analysis of work stealing.* Proceedings of ISAAC (2):291-302, 2010.

[137] C. Tekin and M. Liu. *Online learning of rested and restless bandits.* IEEE Transactions on Information Theory, 58(8):5588-5611, 2012.

[138] L. Tesfatsion. *Stochasic dominance and maximization of expected utility.* Review of Economic Studies, 43:301-315, 1976.

[139] W. Thompson. *On the likelihood that one unknown probability exceeds another in view of the evidence of two samples.* Bulletin of the American Mathematics Society, 25:285-294, 1933.

[140] M. Tokic. *Adaptive ϵ-greedy exploration in reinforcement learning based on value differences.* KI 2010: Advances in Artificial Intelligence, Lecture Notes in Computer Science, 6359, Springer-Verlag, 203-210, 2010.

[141] H. Topcuoglu, S. Hariri, and M.-Y. Wu. *Task scheduling algorithms for heterogeneous processors.* 8th IEEE Heterogeneous Computing Workshop (HCW '99), 3-14, 1999.

[142] F. Topsϕe. *Some bounds for the logarithmic function.* Research report collection, 2004.

[143] M. Uetz. *When greediness fails: Examples from stochastic scheduling.* Operations Research Letters, 31:413-419, 2003.

[144] M. Van Oyen and J. Pichitlamken. *Properties of optimal-weighted flow-time policies with a makespan constraint and set-up times.* Manufacturing & Service Operations Management, 2(1):84-99, 2000.

[145] E. Vecchia, S. Marco, and A. Jean-Marie. *Illustrated review of convergence conditions of the value iteration algorithm and the rolling horizon procedure for average-cost MDPs.* Annals of Operations Research, 199(1):193-214, 2012.

[146] C. J. C. H. Watkins. *Learning from delayed rewards.* PhD tesis, Cambridge University, 1989.

[147] C. J. C. H. Watkins and P. Dayan. *Q-learning.* Machine Learning, 8:279-292, 1992.

[148] M.L. Weitzman. *Optimal search for the best alternative.* Econometrica, 47:641-654, 1979.

[149] N. L. Zhang and W. Zhang. *Speeding up the convergence of value iteration in partially observable Markov decision processes.* Journal of Artificial Intelligence Research, 14:29-51, 2001.

[150] H. B. Zhou. *Scheduling DAGs on a bounded number of processors.* Parallel and Distributed Processing Techniques and Applications, 823-834, 1996.

[151] http://research.microsoft.com/en-us/projects/bandits/

[152] http://supertech.csail.mit.edu/cilk

[153] http://en.wikipedia.org/wiki/List_scheduling