Using Online Algorithms to Solve NP-Hard Problems More Efficiently in Practice

Matthew Streeter

CMU-CS-07-172

December 2007

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Stephen F. Smith, chair
Avrim Blum
Tuomas Sandholm
Carla P. Gomes, Cornell University
John N. Hooker, CMU Tepper School of Business

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was sponsored by the National Science Foundation under contract #9900298, by DARPA under contract #FA8750-05-C-0033, and by the CMU Robotics Institute.

The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies of the U.S. Government.

Keywords: online algorithms, submodular functions, algorithm portfolios, restart schedules, task-switching schedules, multi-armed bandit problems

Abstract

This thesis develops online algorithms that can be used to solve a wide variety of NP-hard problems more efficiently in practice. The common approach taken by all our online algorithms is to improve the performance of one or more existing algorithms for a specific NP-hard problem by adapting the algorithms to the sequence of problem instance(s) they are run on.

We begin by presenting an algorithm for solving a specific class of online resource allocation problems. Our online algorithm can be applied in environments where abstract *jobs* arrive one at a time, and one can complete the jobs by investing time in a number of abstract *activities*. Provided the jobs and activities satisfy certain technical conditions, our online algorithm is guaranteed to perform almost as well as any fixed schedule for investing time in the various activities, according to two natural measures of performance: (*i*) the average time required to complete each job, and (*ii*) the number of jobs completed within time T, for some fixed deadline $T > 0$.

In particular, our online algorithm's guarantees apply if the job can be written as a monotone, submodular function of a set of pairs of the form (v, τ), where τ is the time invested in activity v. Under the first objective, the offline version of this problem generalizes MIN-SUM SET COVER and the related PIPELINED SET COVER problem. Under the second objective, the offline version of this problem generalizes the problem of maximizing a monotone, submodular set function subject to a knapsack constraint. Our online algorithm has potential applications in a number of areas, including the design of algorithm portfolios, database query processing, and sensor placement.

We apply this online algorithm to the following problem. We are given k algorithms, and are fed, one at a time, a sequence of problem instances to solve. We may solve each instance using any of the k algorithms, we may interleave the execution of the algorithms, and, if the algorithms are randomized, we may periodically restart them with a fresh random seed. Our goal is to minimize the total CPU time required to solve all the instances. Using data from eight recent solver competitions, we show that our online algorithm and its offline counterpart can be used to improve the performance of state-of-the-art solvers in a number of problem domains, including Boolean satisfiability, zero-one integer programming, constraint satisfaction, and theorem proving.

We next present an online algorithm that can be used to improve the performance of algorithms that solve an optimization problem by making a sequence of calls to a decision procedure that answers questions of the form "Is there a solution of cost at most k?" We present an adaptive strategy for determining the sequence of questions to ask, along with bounds on the maximum time to spend waiting for an answer to each question. Under the assumption that the time required by the decision procedure to return an answer increases as k gets closer to the optimal solution cost, our strategy's performance is near-optimal when measured in terms of a natural competitive ratio. Experimentally, we show that applying our strategy to recent algorithms for A.I. planning and job shop scheduling allows the algorithms to find approximately optimal solutions more quickly.

Lastly, we develop algorithms for solving the max k-armed bandit problem, a variant of the classical k-armed bandit problem in which one seeks to maximize the highest payoff received on any single trial, rather than the cumulative payoff. A strategy for solving the max k-armed bandit problem can be used to allocate trials among multi-start optimization heuristics. Motivated by results in extreme value theory, we present a no-regret strategy for the special case in which each arm returns payoffs drawn from a generalized extreme value distribution. We also present a heuristic strategy that solves the max k-armed bandit problem using a strategy for the classical k-armed bandit problem as a subroutine. Experimentally, we show that our max k-armed bandit strategy can be used to effectively allocate trials among multi-start heuristics for the RCPSP/max, a difficult real-world scheduling problem.

Acknowledgments

I would like to thank my advisor Stephen Smith, my co-author Daniel Golovin, my committee members Avrim Blum, Carla Gomes, John Hooker, and Tuomas Sandholm, my office-mates David Brumley and Vyas Sekar, all my friends and family, and anyone else who reads this acknowledgments page.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis is about solving NP-hard computational problems more efficiently in practice.

Although conjectured to be worst-case intractable, NP-hard problems arise frequently in the real world. Solving them efficiently is a central concern in fields such as operations research, computational biology, artificial intelligence, and formal verification.

Looking over the past few decades of computer science research, we may distinguish several high-level approaches to dealing with NP-hard problems:

1. *Problem-specific theoretical analysis.* Instances of this approach include the development of constant factor approximation algorithms for a wide variety of NP-hard optimization problems [87], improved exponential-time algorithms [88], and analyses of algorithms for random and semi-random problems [25].

2. *Problem-specific engineering.* Examples of this approach include the ongoing quest for efficient Boolean satisfiability solvers [91], and algorithms for solving specific operations research problems such as job shop scheduling [39].

3. *Black-box optimization.* A number of algorithms have been developed that aim to solve a wide variety of optimization problems, given only black-box access to the to-be-optimized function. Example of such algorithms include the simulated annealing algorithm [50], genetic algorithms [32], and genetic programming [53, 54].

Each of these approaches represents an active area of research unto itself, with entire conferences and hundreds of papers published every year.

This thesis advances an approach that is different from, orthogonal to, and complementary to each of the approaches just mentioned. At a high level, the goal of this thesis

is to improve the performance of existing heuristics for NP-hard problems by adapting the heuristics to the problem instance(s) they are run on. In relationship to the three techniques just discussed, the approach taken in this thesis lies at a level of abstraction somewhere in between the problem-specific engineering approaches and the black-box approaches.

A distinguishing feature of this work is that the adaptation can be performed on-the-fly, while solving a sequence of problem instances. Our online algorithms come with rigorous performance guarantees, stated either as regret bounds or as a competitive ratio.

In addition to proving theoretical guarantees, we evaluate our algorithms experimentally using state-of-the-art solvers in a wide array of real-world problem domains. In many cases, our algorithms are able to automatically produce new solvers that significantly outperform the existing ones.

1.1 Summary

This thesis is organized into six chapters.

- Chapter 1 is the introduction.
- Chapter 2, "Online Algorithms for Maximizing Submodular Functions", develops algorithms for solving an online resource allocation problem that generalizes several previously-studied online problems. The algorithms developed in Chapter 2 form the basis for many of the experimental and theoretical results in Chapter 3.
- Chapter 3, "Combining Multiple Heuristics Online", presents techniques for combining multiple problem-solving algorithms into an improved algorithm by interleaving the execution of the algorithms and, if the algorithms are randomized, periodically restarting them with a fresh random seed. An important feature of the work presented in this chapter is that a schedule for interleaving and restarting the algorithms can be learned on-the-fly while solving a sequence of problems.
- Chapter 4, "Using Decision Procedures Efficiently for Optimization", presents techniques for improving the performance of algorithms that solve an optimization problem by making a sequence of calls to an algorithm for the corresponding decision problem.
- Chapter 5, "The Max k-Armed Bandit Problem", studies a variant of the classical multi-armed bandit problem in which the goal is to maximize the *maximum* payoff

received, rather than the sum of the payoffs. Algorithms for solving the max k-armed bandit problem can be used to improve the performance of *multi-start* heuristics, which obtain a solution to an optimization problem by performing a number of independent runs of a randomized heuristic and returning the best solution obtained.

- Chapter 6 is the conclusion.

In the subsections that follow we formally define the problems considered in chapters 2 through 5, discuss the motivation for studying each problem, and summarize the main theoretical and experimental results. Some of the text in these subsections is duplicated in the introductory sections of the corresponding chapters.

The results in this thesis are based in part on five conference papers [78, 79, 80, 82, 83] and a working paper [76].

1.1.1 Online algorithms for maximizing submodular functions

In this chapter we develop algorithms for solving a class of online resource allocation problems, which can be described formally as follows. We are given as input a set $\mathcal{V}$ of activities. A pair $(v, \tau) \in \mathcal{V} \times \mathbb{R}_{>0}$ is called an *action*, and specifies that time τ is to be invested in activity v. A *schedule* is a sequence of actions. We denote by $\mathcal{S}$ the set of all schedules. A *job* is a function $f : \mathcal{S} \to [0, 1]$, where for any $S \in \mathcal{S}$, $f(S)$ equals the proportion of some task that is accomplished after performing the sequence of actions S. We require that a job f satisfy the following conditions (here $\oplus$ is the concatenation operator):

1. (monotonicity) for any schedules $S_1, S_2 \in \mathcal{S}$, we have $f(S_1) \leq f(S_1 \oplus S_2)$ and $f(S_2) \leq f(S_1 \oplus S_2)$.

2. (submodularity) for any schedules $S_1, S_2 \in \mathcal{S}$ and any action $a \in \mathcal{A}$,

$$f(S_1 \oplus S_2 \oplus \langle a \rangle) - f(S_1 \oplus S_2) \leq f(S_1 \oplus \langle a \rangle) - f(S_1) \,. \tag{1.1}$$

We will evaluate schedules in terms of two objectives. The first objective is to minimize

$$c\,(f, S) = \int_{t=0}^{\infty} 1 - f\left(S_{\langle t \rangle}\right) dt \tag{1.2}$$

where $S_{\langle t \rangle}$ is the schedule that results from truncating schedule S at time t. For example if $S = \langle (h_1, 3), (h_2, 3) \rangle$ then $S_{\langle 5 \rangle} = \langle (h_1, 3), (h_2, 2) \rangle$. We refer to $c\,(f, S)$ as the *cost* of S.

The second objective is to maximize $f\left(S_{\langle T\rangle}\right)$ for some fixed $T > 0$. We refer to $f\left(S_{\langle T\rangle}\right)$ as the coverage of S at time T.

In the online setting, an arbitrary sequence $\langle f_1, f_2, \ldots, f_n\rangle$ of jobs arrive one at a time, and we must finish each job (via some schedule) before moving on to the next job. When selecting a schedule S_i to use to finish job f_i, we have knowledge of the previous jobs $f_1, f_2, \ldots, f_{i-1}$ but we have no knowledge of f_i itself or of any subsequent jobs. In this setting we develop schedule-selection strategies that minimize *regret*, which is a measure of the difference between the average cost (or average coverage) of the schedules produced by our online algorithm and that of the best single schedule (in hindsight) for the given sequence of jobs.

To understand the rationale for studying these two problems, consider the following example. Let each activity v represent a randomized algorithm for solving some decision problem, and let the action (v, τ) represent running the algorithm (with a fresh random seed) for time τ. Fix some particular instance of the decision problem, and for any schedule S, let $f(S)$ be the probability that one (or more) of the runs in the sequence S yields a solution to the instance. We show in §2.1.4 that f satisfies the conditions required of a job. Then $f(S_{\langle T\rangle})$ is (by definition) the probability that performing the runs in schedule S yields a solution to the problem instance in time $\leq T$. For any non-negative random variable X, we have $\mathbb{E}[X] = \int_{t=0}^{\infty} \mathbb{P}[X > t]\, dt$. Thus $c(f, S)$ is the *expected* time that elapses before a solution is obtained.

Under each of the two objectives just defined, the problem introduced in this chapter generalizes a number of previously-studied problems. The problem of minimizing $c(f, S)$ generalizes MIN-SUM SET COVER [26], PIPELINED SET COVER [44, 64], the problem of constructing efficient sequences of trials [22], the problem of constructing task-switching schedules [73, 78], and the problem of constructing restart schedules [35, 61, 79]. The problem of maximizing $f(S_{\langle T\rangle})$ for some fixed $T > 0$ generalizes the problem of maximizing a monotone submodular set function subject to a knapsack constraint [56, 84], which in turns generalizes BUDGETED MAXIMUM COVERAGE [49] and MAX k-COVERAGE [65]. Prior to our work, many of these problems had only been considered in an offline setting. For the problems that had been considered in an online setting, the online algorithms presented in this chapter provide new and stronger guarantees.

We now summarize the main technical contributions of this chapter.

We first consider the problem of computing an optimal schedule in an offline setting, given black-box access to the job f. As immediate corollaries of existing results [24, 26], we obtain that for any $\epsilon > 0$, (*i*) achieving an approximation ratio of $4-\epsilon$ for the problem of minimizing $c(f, S)$ is NP-hard and (*ii*) achieving an approximation ratio of $1-\frac{1}{e}+\epsilon$ for the

problem of maximizing $f(S_{\langle T \rangle})$ is NP-hard. Building on and generalizing previous work [26, 84], we then present an offline greedy approximation algorithm that simultaneously achieves the optimal approximation ratios (of 4 and $1 - \frac{1}{e}$, respectively) for each of these two problems.

We then consider the online setting. In this setting we provide an online algorithm whose worst-case performance approaches that of the offline greedy algorithm asymptotically (as the number of jobs approaches infinity). Assuming P $\neq$ NP, this guarantee is essentially the best possible among online algorithms that make decisions in polynomial time.

Our online algorithms can be used in several different feedback settings. We first consider the feedback setting in which, after using schedule S_i to complete job f_i, we receive complete access to f_i. We then consider more limited feedback settings in which: (*i*) to receive access to f_i we must pay a price C, which is added to the regret, (*ii*) we only observe $f_i\left(S_{i\langle t \rangle}\right)$ for each $t \geq 0$, and (*iii*) we only observe $f_i\left(S_i\right)$. These limited feedback settings arise naturally in the applications discussed in the next chapter.

1.1.2 Combining multiple heuristics online

Many important computational problems are NP-hard and thus seem unlikely to admit algorithms with provably good worst-case performance, yet must be solved as a matter of practical necessity. For many of these problems, heuristics have been developed that perform much better in practice than a worst-case analysis would guarantee. Nevertheless, the behavior of a heuristic on a previously unseen problem instance can be difficult to predict in advance. The running time of a heuristic may vary by orders of magnitude across seemingly similar problem instances or, if the heuristic is randomized, across multiple runs on a single instance that use different random seeds [33, 38]. For this reason, after running a heuristic unsuccessfully for some time one might decide to suspend the execution of that heuristic and start running a different heuristic (or the same heuristic with a different random seed).

In this chapter we consider the problem of allocating CPU time to various heuristics so as to minimize the time required to solve one or more instances of a decision problem. We consider the problem of selecting an appropriate schedule in three settings: *offline*, *learning-theoretic*, and *online*. The results in this chapter significantly generalize and extend previous work on *algorithm portfolios* [33, 38, 68, 73, 90] and *restart schedules* [31, 35, 61].

The problem considered in this chapter can be described formally as follows. We are

given as input a set $\mathcal{H}$ of (randomized) algorithms for solving some decision problem. Given a problem instance, each $h \in \mathcal{H}$ is capable of returning a provably correct "yes" or "no" answer to the problem, but the time required for a given $h \in \mathcal{H}$ to return an answer depends both on the problem instance and on the random seed (and may be infinite). We solve each problem instance by interleaving the execution of the heuristics according to some schedule. Consistent with the framework of Chapter 2, we consider schedules of the form

$$S = \langle (h_1, \tau_1), (h_2, \tau_2), \ldots \rangle$$

where each pair (h_i, τ_i) specifies that time τ_i is to be invested in heuristic h_i.

We allow each heuristic h to be executed in one of two models. If h is executed in the *restart model*, then each action (h, τ) represents an independent run of h with a fresh random seed. If h is executed in the *suspend-and-resume* model, then each action (h, τ) represents continuing a single run of h for an additional τ time units.

This class of schedules includes both *task-switching schedules* [73] and *restart schedules* [61] as special cases. A *task-switching schedule* is a schedule that executes all heuristics in the suspend-and-resume model. A restart schedule is a schedule for a single randomized heuristic (i.e., $|\mathcal{H}| = 1$), executed in the restart model.

Motivations

To appreciate the power of task-switching schedules, consider Table 1.1, which shows the behavior of the top two solvers from the industrial track of the 2007 SAT competition on three of the competition benchmarks.

Table 1.1: Behavior of two solvers on instances from the 2007 SAT competition.

Instance	`Rsat` **CPU (s)**	`picosat` **CPU (s)**
industrial/anbulagan/medium-sat/dated-10-13-s.cnf	45	28
industrial/babic/dspam/dspam_dump_vc1081.cnf	3	$\geq$ 10000
industrial/grieu/vmpc_31.cnf	$\geq$ 10000	238

On these benchmarks, interleaving the execution of the solvers according to an appropriate schedule can dramatically improve average-case running time. Indeed, in this case

simply running the two solvers in parallel (e.g., at equal strength on a single processor) would reduce the average-case running time by orders of magnitude.

To appreciate the power of restart schedules, consider Figure 1, which depicts the run length distribution of the SAT solver `satz-rand` on a Boolean formula derived from a logistics planning benchmark. When run on this formula, `satz-rand` exhibits a heavy-tailed run length distribution. There is about a 20% chance of solving the problem after running for 2 seconds, but also a 20% chance that a run will not terminate after having run for 1000 seconds. Restarting the solver every 2 seconds reduces the mean running time by more than an order of magnitude.

Figure 1.1: Run length distribution of `satz-rand` on a formula derived from a logistics planning benchmark.

Results

We now summarize the main technical results of this chapter. As already mentioned, this chapter considers the schedule-selection problem in three settings: *offline*, *learning-theoretic*, and *online*.

In the offline setting we are given as input the run length distribution of each $h \in \mathcal{H}$ for each problem instance in a set of instances, and wish to compute a schedule with minimum average (expected) running time over the instances in the set. In this setting, the greedy algorithm from Chapter 2 gives a 4 approximation to the optimal schedule and, for any $\epsilon > 0$, computing an $4 - \epsilon$ approximation is NP-hard. We also give exact and approximation

algorithms based on shortest paths that are able to compute an α-approximation to the optimal schedule for any $\alpha \geq 1$, but whose running time is exponential as a function of $|\mathcal{H}|$.

In the learning-theoretic setting, we draw training instances from a fixed distribution, compute an (approximately) optimal schedule for the training instances, and then use that schedule to solve additional test instances drawn from the same distribution. In this setting, we give bounds on the number of training instances required to learn a schedule that is *probably approximately correct*.

In the online setting we are fed a sequence of problem instances one at a time and must obtain a solution to each instance before moving on to the next. In this setting we show that the online greedy algorithm from Chapter 2 converges to a 4 approximation to the best fixed schedule for the instance sequence, and requires decision-making time polynomial in $|\mathcal{H}|$. We also present online shortest paths algorithms that, for any $\alpha \geq 1$, can be guaranteed to converge to an α-approximation to the best fixed schedule, but these online algorithms require decision-making time exponential in $|\mathcal{H}|$.

Our results in each of these three settings can be extended in two ways. First, our algorithms can be applied in an interesting way to heuristics for optimization rather than decision problems. Second, quickly-computable features of problem instances can be exploited in a principled way to improve the schedule selection process.

This chapter concludes with an experimental evaluation of the techniques developed for both the offline and online settings. The main results of our experimental evaluation can be summarized as follows.

1. Using data from recent solver competitions, we show that schedules computed by our algorithms can be used improve the performance of state-of-the-art solvers in several problem domains, including Boolean satisfiability, A.I. planning, constraint satisfaction, and theorem proving.

2. We apply our algorithms to optimization problems (as opposed to decision problems), and demonstrate that they can be used to improve the performance of state-of-the-art algorithms for pseudo-Boolean optimization (also known as zero-one integer programming).

3. We show that additional performance improvements can be obtained by using instance-specific features to tailor the choice of schedule to a particular problem instance.

4. We use our offline algorithms to construct a restart schedule for the SAT solver `satz-rand` that improves its performance on an ensemble of problem instances

derived from logistics planning benchmarks.

1.1.3 Using decision procedures efficiently for optimization

Optimization problems are often solved by making repeated calls to a decision procedure that answers questions of the form "Does there exist a solution with cost at most k?". Each query to the decision procedure can be represented as a pair $\langle k, t \rangle$, where t is a bound on the CPU time the decision procedure may consume in answering the question. The result of a query is either a (provably correct) "yes" or "no" answer or a timeout. A *query strategy* is a rule for determining the next query $\langle k, t \rangle$ as a function of the responses to previous queries.

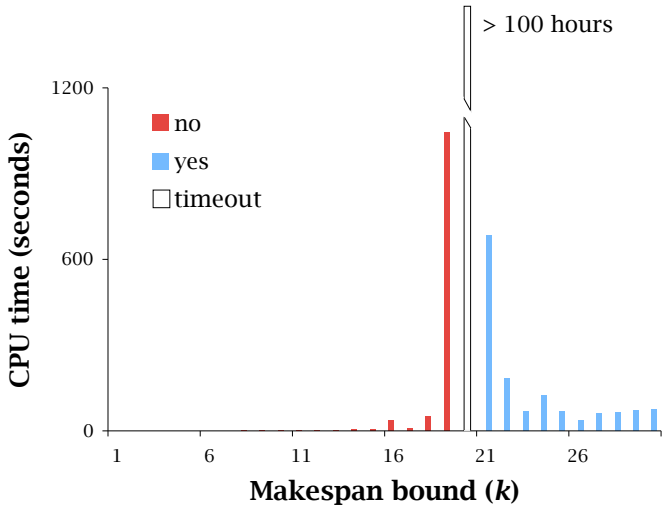

Figure 1.2: Behavior of the SAT solver `siege` running on formulae generated by `SATPLAN` to solve instance `p17` from the `pathways` domain of the 2006 International Planning Competition.

One optimization algorithm of this form is `SATPLAN`, a state-of-the-art algorithm for classical planning. `SATPLAN` finds a minimum-length plan by making a series of calls to a SAT solver, where each call determines whether there exists a feasible plan of makespan $\leq k$ (where the value of k varies across calls). The original version of `SATPLAN` uses the *ramp-up* query strategy, which simply executes the queries $\langle 1, \infty \rangle, \langle 2, \infty \rangle, \langle 3, \infty \rangle, \ldots$ in sequence (stopping as soon as a "yes" answer is obtained).

The motivation for the work in this chapter is that the choice of query strategy often has a dramatic effect on the time required to obtain a (provably) approximately optimal solution. As an example, consider Figure 1.2, which shows the CPU time required by the query $\langle k, \infty \rangle$ as a function of k, on a particular planning benchmark instance. On this instance, using the ramp-up query strategy requires one to invest over 100 hours of CPU time before obtaining a feasible plan. On the other hand, executing the queries $\langle 18, \infty \rangle$ and $\langle 23, \infty \rangle$ takes less than two minutes and yields a plan whose makespan is provably at most $\frac{23}{18+1} \approx 1.21$ times optimal.

This chapter presents both a theoretical and an experimental study of query strategies. We consider the problem of devising query strategies in two settings. In the *single-instance* setting, we are confronted with a single optimization problem, and wish to obtain an (approximately) optimal solution as quickly as possible. In the *multiple-instance* setting, we use the same decision procedure to solve a number of optimization problems, and our goal is to learn from experience in order to improve performance.

In the single-instance setting, we are interested in minimizing the CPU time required to obtain a given upper or lower bound on OPT, where OPT is the minimum cost of any solution. Fix a problem instance, and let $\tau(k)$ denote the CPU time required by the decision procedure when run on input k. We define the *competitive ratio* of a query strategy (on that instance) as the maximum, over all k, of the time required by the query strategy to determine what side of OPT that k is on (either by obtaining a "yes" answer for some $k' \leq k$, or by obtaining a "no" answer for some $k' \geq k$), divided by $\tau(k)$. We analyze query strategies in terms of their competitive ratio on the worst-case instance within some well-defined class of instances.

The competitive ratio of our query strategies will depend on the behavior of the function τ (as just mentioned, $\tau(k)$ is the CPU time required by the decision procedure when run on input k). For most decision procedures used in practice, we expect $\tau(k)$ to be an increasing function for $k \leq \mathsf{OPT}$ and a decreasing function for $k \geq \mathsf{OPT}$ (e.g., see Figure 1.2), and our query strategies are designed to take advantage of this behavior. More specifically, our query strategies are designed to work well when τ is close to its *hull*, which is the function

$$\mathrm{hull}(k) = \min \left\{ \max_{k_0 \leq k} \tau(k_0), \max_{k_1 \geq k} \tau(k_1) \right\} .$$

Figure 4.2 gives an example of a function τ (gray bars) and its hull (dots). Note that τ and hull are identical if τ is monotonically increasing (or monotonically decreasing), or if there exists a K such that τ is monotonically increasing for $k \leq K$ and monotonically decreasing for $k > K$.

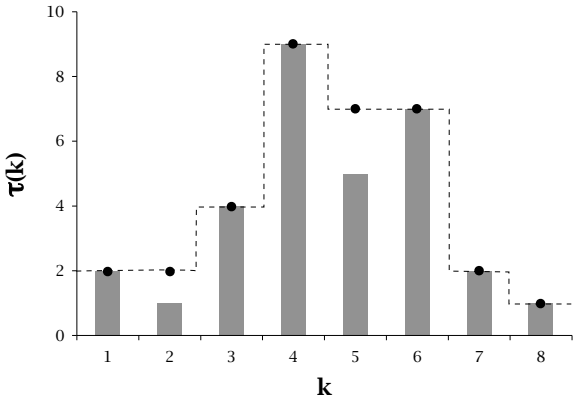

Figure 1.3: A function τ (gray bars) and its hull (dots).

We measure the discrepancy between τ and its hull in terms of the quantity

$$\Delta = \max_k \frac{\text{hull}(k)}{\tau(k)}$$

which we refer to as the *stretch* of τ. The instance depicted in Figure 4.2 has a stretch of 2 because $\tau(2) = 1$ while $\text{hull}(2) = 2$.

In the single-instance setting, our main result is a query strategy S_2 whose worst-case competitive ratio is $O(\Delta \log U)$, where U is the difference between the initial upper and lower bounds on OPT. S_2 makes use of a form of guessing-and-doubling in combination with a two-sided binary search. We prove a matching lower bound, showing that any query strategy has a competitive ratio $\Omega(\Delta \log U)$ on some instance. We also show that, in the absence of any assumptions about Δ, a trivial query strategy S_1 based on guessing-and-doubling obtains a worst-case competitive ratio that is $O(U)$, and we prove a matching $\Omega(U)$ lower bound.

In the multiple-instance setting, we prove that computing an optimal query strategy is NP-hard, and discuss how algorithms from machine learning theory can be used to learn an appropriate query on-the-fly while solving a sequence of problems.

In the experimental section of this chapter, we use the query strategy S_2 to create a modified version of `SATPLAN` that finds (provably) approximately optimal plans more quickly than the original version of `SATPLAN` (which uses the ramp-up query strategy). We also create a modified version of a branch and bound algorithm for job shop scheduling that yields improved upper and lower bounds relative to the original algorithm. In the course of the latter experiments we develop a simple method for applying query strategies

to branch and bound algorithms, which seems likely to be useful in other domains besides job shop scheduling.

1.1.4 The max k-armed bandit problem

The max k-armed bandit problem [19, 21] can be described as follows. Imagine that you find yourself in the following unusual casino. The casino contains k slot machines. Each machine has an arm that, when pulled, yields a payoff drawn from a fixed (but unknown) distribution. You are given n tokens to use in playing the machines, and you may decide how to spend these tokens adaptively based on the payoffs you receive from playing the various machines. The catch is that, when you leave the casino, you only get to keep the *maximum* of the payoffs you received on any individual pull. The max k-armed bandit problem differs from the well-studied classical k-armed bandit problem in that one seeks to optimize the maximum payoff received, rather than the sum of the payoffs.

Our motivation for studying this problem is to boost the performance of *multi-start* heuristics, which obtain a solution to an optimization problem by performing a number of independent runs of a randomized heuristic and returning the best solution obtained. Despite their simplicity, multi-start heuristics are used widely in practice, and represent the state of the art in a number of domains [14, 20, 27]. A max k-armed bandit strategy can be used to distribute trials among different multi-start heuristics or among different parameter settings for the same multi-start heuristic. Previous work has demonstrated the effectiveness of such an approach on the RCPSP/max, a difficult real-world scheduling problem [19, 21, 82].

In this chapter our goal is to develop strategies for the max k-armed bandit problem that minimize *regret*, which we define to be the difference between the (expected) maximum payoff our strategy receives and that of the best pure strategy, where a pure strategy is one that plays the same arm every time. It is not difficult to show that regret-minimization is hopeless in the absence of any assumptions about the payoff distributions. As a simple example, imagine that all payoffs are either 0 or 1, that $k-1$ of the arms always return a payoff of 0, and that one randomly-selected "good" arm returns a payoff of 1 with probability $\frac{1}{n}$. In this case, we show that one cannot obtain an expected maximum payoff larger than $\frac{1}{k}$ after n pulls, whereas the pure strategy that invests all n pulls on the "good" arm obtains expected maximum payoff $1-(1-\frac{1}{n})^n \approx 1-\frac{1}{e}$.

We present two strategies for solving the max k-armed bandit problem. The first strategy, Threshold Ascent, is designed to work well when the payoff distributions have certain characteristics which we expect to be present in cases of practical interest. Roughly speak-

ing, Threshold Ascent will work best when the following two criteria are satisfied.

1. There is a (relatively low) threshold $t_{critical}$ such that, for all $t > t_{critical}$, the arm that is most likely to yield a payoff $> t$ is the same as the arm most likely to yield a payoff $> t_{critical}$. Call this arm i^*.

2. As t increases beyond $t_{critical}$, there is a growing gap between the probability that arm i^* yields a payoff $> t$ and the corresponding probability for other arms. Specifically, if we let $p_i(t)$ denote the probability that the i^{th} arm returns a payoff $> t$, the ratio $\frac{p_{i^*}(t)}{p_i(t)}$ should increase as a function of t for $t > t_{critical}$, for any $i \neq i^*$.

Figure 1.4 illustrates a set of two payoff distributions that satisfy these assumptions.

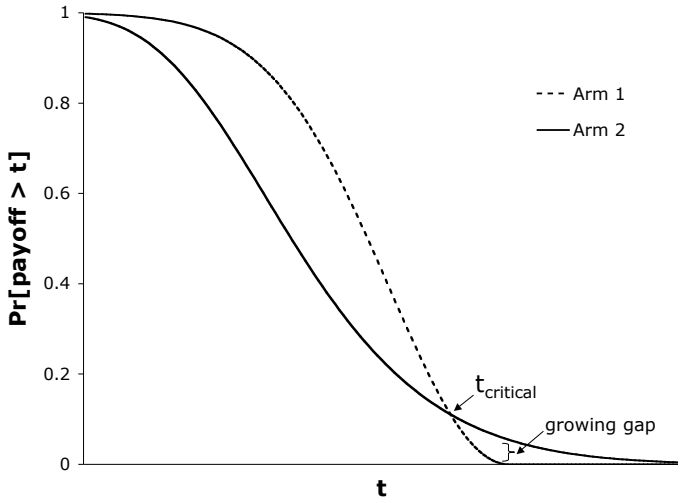

Figure 1.4: A max k-armed bandit instance on which Threshold Ascent should perform well.

The idea of Threshold Ascent is very simple. Threshold Ascent attempts to maximize the number of payoffs $> T$ that it receives, where T is a threshold that is gradually increased over time. For any fixed T, this goal is accomplished by mapping payoffs $> T$ to 1 and mapping payoffs $\leq T$ to zero, then treating the problem as an instance of the classical k-armed bandit problem (where the goal is to maximize the sum of the payoffs received).

As T increases, non-zero payoffs become increasingly rare, and thus we would like to have an algorithm for solving the classical k-armed bandit problem that works well when

the mean payoff of each arm is very small. Toward this end, we design and analyze a new algorithm for the classical k-armed bandit problem called Chernoff Interval Estimation, which yields improved regret bounds when each arm has a small mean payoff.

In the experimental section of this chapter, we demonstrate the effectiveness of Threshold Ascent by using it to select among multi-start heuristics for the RCPSP/max, a difficult real-world scheduling problem. We find that Threshold Ascent (*i*) performs better than any of the multi-start heuristics performs in isolation, and (*ii*) outperforms the recent QD-BEACON max k-armed bandit algorithm of Cicirello and Smith [19, 21].

Following the lead of Cicirello and Smith [19, 21], we also consider the special case where each payoff distribution is a generalized extreme value (GEV) distribution. The motivation for studying this special case is the Extremal Types Theorem [23], which singles out the GEV as the limiting distribution of the maximum of a large number of independent identically distributed (i.i.d.) random variables. Roughly speaking, one can think of the Extremal Types Theorem as an analogue of the Central Limit Theorem. Just as the Central Limit Theorem states that the average of a large number of i.i.d. random variables converges in distribution to a Gaussian, the Extremal Types Theorem states that the maximum of a large number of i.i.d. random variables converges in distribution to a GEV. We provide a no-regret strategy for this special case, generalizing and improving upon earlier theoretical work by Cicirello & Smith [19, 21].

Chapter 2

Online Algorithms for Maximizing Submodular Functions

2.1 Introduction

In this chapter we present algorithms for solving a specific class of online resource allocation problems. Our online algorithms can be applied in environments where abstract *jobs* arrive one at a time, and one can complete the jobs by investing time in a number of abstract *activities*. Provided that the jobs and activities satisfy certain technical conditions, our online algorithm is guaranteed to perform almost as well as any fixed schedule for investing time in the various activities, according to two natural measures of performance. As we discuss further in §2.1.5, our problem formulation captures a number of previously-studied problems, including selection of algorithm portfolios [33, 38], selection of restart schedules [35, 61], and database query optimization [9, 64]. Additionally, this online algorithm forms the basis for many of the theoretical and experimental results in Chapter 3, "Combining Multiple Heuristics Online".

2.1.1 Formal setup

The problem considered in this chapter can be defined as follows. We are given as input a finite set $\mathcal{V}$ of *activities*. A pair $(v, \tau) \in \mathcal{V} \times \mathbb{R}_{>0}$ is called an *action*, and represents spending time τ performing activity v. A *schedule* is a sequence of actions. We use $\mathcal{S}$ to denote the set of all schedules. A *job* is a function $f : \mathcal{S} \to [0, 1]$, where for any schedule $S \in \mathcal{S}$, $f(S)$ represents the proportion of some task that is accomplished by performing

the sequence of actions S. We require that a job f satisfy the following conditions (here $\oplus$ is the concatenation operator):

1. (monotonicity) for any schedules $S_1, S_2 \in \mathcal{S}$, we have $f(S_1) \leq f(S_1 \oplus S_2)$ and $f(S_2) \leq f(S_1 \oplus S_2)$.

2. (submodularity) for any schedules $S_1, S_2 \in \mathcal{S}$ and any action $a \in \mathcal{V} \times \mathbb{R}_{>0}$,

$$f(S_1 \oplus S_2 \oplus \langle a \rangle) - f(S_1 \oplus S_2) \leq f(S_1 \oplus \langle a \rangle) - f(S_1)\,. \tag{2.1}$$

We will evaluate schedules in terms of two objectives. The first objective is to maximize $f(S)$ subject to the constraint $\ell(S) \leq T$, for some fixed $T > 0$, where $\ell(S)$ equals the sum of the durations of the actions in S. For example if $S = \langle (v_1, 3), (v_2, 3) \rangle$, then $\ell(S) = 6$. We refer to this problem as BUDGETED MAXIMUM SUBMODULAR COVERAGE (the origin of this terminology is explained in §2.2).

The second objective is to minimize the *cost* of a schedule, which we define as

$$c(f, S) = \int_{t=0}^{\infty} 1 - f\left(S_{\langle t \rangle}\right) dt \tag{2.2}$$

where $S_{\langle t \rangle}$ is the schedule that results from truncating schedule S at time t. For example if $S = \langle (v_1, 3), (v_2, 3) \rangle$ then $S_{\langle 5 \rangle} = \langle (v_1, 3), (v_2, 2) \rangle$.[1] One way to interpret this objective is to imagine that $f(S)$ is the probability that some desired event occurs as a result of performing the actions in S. For any non-negative random variable X, we have $\mathbb{E}[X] = \int_{t=0}^{\infty} \mathbb{P}[X > t]\, dt$. Thus $c(f, S)$ is the expected time we must wait before the event occurs if we execute actions according to the schedule S. We refer to the problem of computing a schedule that minimizes $c(f, S)$ as MIN-SUM SUBMODULAR COVER.

In the online setting, an arbitrary sequence $\langle f_1, f_2, \ldots, f_n \rangle$ of jobs arrive one at a time, and we must finish each job (via some schedule) before moving on to the next job. When selecting a schedule S_i to use to finish job f_i, we have knowledge of the previous jobs $f_1, f_2, \ldots, f_{i-1}$ but we have no knowledge of f_i itself or of any subsequent jobs. In this setting our goal is to develop schedule-selection strategies that minimize *regret*, which is a measure of the difference between the average cost (or average coverage) of the schedules produced by our online algorithm and that of the best single schedule (in hindsight) for the given sequence of jobs.

The following example illustrates these definitions.

[1] More formally, if $S = \langle a_1, a_2, \ldots \rangle$, where $a_i = (v_i, \tau_i)$, then $S_{\langle t \rangle} = \langle a_1, a_2, \ldots, a_{k-1}, a_k, (v_{k+1}, \tau') \rangle$, where k is the largest integer such that $\sum_{i=1}^{k} \tau_i < t$ and $\tau' = t - \sum_{i=1}^{k} \tau_i$.

Example 1. Let each activity v represent a randomized algorithm for solving some decision problem, and let the action (v, τ) represent running the algorithm (with a fresh random seed) for time τ. Fix some particular instance of the decision problem, and for any schedule S, let $f(S)$ be the probability that one (or more) of the runs in the sequence S yields a solution to that instance. So $f(S_{\langle T \rangle})$ is (by definition) the probability that performing the runs in schedule S yields a solution to the problem instance in time $\leq T$, while $c\,(f, S)$ is the *expected* time that elapses before a solution is obtained. It is clear that $f(S)$ satisfies the monotonicity condition required of a job, because adding runs to the sequence S can only increase the probability that one of the runs is successful. The fact that f is submodular can be seen as follows. For any schedule S and action a, $f(S \oplus \langle a \rangle) - f(S)$ equals the probability that action a succeeds after every action in S has failed, which can also be written as $(1 - f(S)) \cdot f(\langle a \rangle)$. This, together with the monotonicity of f, implies that for any schedules S_1, S_2 and any action a, we have

$$\begin{aligned} f(S_1 \oplus S_2 \oplus \langle a \rangle) - f(S_1 \oplus S_2) &= (1 - f(S_1 \oplus S_2)) \cdot f(\langle a \rangle) \\ &\leq (1 - f(S_1)) \cdot f(\langle a \rangle) \\ &= f(S_1 \oplus \langle a \rangle) - f(S_1) \end{aligned}$$

so f is submodular.

2.1.2 Sufficient conditions

In some cases of practical interest, f will not satisfy the submodularity condition but will still satisfy weaker conditions that are sufficient for our results to carry through.

In the offline setting, our results will hold for any function f that satisfies the monotonicity condition and, additionally, satisfies the following condition (we prove in §2.3 that any submodular function satisfies this weaker condition).

Condition 1. For any $S_1, S \in \mathcal{S}$,

$$\frac{f(S_1 \oplus S) - f(S_1)}{\ell\,(S)} \leq \max_{(v,\tau) \in \mathcal{V} \times \mathbb{R}_{>0}} \left\{ \frac{f(S_1 \oplus \langle (v, \tau) \rangle) - f(S_1)}{\tau} \right\} .$$

Recall that $\ell\,(S)$ equals the sum of the durations of the actions in S. Informally, Condition 1 says that the increase in f per unit time that results from performing a sequence of actions S is always bounded by the maximum, over all actions (v, τ), of the increase in f per unit time that results from performing that action.

In the online setting, our results will apply if each function f_i in the sequence $\langle f_1, f_2, \ldots, f_n \rangle$ satisfies the monotonicity condition and, additionally, the sequence as a whole

satisfies the following condition (we prove in §2.4 that if each f_i is a job, then this condition is satisfied).

Condition 2. For any sequence $S_1, S_2, \ldots, S_n$ of schedules and any schedule S,

$$\frac{\sum_{i=1}^{n} f_i(S_i \oplus S) - f_i(S_i)}{\ell(S)} \leq \max_{(v,\tau)\in\mathcal{V}\times\mathbb{R}_{>0}} \left\{ \frac{\sum_{i=1}^{n} f_i(S_i \oplus \langle(v,\tau)\rangle) - f_i(S_i)}{\tau} \right\}.$$

As we discuss further in Chapter 3, this generality allows us to handle jobs similar to the job defined in Example 1, but where an action (v, τ) may represent *continuing* a run of algorithm v for an additional τ time units (rather than running v with a fresh random seed). Note that the function f defined in Example 1 is no longer submodular when actions of this form are allowed.

2.1.3 Summary of results

We first consider the offline problems BUDGETED MAXIMUM SUBMODULAR COVERAGE and MIN-SUM SUBMODULAR COVER. As immediate consequences of existing results [24, 26], we find that, for any $\epsilon > 0$, (*i*) achieving an approximation ratio of $4 - \epsilon$ for MIN-SUM SUBMODULAR COVER is NP-hard and (*ii*) achieving an approximation ratio of $1 - \frac{1}{e} + \epsilon$ for BUDGETED MAXIMUM SUBMODULAR COVERAGE is NP-hard. We then present a greedy approximation algorithm that simultaneously achieves the optimal approximation ratio of 4 for MIN-SUM SUBMODULAR COVER and the optimal approximation ratio of $1 - \frac{1}{e}$ for BUDGETED MAXIMUM SUBMODULAR COVERAGE, building on and generalizing previous work on special cases of these two problems [26, 84].

The main contribution of this chapter, however, is to address the online setting. In this setting we provide an online algorithm whose worst-case performance approaches that of the offline greedy approximation algorithm asymptotically (as the number of jobs approaches infinity). More specifically, we analyze the online algorithm's performance in terms of "α-regret". For the cost-minimization objective, α-regret is defined as the difference between the average cost of the schedules selected by the online algorithm and α times the average cost of the optimal schedule for the given sequence of jobs. For the coverage-maximization objective, α-regret is the difference between α times the average coverage of the optimal fixed schedule and the average coverage of the schedules selected by the online algorithm. For the objective of minimizing cost, the online algorithm's 4-regret approaches zero as $n \to \infty$, while for the objective of maximizing coverage, its $1-\frac{1}{e}$ regret approaches zero as $n \to \infty$. Assuming P $\neq$ NP, these guarantees are essentially the best possible among online algorithms that make decisions in polynomial time.

Our online algorithms can be used in several different feedback settings. We first consider the feedback setting in which, after using schedule S_i to complete job f_i, we receive complete access to f_i. We then consider more limited feedback settings in which: (*i*) to receive access to f_i we must pay a price C, which is added to the regret, (*ii*) we only observe $f_i\left(S_{i\langle t\rangle}\right)$ for each $t \geq 0$, and (*iii*) we only observe $f_i\left(S_i\right)$.

We also prove tight information-theoretic lower bounds on 1-regret, and discuss exponential time online algorithms whose regret matches the lower bounds to within logarithmic factors. Interestingly, these lower bounds also match the upper bounds from our online greedy approximation algorithm up to logarithmic factors, although the latter apply to α-regret (for $\alpha = 4$ or $\alpha = 1 - \frac{1}{e}$) rather than 1-regret.

The results in this chapter are based on a working paper [76].

2.1.4 Problems that fit into this framework

We now discuss how a number of previously-studied problems fit into the framework of this chapter.

Special cases of BUDGETED MAXIMUM SUBMODULAR COVERAGE

The BUDGETED MAXIMUM SUBMODULAR COVERAGE problem introduced in this chapter is a slight generalization of the problem of maximizing a monotone submodular set function subject to a knapsack constraint [56, 84]. The only difference between the two problems is that, in the latter problem, $f(S)$ may only depend on the *set* of actions in the sequence S, and not on the order in which the actions appear. The problem of maximizing a monotone submodular set function subject to a knapsack constraint in turn generalizes BUDGETED MAXIMUM COVERAGE [49], which generalizes MAX k-COVERAGE [65].

Special cases of MIN-SUM SUBMODULAR COVER

The MIN-SUM SUBMODULAR COVER problem introduced in this chapter generalizes several previously-studied problems, including MIN-SUM SET COVER [26], PIPELINED SET COVER [44, 64], the problem of constructing efficient sequences of trials [22], and the problem of constructing restart schedules [35, 61, 79]. Specifically, these problems

can be represented in our framework by jobs of the form

$$f\left(\langle(v_1,\tau_1),(v_2,\tau_2),\ldots,(v_L,\tau_L)\rangle\right) = \frac{1}{n}\sum_{i=1}^{n}\left(1-\prod_{l=1}^{L}\left(1-p_i(v_l,\tau_l)\right)\right) . \tag{2.3}$$

This expression can be interpreted as follows: the job f consists of n subtasks, and $p_i(v,\tau)$ is the probability that investing time τ in activity v completes the i^{th} subtask. Thus, $f(S)$ is the expected fraction of subtasks that are finished after performing the sequence of actions in S. Assuming $p_i(v,\tau)$ is a non-decreasing function of τ for all i and v, it can be shown that any function f of this form satisfies the monotonicity and submodularity properties required of a job. In the special case $n = 1$, this follows from Example 1. In the general case $n > 1$, this follows from the fact (which follows immediately from the definitions) that any convex combination of jobs is a job.

The problem of computing restart schedules places no further restrictions on $p_i(v,\tau)$. PIPELINED SET COVER is the special case in which for each activity v there is an associated time τ_v, and $p_i(v,\tau) = 1$ if $\tau \geq \tau_v$ and $p_i(v,\tau) = 0$ otherwise. MIN-SUM SET COVER is the special case in which, additionally, $\tau_v = 1$ or $\tau_v = \infty$ for all $v \in \mathcal{V}$. The problem of constructing efficient sequences of trials corresponds to the case in which we are given a matrix q, and $p_i(v,\tau) = q_{v,i}$ if $\tau \geq 1$ and $p_i(v,\tau) = 0$ otherwise.

2.1.5 Applications

We now discuss applications of the results presented in this chapter. The first application, "Combining multiple heuristics online", is evaluated experimentally in Chapter 3. Evaluating the remaining applications is an interesting area of future work.

Combining multiple heuristics online

An *algorithm portfolio* [38] is a schedule for interleaving the execution of multiple (randomized) algorithms and periodically restarting them with a fresh random seed. Previous work has shown that combining multiple heuristics for NP-hard problems into a portfolio can dramatically reduce average-case running time [33, 38, 78]. In particular, algorithms based on chronological backtracking often exhibit heavy-tailed run length distributions, and periodically restarting them with a fresh random seed can reduce the mean running time by orders of magnitude [34]. Our algorithms can be used to learn an effective algorithm portfolio online, in the course of solving a sequence of problem instances. Chapter 3 considers this application in detail and presents an experimental evaluation.

Database query optimization

In database query processing, one must extract all the records in a database that satisfy every predicate in a list of one or more predicates (the conjunction of predicates comprises the query). To process the query, each record is evaluated against the predicates one at a time until the record either fails to satisfy some predicate (in which case it does not match the query) or all predicates have been examined. The order in which the predicates are examined affects the time required to process the query. Munagala *et al.* [64] introduced and studied a problem called PIPELINED SET COVER, which entails finding an evaluation order for the predicates that minimizes the average time required to process a record. As discussed in §2.1.4, PIPELINED SET COVER is a special case of MIN-SUM SUBMODULAR COVER. In the online version of PIPELINED SET COVER, records arrive one at a time and one may select a different evaluation order for each record. In our terms, the records are jobs and predicates are activities.

Sensor placement

Sensor placement is the task of assigning locations to a set of sensors so as to maximize the value of the information obtained (e.g., to maximize the number of intrusions that are detected by the sensors). Many sensor placement problems can be optimally solved by maximizing a monotone submodular set function subject to a knapsack constraint [55]. As discussed in §2.1.4, this problem is a special case of BUDGETED MAXIMUM SUBMODULAR COVERAGE. Our online algorithms could be used to select sensor placements when the same set of sensors is repeatedly deployed in an unknown or adversarial environment.

Viral marketing

Viral marketing infects a set of agents (e.g., individuals or groups) with an advertisement which they may pass on to other potential customers. Under a standard model of social network dynamics, the total number of potential customers that are influenced by the advertisement is a submodular function of the set of agents that are initially infected [48]. Previous work [48] gave an algorithm for selecting a set of agents to initially infect so as to maximize the influence of an advertisement, assuming the dynamics of the social network are known. In theory, our online algorithms could be used to adapt a marketing campaign to unknown or time-varying social network dynamics.

2.2 Related Work

As discussed in §2.1.4, the MIN-SUM SUBMODULAR COVER problem introduced in this chapter generalizes several previously-studied problems, including MIN-SUM SET COVER [26], PIPELINED SET COVER [44, 64], the problem of constructing efficient sequences of trials [22], and the problem of constructing restart schedules [61, 35, 79].

Several of these problems have been considered in the online setting. Munagala *et al.* [64] gave an online algorithm for PIPELINED SET COVER whose $O(\log |\mathcal{V}|)$-regret is $o(n)$, where n is the number of records (jobs). Babu *et al.* [9] and Kaplan *et al.* [44] gave online algorithms for PIPELINED SET COVER whose 4-regret is $o(n)$, but these bounds hold only in the special case where the jobs are drawn independently at random from a fixed probability distribution. The online setting in this chapter, where the sequence of jobs may be arbitrary, is more challenging from a technical point of view.

As already mentioned, BUDGETED MAXIMUM SUBMODULAR COVERAGE generalizes the problem of maximizing a monotone submodular set function subject to a knapsack constraint. Previous work gave offline greedy approximation algorithms for this problem [56, 84], which generalized earlier algorithms for BUDGETED MAXIMUM COVERAGE [49] and MAX k-COVERAGE [65]. To our knowledge, none of these three problems have previously been studied in an online setting.

It is worth pointing out that the online problems we consider here are quite different from online set cover problems that require one to construct a *single* collection of sets that cover each element in a sequence of elements that arrive online [1, 7]. Likewise, our work is orthogonal to work on online facility location problems [62].

The main technical contribution of this chapter is to convert some specific greedy approximation algorithms into online algorithms. Recently, Kakade *et al.* [41] gave a generic procedure for converting an α-approximation algorithm for a linear problem into an online algorithm whose α-regret is $o(n)$, and this procedure could be applied to the problems considered in this chapter. However, both the running time of their algorithm and the resulting regret bounds depend on the dimension of the linear problem, and a straightforward application of their algorithm leads to running time and regret bounds that are exponential in $|\mathcal{V}|$.

2.3 Offline Algorithms

In this section we consider the offline problems BUDGETED MAXIMUM SUBMODULAR COVERAGE and MIN-SUM SUBMODULAR COVER. In the offline setting, we are given as input a job $f : \mathcal{S} \to [0, 1]$. Our goal is to compute a schedule S that achieves one of two objectives: for BUDGETED MAXIMUM SUBMODULAR COVERAGE, we wish to maximize $f(S)$ subject to the constraint $\ell(S) \leq T$ (for some fixed $T > 0$), while for MIN-SUM SUBMODULAR COVER, we wish to minimize the cost $c(f, S)$.

The offline algorithms presented in this section will serve as the basis for the online algorithms we develop in the next section.

Note that we have defined the offline problem in terms of optimizing a *single* job. However, given a set $\{f_1, f_2, \ldots, f_n\}$, we can optimize average schedule cost (or coverage) by applying our offline algorithm to the job $f = \frac{1}{n}\sum_{i=1}^{n} f_i$ (as already mentioned, any convex combination of jobs is a job).

2.3.1 Computational complexity

Both of the offline problems considered in this chapter are NP-hard even to approximate. As discussed in §2.1.4, MIN-SUM SUBMODULAR COVER generalizes MIN-SUM SET COVER, and BUDGETED MAXIMUM SUBMODULAR COVERAGE generalizes MAX k-COVERAGE. In a classic paper, Feige proved that for any $\epsilon > 0$, acheiving an approximation ratio of $1 - \frac{1}{e} + \epsilon$ for MAX k-COVERAGE is NP-hard [24]. Recently, Feige, Lovász, and Tetali [26] introduced MIN-SUM SET COVER and proved that for any $\epsilon > 0$, achieving a $4 - \epsilon$ approximation ratio for MIN-SUM SET COVER is NP-hard. These observations immediately yield the following theorems.

Theorem 1. *For any $\epsilon > 0$, achieving a $1 - \frac{1}{e} + \epsilon$ approximation ratio for* BUDGETED MAXIMUM SUBMODULAR COVERAGE *is NP-hard.*

Theorem 2. *For any $\epsilon > 0$, achieving a $4 - \epsilon$ approximation ratio for* MIN-SUM SUBMODULAR COVER *is NP-hard.*

2.3.2 Greedy approximation algorithm

In this section we present a greedy approximation algorithm that can be used to achieve a 4 approximation for MIN-SUM SUBMODULAR COVER and a $1 - \frac{1}{e}$ approximation for

BUDGETED MAXIMUM SUBMODULAR COVERAGE. By Theorems 1 and 2, achieving a better approximation ratio for either problem is NP-hard.

Consider the schedule defined by the following simple greedy rule. Let $G = \langle g_1, g_2, \ldots \rangle$ be the schedule defined inductively as follows: $G_1 = \langle\rangle$, $G_j = \langle g_1, g_2, \ldots, g_{j-1} \rangle$ for $j > 1$, and

$$g_j = \operatorname*{arg\,max}_{(v,\tau) \in \mathcal{V} \times \mathbb{R}_{>0}} \left\{ \frac{f\left(G_j \oplus \langle (v, \tau) \rangle\right) - f\left(G_j\right)}{\tau} \right\} . \tag{2.4}$$

That is, G is constructed by greedily appending an action (v, τ) to the schedule so as to maximize the resulting increase in f per unit time.

Once we reach a j such that $f(G_j) = 1$, we may stop adding actions to the schedule. In general, however, G may contain an infinite number of actions. For example, if each action (v, τ) represents running a Las Vegas algorithm v for time τ and $f(S)$ is the probability that any of the runs in S return a solution to some problem instance (see Example 1), it is possible that $f(S) < 1$ for any finite schedule S. The best way of dealing with this is application-dependent. In the case of Example 1, we might stop computing G when $f(G_j) \geq 1 - \delta$ for some small $\delta > 0$.

The time required to compute G is also application-dependent. In the applications of interest to us, evaluating the $\arg\max$ in (2.4) will only require us to consider a finite number of actions (v, τ). In some cases, the evaluation of the $\arg\max$ in (2.4) can be sped up using application-specific data structures. In Chapter 3, we discuss the time required to compute G for various applications of interest.

As mentioned in §2.1.2, our analysis of the greedy approximation algorithm will only require that f is monotone and that f satisfies Condition 1. The following lemma shows that if f is a job, then f also satisfies these conditions.

Lemma 1. *If f satisfies* (2.1)*, then f satisfies Condition 1. That is, for any schedules $S_1, S \in \mathcal{S}$, we have*

$$\frac{f(S_1 \oplus S) - f(S_1)}{\ell\left(S\right)} \leq \max_{(v,\tau) \in \mathcal{V} \times \mathbb{R}_{>0}} \left\{ \frac{f(S_1 \oplus \langle (v, \tau) \rangle) - f(S_1)}{\tau} \right\} .$$

Proof. Let r denote the right hand side of the inequality. Let $S = \langle a_1, a_2, \ldots, a_L \rangle$, where $a_l = (v_l, \tau_l)$. Let

$$\Delta_l = f(S_1 \oplus \langle a_1, a_2, \ldots, a_l \rangle) - f(S_1 \oplus \langle a_1, a_2, \ldots, a_{l-1} \rangle) .$$

We have

$$\begin{aligned} f(S_1 \oplus S) &= f(S_1) + \sum_{l=1}^{L} \Delta_l && \text{(telescoping series)} \\ &\leq f(S_1) + \sum_{l=1}^{L} \left(f(S_1 \oplus \langle a_l \rangle) - f(S_1)\right) && \text{(submodularity)} \\ &\leq f(S_1) + \sum_{l=1}^{L} r \cdot \tau_l && \text{(definition of } r\text{)} \\ &= f(S_1) + r \cdot \ell\left(S\right) \ . \end{aligned}$$

Rearranging this inequality gives $\frac{f(S_1 \oplus S) - f(S_1)}{\ell(S)} \leq r$, as claimed. □

The key to the analysis of the greedy approximation algorithm is the following fact, which is the only property of G that we will use in our analysis.

Fact 1. *For any schedule S, any positive integer j, and any $t > 0$, we have*

$$f(S_{\langle t \rangle}) \leq f(G_j) + t \cdot s_j$$

where s_j is the j^{th} value of the maximum in (2.4)*.*

Fact 1 holds because $f(S_{\langle t \rangle}) \leq f(G_j \oplus S_{\langle t \rangle})$ by monotonicity, while $f(G_j \oplus S_{\langle t \rangle}) \leq f(G_j) + t \cdot s_j$ by Condition 1 and the definition of s_j.

Maximizing coverage

We first analyze the performance of the greedy algorithm on the BUDGETED MAXIMUM SUBMODULAR COVERAGE problem. The following theorem shows that, for certain values of T, the greedy schedule achieves the optimal approximation ratio of $1 - \frac{1}{e}$ for this problem. The proof of the theorem is similar to arguments in [56, 84].

Theorem 3. *Let L be a positive integer, and let $T = \sum_{j=1}^{L} \tau_j$, where $g_j = (v_j, \tau_j)$. Then $f\left(G_{\langle T \rangle}\right) > \left(1 - \frac{1}{e}\right) \max_{S \in \mathcal{S}} \left\{f\left(S_{\langle T \rangle}\right)\right\}$.*

Proof. Let $C^* = \max_{S \in \mathcal{S}} \left\{f\left(S_{\langle T \rangle}\right)\right\}$, and for any positive integer j, let $\Delta_j = C^* - f\left(G_j\right)$. By Fact 1, $C^* \leq f\left(G_j\right) + T s_j$. Thus

$$\Delta_j \leq T s_j = T \left(\frac{\Delta_j - \Delta_{j+1}}{\tau_j} \right) \ .$$

Rearranging this inequality gives $\Delta_{j+1} \leq \Delta_j \left(1 - \frac{\tau_j}{T}\right)$. Unrolling this inequality, we get

$$\Delta_{L+1} \leq \Delta_1 \left(\prod_{j=1}^{L} 1 - \frac{\tau_j}{T} \right) .$$

Subject to the constraint $\sum_{j=1}^{L} \tau_j = T$, the product series is maximized when $\tau_j = \frac{T}{L}$ for all j. Thus we have

$$C^* - f\left(G_{L+1}\right) = \Delta_{L+1} \leq \Delta_1 \left(1 - \frac{1}{L}\right)^L < \Delta_1 \frac{1}{e} \leq C^* \frac{1}{e} .$$

Thus $f\left(G_{L+1}\right) > (1 - \frac{1}{e})C^*$, as claimed. □

Theorem 3 shows that G gives a $1 - \frac{1}{e}$ approximation to the problem of maximizing coverage at time T, provided that T equals the sum of the durations of the actions in G_j for some positive integer j. Under the assumption that f is a job (as opposed to the weaker assumption that f satisfies Condition 1), the greedy algorithm can be combined with the partial enumeration approach of Khuller *et al.* [49] to achieve a $1 - \frac{1}{e}$ approximation ratio for any fixed T. The idea of this approach is to guess a sequence $Y = \langle a_1, a_2, a_3 \rangle$ of three actions, and then run the greedy algorithm on the job $f'(S) = f\left(Y \oplus S\right) - f\left(Y\right)$ with budget $T - T_0$, where T_0 is the total time consumed by the actions in Y. The arguments of [49, 84] show that, for some choice of Y, this yields a $\left(1 - \frac{1}{e}\right)$-approximation. In order for this approach to be feasible, actions must have discrete durations, so that the number of possible choices of Y is finite.

Minimizing cost

We next analyze the performance of the greedy algorithm on the MIN-SUM SUBMODULAR COVER problem. The following theorem uses the proof technique of [26] to show that the greedy schedule G has cost at most 4 times that of the optimal schedule, generalizing results of [26, 44, 64, 78, 79]. As already mentioned, achieving a better approximation ratio is NP-hard.

Theorem 4. $c\left(f, G\right) \leq 4 \int_{t=0}^{\infty} 1 - \max_{S \in \mathcal{S}} \left\{ f\left(S_{\langle t \rangle}\right) \right\} dt \leq 4 \min_{S \in \mathcal{S}} c\left(f, S\right)$.

Proof. Let $R_j = 1 - f\left(G_j\right)$; let $x_j = \frac{R_j}{2 s_j}$; let $y_j = \frac{R_j}{2}$; and let $h(x) = 1 - \max_S \left\{ f\left(S_{\langle x \rangle}\right) \right\}$. By Fact 1,

$$\max_S \left\{ f(S_{\langle x_j \rangle}) \right\} \leq f\left(G_j\right) + x_j s_j = f\left(G_j\right) + \frac{R_j}{2} .$$

Thus $h(x_j) \geq R_j - \frac{R_j}{2} = y_j$. The monotonicity of f implies that $h(x)$ is non-increasing and also that the sequence $\langle y_1, y_2, \ldots \rangle$ is non-increasing. As illustrated in Figure 2.1, these facts imply that $\int_{x=0}^{\infty} h(x)\, dx \geq \sum_{j \geq 1} x_j \left(y_j - y_{j+1}\right)$. Thus we have

$$\begin{aligned}
\int_{t=0}^{\infty} 1 - \max_{S \in \mathcal{S}} \left\{ f\left(S_{\langle t \rangle}\right) \right\} dt &= \int_{x=0}^{\infty} h(x)\, dx \\
&\geq \sum_{j \geq 1} x_j \left(y_j - y_{j+1}\right) && \text{(Figure 2.1)} \\
&= \frac{1}{4} \sum_{j \geq 1} R_j \frac{\left(R_j - R_{j+1}\right)}{s_j} \\
&= \frac{1}{4} \sum_{j \geq 1} R_j \tau_j \\
&\geq \frac{1}{4} c\left(f, G\right) && \text{(monotonicity of } f\text{)}
\end{aligned}$$

which proves the theorem. □

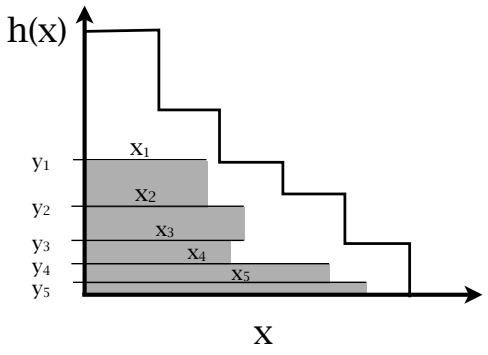

Figure 2.1: An illustration of the inequality $\int_{x=0}^{\infty} h(x)\, dx \geq \sum_{j \geq 1} x_j \left(y_j - y_{j+1}\right)$. The left hand side is the area under the curve, whereas the right hand side is the sum of the areas of the shaded rectangles.

A refined greedy approximation algorithm

A drawback of G is that it greedily chooses an action $g_j = (v, \tau)$ that maximizes the marginal increase in f divided by τ, whereas the contribution of (v, τ) to the cost of G is

not τ but rather

$$\int_{t=0}^{\tau} 1 - f\left(G_j \oplus \langle (v,t) \rangle\right) dt \ .$$

This can lead G to perform suboptimally even in seemingly easy cases. To see this, let $\mathcal{V} = \{v_1, v_2\}$, let $S_t^1 = \langle (v_1, t) \rangle$, and let $S_t^2 = \langle (v_2, t) \rangle$. Let f be a job defined by

$$f(S_t^1) = \begin{cases} 1 & \text{if } t \geq 1 \\ 0 & \text{otherwise} \end{cases}$$

whereas

$$f(S_t^2) = \min\{1, t\} \ .$$

For any schedule $S = \langle a_1, a_2, \ldots, a_L \rangle$ containing more than one action, let $f(S) = \max_{l=1}^{L} f(\langle a_l \rangle)$. It is straightforward to check that f satisfies the monotonicity and submodularity conditions required of a job.

Here the optimal schedule is $S^* = \langle (v_2, 1) \rangle$, with cost $c(f, S^*) = \int_{t=0}^{1} 1 - t \ dt = \frac{1}{2}$. However, if ties in the evaluation of the $\arg\max$ in (2.4) are broken appropriately, the greedy algorithm will choose the schedule $G = \langle (v_1, 1) \rangle$, with cost $c(f, G) = 1$.

To improve performance in cases such as this, it is natural to consider the schedule $G' = \langle g'_1, g'_2, \ldots \rangle$ defined inductively as follows: $G'_j = \left\{g'_1, g'_2, \ldots, g'_{j-1}\right\}$ and

$$g'_j = \underset{(v,\tau) \in \mathcal{V} \times \mathbb{R}_{>0}}{\arg\max} \left\{ \frac{f\left(G'_j \oplus \langle (v,\tau) \rangle\right) - f\left(G'_j\right)}{\int_{t=0}^{\tau} 1 - f\left(G'_j \oplus \langle (v,t) \rangle\right) dt} \right\} . \tag{2.5}$$

Theorem 5 shows that G' achieves the same approximation ratio as G. The proof is similar to the proof of Theorem 4, and is given in Appendix A.

Theorem 5. $c(f, G') \leq 4 \int_{t=0}^{\infty} 1 - \max_{S \in \mathcal{S}} \left\{ f\left(S_{\langle t \rangle}\right) \right\} dt \leq 4 \min_{S \in \mathcal{S}} \{c(f, S)\}$.

Furthermore, we prove in Chapter 3 (see Theorem 18) that, in contrast to G, G' is *optimal* in the important special case when $\mathcal{V} = \{v\}$, action (v, τ) represents running a Las Vegas algorithm v (with a fresh random seed) for time τ, and $f(S)$ equals the probability that at least one of the runs in S returns a solution to some particular problem instance (as described in Example 1).

Handling non-uniform additive error

We now consider the case in which the j^{th} decision made by the greedy algorithm is performed with some additive error ϵ_j. This case is of interest for two reasons. First, in

some cases it may not be practical to evaluate the $\arg\max$ in (2.4) exactly. Second, and more importantly, we will end up viewing our online algorithm as a version of the offline greedy algorithm in which each decision is made with some additive error. In this section we analyze the original greedy schedule G as opposed to the refined schedule G' described in the previous section, because it is the original schedule G that will form the basis of our online algorithm (as we discuss further in §2.5, devising an online algorithm based on G' is an interesting open problem).

We denote by $\bar{G} = \langle \bar{g}_1, \bar{g}_2, \ldots \rangle$ a variant of the schedule G in which the j^{th} $\arg\max$ in (2.4) is evaluated with additive error ϵ_j. More formally, $\bar{G}$ is a schedule that, for any $j \geq 1$, satisfies

$$\frac{f\left(\bar{G}_j \oplus \bar{g}_j\right) - f\left(\bar{G}_j\right)}{\bar{\tau}_j} \geq \max_{(v,\tau)\in\mathcal{V}\times\mathbb{R}_{>0}} \left\{ \frac{f\left(\bar{G}_j \oplus \langle (v,\tau) \rangle\right) - f\left(\bar{G}_j\right)}{\tau} \right\} - \epsilon_j \tag{2.6}$$

where $\bar{G}_0 = \langle\rangle$, $\bar{G}_j = \langle \bar{g}_1, \bar{g}_2, \ldots, \bar{g}_{j-1} \rangle$ for $j > 1$, and $\bar{g}_j = (\bar{v}_j, \bar{\tau}_j)$.

The following two theorems summarize the performance of $\bar{G}$. The proofs are given in Appendix A, and are along the same lines as that those of theorems 3 and 4.

Theorem 6. *Let L be a positive integer, and let $T = \sum_{j=1}^{L} \bar{\tau}_j$, where $\bar{g}_j = (\bar{v}_j, \bar{\tau}_j)$. Then*

$$f\left(\bar{G}_{\langle T \rangle}\right) > \left(1 - \frac{1}{e}\right) \max_{S\in\mathcal{S}} \left\{ f\left(S_{\langle T \rangle}\right) \right\} - \sum_{j=1}^{L} \epsilon_j \bar{\tau}_j \,.$$

Theorem 7. *Let L be a positive integer, and let $T = \sum_{j=1}^{L} \bar{\tau}_j$, where $\bar{g}_j = (\bar{v}_j, \bar{\tau}_j)$. For any schedule S, define $c^T(f, S) \equiv \int_{t=0}^{T} 1 - f\left(S_{\langle t \rangle}\right) dt$. Then*

$$c^T\left(f, \bar{G}\right) \leq 4 \int_{t=0}^{\infty} 1 - \max_{S\in\mathcal{S}} \left\{ f\left(S_{\langle t \rangle}\right) \right\} dt + \sum_{j=1}^{L} E_j \bar{\tau}_j \,.$$

where $E_j = \sum_{l<j} \epsilon_l \bar{\tau}_l$.

2.4 Online Algorithms

In this section we consider the online versions of BUDGETED MAXIMUM SUBMODULAR COVERAGE and MIN-SUM SUBMODULAR COVER. In the online setting we are fed, one at a time, a sequence $\langle f_1, f_2, \ldots, f_n \rangle$ of jobs. Prior to receiving job f_i, we must

specify a schedule S_i. We then receive complete access to the function f_i. We measure the performance of our online algorithm using two different notions of regret. For the cost objective, our goal is to minimize the 4-regret

$$R_{cost} \equiv \sum_{i=1}^{n} c^T(S_i, f_i) - 4 \cdot \min_{S \in \mathcal{S}} \left\{ \sum_{i=1}^{n} c(S, f_i) \right\}$$

for some fixed $T > 0$. Here, for any schedule S and job f, we define $c^T(S, f) = \int_{t=0}^{T} 1 - f\left(S_{\langle t \rangle}\right) dt$ to be the value of $c(S, f)$ when the integral is truncated at time T. Some form of truncation is necessary because $c(S_i, f_i)$ could be infinite, and without bounding it we could not prove any finite bound on regret (our regret bounds will be stated as a function of T).

For the objective of maximizing the coverage at time T, our goal is to minimize the $(1 - \frac{1}{e})$-regret

$$R_{coverage} \equiv \left(1 - \frac{1}{e}\right) \max_{S \in \mathcal{S}} \left\{ \sum_{i=1}^{n} f_i\left(S_{\langle T \rangle}\right) \right\} - \sum_{i=1}^{n} f_i(S_i)$$

where we require that $\mathbb{E}[\ell(S_i)] = T$, in expectation over the online algorithm's random bits. In other words, we allow the online algorithm to treat T as a budget in expectation, rather than a hard budget.

Our goal is to bound the expected values of R_{cost} (resp. $R_{coverage}$) on the worst-case sequence of n jobs. We consider the so-called *oblivious adversary model*, in which the sequence of jobs is fixed in advance and does not change in response to the decisions made by our online algorithm, although we believe our results can be readily extended to the case of adaptive adversaries. Note that the constant of 4 in the definition of R_{cost} and the constant of $1 - \frac{1}{e}$ in the definition of $R_{coverage}$ stem from the NP-hardness of the corresponding offline problems, as discussed in §2.3.1.

For the purposes of the results in this section, we confine our attention to schedules that consist of actions that come from some finite set $\mathcal{A}$, and we assume that the actions in $\mathcal{A}$ have integer durations (i.e. $\mathcal{A} \subseteq \mathcal{V} \times \mathbb{Z}_{>0}$). Note that this is not a serious limitation, because real-valued action durations can always be discretized at whatever level of granularity is desired.

As mentioned in §2.1.2, our results in the online setting will hold for any sequence $\langle f_1, f_2, \ldots, f_n \rangle$ of functions that satisfies Condition 2. The following lemma shows that any sequence of jobs satisfies this condition. The proof follows along the same lines as the proof of Lemma 1, and is given in Appendix A.

Lemma 2. *Any sequence $\langle f_1, f_2, \ldots, f_n \rangle$ of jobs satisfies Condition 2. That is, for any sequence $S_1, S_2, \ldots, S_n$ of schedules and any schedule S,*

$$\frac{\sum_{i=1}^{n} f_i(S_i \oplus S) - f_i(S_i)}{\ell(S)} \leq \max_{(v,\tau) \in \mathcal{V} \times \mathbb{R}_{>0}} \left\{ \frac{\sum_{i=1}^{n} f_i(S_i \oplus \langle (v,\tau) \rangle) - f_i(S_i)}{\tau} \right\}.$$

2.4.1 Background: the experts problem

In the experts problem, one has access to a set of k experts, each of whom gives out a piece of advice every day. On each day i, one must select an expert e_i whose advice to follow. Following the advice of expert j on day i yields a reward x_j^i. At the end of day i, the value of the reward x_j^i for each expert j is made public, and can be used as the basis for making choices on subsequent days. One's *regret* at the end of n days is equal to

$$\max_{1 \leq j \leq k} \left\{ \sum_{i=1}^{n} x_j^i \right\} - \sum_{i=1}^{n} x_{e_i}^i .$$

Note that the historical performance of an expert does not imply any guarantees about its future performance. Remarkably, randomized decision-making algorithms nevertheless exist whose regret grows sub-linearly in the number of days. By picking experts using such an algorithm, one can guarantee to obtain (asymptotically as $n \to \infty$) an average reward that is as large as the maximum reward that could have been obtained by following the advice of any fixed expert for all n days.

In particular, for any fixed value of G_{max}, where $G_{max} = \max_{1 \leq j \leq k} \left\{ \sum_{i=1}^{n} x_j^i \right\}$, the randomized weighted majority algorithm (WMR) [60] can be used to achieve worst-case regret $O\left(\sqrt{G_{max} \ln k}\right)$. If G_{max} is not known in advance, a putative value can be guessed and doubled to achieve the same guarantee up to a constant factor.

2.4.2 Unit-cost actions

In the special case in which each action takes unit time (i.e., $\mathcal{A} \subseteq \mathcal{V} \times \{1\}$), our online algorithm $\mathbf{OG_{unit}}$ is very simple. $\mathbf{OG_{unit}}$ runs T experts algorithms:[2] $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_T$, where T is the number of time steps for which our schedule is defined. The set of experts is $\mathcal{A}$. Just before job f_i arrives, each experts algorithm $\mathcal{E}_t$ selects an action a_t^i. The schedule used by $\mathbf{OG_{unit}}$ on job f_i is $S_i = \langle a_1^i, a_2^i, \ldots, a_T^i \rangle$. The payoff that $\mathcal{E}_t$ associates with action a is $f_i\left(S_{i\langle t-1 \rangle} \oplus a\right) - f_i\left(S_{i\langle t-1 \rangle}\right)$.

[2] In general, $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_T$ will be T distinct copies of a single experts algorithm, such as randomized weighted majority.

Algorithm $\mathbf{OG_{unit}}$
Input: integer T, experts algorithms $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_T$.

For i from 1 to n:

1. For each t, $1 \le t \le T$, use $\mathcal{E}_t$ to select an action a_t^i.
2. Select the schedule $S_i = \langle a_1^i, a_2^i, \ldots, a_T^i \rangle$.
3. Receive the job f_i.
4. For each t, $1 \le t \le T$, and each action $a \in \mathcal{A}$, feed back $f_i\left(S_{i\langle t-1\rangle} \oplus a\right) - f_i\left(S_{i\langle t-1\rangle}\right)$ as the payoff $\mathcal{E}_t$ would have received by choosing action a.

Let r_t be the regret experienced by experts algorithm $\mathcal{E}_t$ when running $\mathbf{OG_{unit}}$, and let $R = \sum_{t=1}^{T} r_t$. The key to the analysis of $\mathbf{OG_{unit}}$ is the following lemma, which relates the regret experienced by the experts algorithms to the regret on the original online problem.

Lemma 3. *$R_{coverage} \le R$ and $R_{cost} \le TR$.*

Proof. We will view $\mathbf{OG_{unit}}$ as producing an approximate version of the offline greedy schedule for the function $f = \frac{1}{n}\sum_{i=1}^{n} f_i$. First, view the sequence of actions selected by $\mathcal{E}_t$ as a single "meta-action" $\tilde{a}_t$, and extend the domain of each f_i to include the meta-actions by defining $f_i(S \oplus \tilde{a}_t) = f_i(S \oplus a_t^i)$ for all $S \in \mathcal{S}$. Thus, the online algorithm produces a single schedule $S_i = \tilde{S} = \langle \tilde{a}_1, \tilde{a}_2, \ldots, \tilde{a}_T \rangle$ for all i. By construction,

$$\frac{r_t}{n} = \max_{a \in \mathcal{A}} \left\{ f\left(\tilde{S}_{\langle t-1\rangle} \oplus a\right) - f\left(\tilde{S}_{\langle t-1\rangle}\right) \right\} - \left(f\left(\tilde{S}_{\langle t-1\rangle} \oplus \tilde{a}_t\right) - f\left(\tilde{S}_{\langle t-1\rangle}\right) \right) .$$

Thus $\mathbf{OG_{unit}}$ behaves exactly like the greedy schedule $\bar{G}$ for the function f, where the t^{th} decision is made with additive error $\frac{r_t}{n}$.

Furthermore, the fact that the sequence $\langle f_1, f_2, \ldots, f_n \rangle$ satisfies Condition 2 implies that for any integer t ($1 \le t \le T$) and any schedule S, we have

$$\frac{f(\tilde{S}_{\langle t-1\rangle} \oplus S) - f(\tilde{S}_{\langle t-1\rangle})}{\ell(S)} \le \max_{(v,\tau) \in \mathcal{V} \times \mathbb{R}_{>0}} \left\{ \frac{f(\tilde{S}_{\langle t-1\rangle} \oplus \langle (v,\tau) \rangle) - f(\tilde{S}_{\langle t-1\rangle})}{\tau} \right\} .$$

Thus the function f satisfies Condition 1, so the analysis of the greedy approximation algorithm in §2.3.2 applies to the schedule $\tilde{S}$. In particular, Theorem 6 implies that $R_{coverage} \le \sum_{t=1}^{T} r_t = R$. Similarly, Theorem 7 implies that $R_{cost} \le TR$. □

To complete the analysis, it remains to bound $\mathbb{E}[R]$. First, note that the payoffs to each experts algorithm $\mathcal{E}_t$ depend on the choices made by experts algorithms $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_{t-1}$, but not on the choices made by $\mathcal{E}_t$ itself. Thus, from the point of view of $\mathcal{E}_t$, the payoffs are generated by a non-adaptive adversary. Suppose that randomized weighted majority (WMR) is used as the subroutine experts algorithm. Because each payoff is at most 1 and there are n rounds, $\mathbb{E}[r_t] = O\left(\sqrt{G_{max} \ln |\mathcal{A}|}\right) = O\left(\sqrt{n \ln |\mathcal{A}|}\right)$, so a trivial bound is $\mathbb{E}[R] = O\left(T\sqrt{n \ln |\mathcal{A}|}\right)$. In fact, we can show that the worst case is when $G_{max} = \Theta\left(\frac{n}{T}\right)$ for all T experts algorithms, leading to the following improved bound. The proof is given in Appendix A.

Lemma 4. *Algorithm* $\mathbf{OG_{unit}}$*, run with WMR as the subroutine experts algorithm, has* $\mathbb{E}[R] = O\left(\sqrt{Tn \ln |\mathcal{A}|}\right)$ *in the worst case.*

Combining Lemmas 3 and 4 yields the following theorem.

Theorem 8. *Algorithm* $\mathbf{OG_{unit}}$*, run with WMR as the subroutine experts algorithm, has* $\mathbb{E}[R_{coverage}] = O\left(\sqrt{Tn \ln |\mathcal{A}|}\right)$ *and* $\mathbb{E}[R_{cost}] = O\left(T\sqrt{Tn \ln |\mathcal{A}|}\right)$ *in the worst case.*

2.4.3 From unit-cost actions to arbitrary actions

In this section we generalize the online greedy algorithm presented in the previous section to accommodate actions with arbitrary durations. Like $\mathbf{OG_{unit}}$, our generalized algorithm $\mathbf{OG}$ makes use of a series of experts algorithms $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_L$ (for L to be determined). On each round i, $\mathbf{OG}$ constructs a schedule S_i as follows: for $t = 1, 2, \ldots, L$, it uses $\mathcal{E}_t$ to choose an action $a_t^i = (v, \tau) \in \mathcal{A}$, and appends this action to S_i with probability $\frac{1}{\tau}$. The payoff that $\mathcal{E}_t$ associates with action a equals $\frac{1}{\tau}$ times the increase in f that would have resulted from appending a to the schedule-under-construction.

Algorithm OG
Input: integer L, experts algorithms $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_L$.

For i from 1 to n:

1. Let $S_{i,0} = \langle\rangle$ be the empty schedule.
2. For each t, $1 \leq t \leq L$,
   (a) Use $\mathcal{E}_t$ to choose an action $a_t^i = (v, \tau) \in \mathcal{A}$.
   (b) With probability $\frac{1}{\tau}$, set $S_{i,t} = S_{i,t-1} \oplus \langle a \rangle$; else set $S_{i,t} = S_{i,t-1}$.
3. Select the schedule $S_i = S_{i,L}$.
4. Receive the job f_i.
5. For each t, $1 \leq t \leq L$, and each action $a \in \mathcal{A}$, feed back
$$x_{t,a}^i = \frac{1}{\tau}\left(f_i\left(S_{i,t-1} \oplus a\right) - f_i\left(S_{i,t-1}\right)\right)$$
as the payoff $\mathcal{E}_t$ would have received by choosing action a.

Our analysis of $\mathbf{OG}$ follows along the same lines as the analysis of $\mathbf{OG_{unit}}$ in the previous section. As in the previous section, we will view each experts algorithm $\mathcal{E}_t$ as selecting a single "meta-action" $\tilde{a}_t$. We extend the domain of each f_i to include the meta-actions by defining

$$f_i(S \oplus \tilde{a}_t) = \begin{cases} f_i(S \oplus a_t^i) & \text{if } a_t^i \text{ was appended to } S_i \\ f_i(S) & \text{otherwise.} \end{cases}$$

Thus, the online algorithm produces a single schedule $S_i = \tilde{S} = \langle \tilde{a}_1, \tilde{a}_2, \ldots, \tilde{a}_L \rangle$ for all i.

For the purposes of analysis, we will imagine that each meta-action $\tilde{a}_t$ *always* takes unit time (whereas in fact, $\tilde{a}_t$ takes unit time per job in expectation). We show later that this assumption does not invalidate any of our arguments.

Let $f = \frac{1}{n}\sum_{i=1}^n f_i$, and let $\tilde{S}_t = \langle \tilde{a}_1, \tilde{a}_2, \ldots, \tilde{a}_t \rangle$. As in the previous section, the fact that the sequence $\langle f_1, f_2, \ldots, f_n \rangle$ satisfies Condition 2 implies that f satisfies Condition 1 (even if the schedule S_1 in the statement of Condition 1 contains meta-actions). Thus $\tilde{S}$ can be viewed as a version of the greedy schedule in which the t^{th} decision is made with additive error (by definition) equal to

$$\epsilon_t = \max_{(v,\tau)\in\mathcal{A}} \left\{ \frac{1}{\tau}\left(f(\tilde{S}_{t-1} \oplus a) - f(\tilde{S}_{t-1})\right)\right\} - \left(f(\tilde{S}_{t-1} \oplus \tilde{a}_t) - f(\tilde{S}_{t-1})\right)$$

(where we have used the assumption that $\tilde{a}_t$ takes unit time).

As in the previous section, let r_t be the regret experienced by $\mathcal{E}_t$. In general, $\frac{r_t}{n} \neq \epsilon_t$. However, we claim that $\mathbb{E}[\epsilon_t] = \mathbb{E}\left[\frac{r_t}{n}\right]$. To see this, fix some integer t ($1 \leq t \leq L$), let $A_t = \langle a_t^1, a_t^2, \ldots, a_t^n \rangle$ be the sequence of actions selected by $\mathcal{E}_t$, and let y_t^i be the payoff received by $\mathcal{E}_t$ on round i (i.e., $y_t^i = x_{t,a_t^i}^i$). By construction,

$$y_t^i = \mathbb{E}\left[f_i(\tilde{S}_{t-1} \oplus \tilde{a}_t) - f_i(\tilde{S}_{t-1}) | A_t, \tilde{S}_{t-1}\right] .$$

Thus,

$$\frac{r_t}{n} = \max_{(v,\tau)\in\mathcal{A}} \left\{ \frac{1}{\tau}\left(f(\tilde{S}_{t-1} \oplus a) - f(\tilde{S}_{t-1})\right)\right\} - \mathbb{E}\left[f(\tilde{S}_{t-1} \oplus \tilde{a}_t) - f(\tilde{S}_{t-1}) | A_t, \tilde{S}_{t-1}\right] .$$

Taking the expectation of both sides of the equations for ϵ_t and r_t then shows that $\mathbb{E}[\epsilon_t] = \mathbb{E}\left[\frac{r_t}{n}\right]$, as claimed.

We now prove a bound on $\mathbb{E}[R_{coverage}]$. As already mentioned, f satisfies Condition 1, so the greedy schedule's approximation guarantees apply to f. In particular, by Theorem 6, we have $R_{coverage} \leq \sum_{t=1}^{T} r_t$. Thus $\mathbb{E}[R_{coverage}] \leq \mathbb{E}[R]$, where $R = \sum_{t=1}^{T} r_t$.

To bound $\mathbb{E}[R_{coverage}]$, it remains to justify the assumption that each meta-action $\tilde{a}_t$ always takes unit time. Regardless of what actions are chosen by each experts algorithm, the schedule is defined for L time steps in expectation. Thus if we set $L = T$, the schedules S_i returned by **OG** satisfy the budget in expectation, as required in the definition of $R_{coverage}$. Thus, as far as $R_{coverage}$ is concerned, the meta-actions may as well take unit time (in which case $\ell(S_i) = T$ with probability 1). Combining the bound on $\mathbb{E}[R]$ stated in Lemma 4 with the fact that $\mathbb{E}[R_{coverage}] \leq \mathbb{E}[R]$ yields the following theorem.

Theorem 9. *Algorithm* **OG***, run with input $L = T$, has $\mathbb{E}[R_{coverage}] \leq \mathbb{E}[R]$. If WMR is used as the subroutine experts algorithm, then $\mathbb{E}[R] = O\left(\sqrt{Tn \ln |\mathcal{A}|}\right)$.*

The argument bounding $\mathbb{E}[R_{cost}]$ is similar, although somewhat more involved, and is given in Appendix A. Relative to the case of unit-cost actions addressed in the previous section, the additional complication here is that $\ell(S_i)$ is now a random variable, whereas in the definition of R_{cost} the cost of a schedule is always calculated up to time T. This complication can be overcome by making the probability that $\ell(S_i) < T$ sufficiently small, which can be accomplished by setting $L \gg T$ and applying concentration inequalities. However, $\mathbb{E}[R]$ grows as a function of L, so we do not want to make L too large. It turns out that the (approximately) best bound is obtained by setting $L = T \ln n$.

Theorem 10. *Algorithm* **OG***, run with input $L = T \ln n$, has $\mathbb{E}[R_{cost}] = O(T \ln n \cdot \mathbb{E}[R] + T\sqrt{n})$. In particular, $\mathbb{E}[R_{cost}] = O\left((\ln n)^{\frac{3}{2}} T\sqrt{Tn \ln |\mathcal{A}|}\right)$ if WMR is used as the subroutine experts algorithm.*

2.4.4 Dealing with limited feedback

Thus far we have assumed that, after specifying a schedule S_i, the online algorithm receives complete access to the job f_i. We now consider three more limited feedback settings that may arise in practice:

1. In the *priced feedback model*, to receive access to f_i we must pay a price C. Each time we do so, C is added to the regret $R_{coverage}$, and TC is added to the regret R_{cost}.
2. In the *partially transparent feedback model*, we only observe $f_i\left(S_{i\langle t\rangle}\right)$ for each $t > 0$.
3. In the *opaque feedback model*, we only observe $f_i(S_i)$.

The priced and partially transparent feedback models arise naturally in the case where action (v, τ) represents running a deterministic algorithm v for τ (additional) time units in order to solve some decision problem. Assuming we halt once some v returns an answer, we obtain exactly the information that is revealed in the partially transparent model. Alternatively, running each v until it terminates would completely reveal the function f_i, but incurs a computational cost.

Algorithm **OG** can be adapted to work in each of these three feedback settings. In all cases, the high-level idea is to replace the unknown quantities used by **OG** with (unbiased) estimates of those quantities. This technique has been used in a number of online algorithms (e.g., see [5, 8, 17]).

Specifically, for each day i and expert j, let $\hat{x}_j^i \in [0, 1]$ be an estimate of x_j^i, such that

$$\mathbb{E}\left[\hat{x}_j^i\right] = \gamma x_j^i + \delta^i$$

for some constant δ^i (which is independent of j). In order words, we require that $\frac{1}{\gamma}\left(\hat{x}_j^i - \delta^i\right)$ is an unbiased estimate of x_j^i. Furthermore, let $\hat{x}_j^i$ be independent of the choices made by the experts algorithm.

Let $\mathcal{E}$ be an experts algorithm, and let $\mathcal{E}'$ be the experts algorithm that results from feeding back $\hat{x}_j^i$ to $\mathcal{E}$ (in place of x_j^i) as the payoff $\mathcal{E}$ would have received by selecting expert j on day i. The following lemma relates the performance of $\mathcal{E}'$ to that of $\mathcal{E}$.

Lemma 5. *The worst-case expected regret that $\mathcal{E}'$ can incur over a sequence of n days is at most $\frac{R}{\gamma}$, where R is the worst-case expected regret that $\mathcal{E}$ can incur over a sequence of n days.*

Proof. Let $\hat{x} = \langle \hat{x}^1, \hat{x}^2, \ldots, \hat{x}^n \rangle$ be the sequence of estimated payoffs. Because the estimates $\hat{x}^i_j$ are independent of the choices made by $\mathcal{E}'$, we may imagine for the purposes of analysis that $\hat{x}$ is fixed in advance. Fix some expert j. By definition of R,

$$\mathbb{E}\left[\sum_{i=1}^{n} \hat{x}^i_{e_i} | \hat{x}\right] \geq \left(\sum_{i=1}^{n} \hat{x}^i_j\right) - R\,.$$

Taking the expectation of both sides with respect to the choice of $\hat{x}$ then yields

$$\mathbb{E}\left[\sum_{i=1}^{n} \left(\gamma x^i_{e_i} + \delta^i\right)\right] \geq \sum_{i=1}^{n} \left(\gamma x^i_j + \delta^i\right) - R$$

or rearranging,

$$\mathbb{E}\left[\sum_{i=1}^{n} x^i_{e_i}\right] \geq \left(\sum_{i=1}^{n} x^i_j\right) - \frac{R}{\gamma}\,.$$

Because j was arbitrary, it follows that $\mathcal{E}'$ has worst-case expected regret $\frac{R}{\gamma}$. □

The priced feedback model

In the priced feedback model, we use a technique similar to that of [17]. With probabiltiy γ, we will pay cost C in order to reveal f_i, and then feed the usual payoffs back to each experts algorithm $\mathcal{E}_t$. Otherwise, with probability $1-\gamma$, we feed back zero payoffs to each $\mathcal{E}_t$ (note that without paying cost C, we receive no information whatsoever about f_i, and thus we have no basis for assigning different payoffs to different actions). We refer to this algorithm as $\mathbf{OG^P}$. By Lemma 5, $\mathbb{E}[r_t]$ is bounded by $\frac{1}{\gamma}$ times the worst-case regret of $\mathcal{E}_t$. By bounding $\mathbb{E}[R_{coverage}]$ and $\mathbb{E}[R_{cost}]$ as a function of γ and then optimizing γ to minimize the bounds, we obtain the following theorem, a complete proof of which is given in Appendix A.

Theorem 11. *Algorithm $\mathbf{OG^P}$, run with WMR as the subroutine experts algorithm, has $\mathbb{E}[R_{coverage}] = O\left((C \ln |\mathcal{A}|)^{\frac{1}{3}}(Tn)^{\frac{2}{3}}\right)$ (when run with input $L = T$) and has $\mathbb{E}[R_{cost}] = O\left((T \ln n)^{\frac{5}{3}}(C \ln |\mathcal{A}|)^{\frac{1}{3}}(n)^{\frac{2}{3}}\right)$ (when run with input $L = T \ln n$) in the priced feedback model.*

The partially transparent feedback model

In the partially transparent feedback model, each $\mathcal{E}_t$ will run a copy of the **Exp3** algorithm [5], which is a randomized experts algorithm that only requires as feedback the payoff of the expert it actually selects. In the partially transparent feedback model, if $\mathcal{E}_t$ selects action $a_t^i = (v, \tau)$ on round i, it will receive feedback $f_i\left(S_{i,t-1} \oplus a_t^i\right) - f_i\left(S_{i,t-1}\right)$ if a_t^i is appended to the schedule (with probability $\frac{1}{\tau}$), and will receive zero payoff otherwise. Observe that the information necessary to compute these payoffs is revealed in the partially transparent feedback model. Furthermore, the expected payoff that $\mathcal{E}_t$ receives if it selects action a is $x_{t,a}^i$, and the payoff that $\mathcal{E}_t$ receives from choosing action a on round i is independent from the choices made by $\mathcal{E}_t$ on previous rounds. Thus, by Lemma 5, the worst-case expected regret bounds of **Exp3** can be applied to the true payoffs $x_{t,a}^i$. The worst-case expected regret of **Exp3** is $O\left(\sqrt{n\left|\mathcal{A}\right|\ln\left|\mathcal{A}\right|}\right)$, so $\mathbb{E}\left[R\right] = O\left(L\sqrt{n\left|\mathcal{A}\right|\ln\left|\mathcal{A}\right|}\right)$. This bound, combined with Theorems 9 and 10, establishes the following theorem.

Theorem 12. *Algorithm* $\mathbf{OG}$*, run with* **Exp3** *as the subroutine experts algorithm, has* $\mathbb{E}\left[R_{coverage}\right] = O\left(T\sqrt{n\left|\mathcal{A}\right|\ln\left|\mathcal{A}\right|}\right)$ *(when run with input* $L = T$*) and has* $\mathbb{E}\left[R_{cost}\right] = O\left((T\ln n)^2\sqrt{n\left|\mathcal{A}\right|\ln\left|\mathcal{A}\right|}\right)$ *(when run with input* $L = T\ln n$*) in the partially transparent feedback model.*

The opaque feedback model

In the opaque feedback model, our algorithm and its analysis are similar to those of $\mathbf{OG^P}$. With probability $1-\gamma$, we feed back zero payoffs to each $\mathcal{E}_t$. Otherwise, with probability γ, we *explore* as follows. Pick t uniformly at random from $\{1, 2, \ldots, L\}$, and pick an action $a = (v, \tau)$ uniformly at random from $\mathcal{A}$. Select the schedule $S_i = S_{i,t-1} \oplus a$. Observe $f_i(S_i)$, and feed $\frac{1}{\tau}$ times this value back to $\mathcal{E}_t$ as the payoff associated with action a. Finally, feed back zero for all other payoffs.

We refer to this algorithm as $\mathbf{OG^o}$. The key to its analysis is the following observation. Letting $\hat{x}_{t,a}^i$ denote the payoff to experts algorithm $\mathcal{E}_t$ for choosing action $a = (v, \tau)$ on round i, we have

$$\mathbb{E}\left[\hat{x}_{t,a}^i\right] = \gamma \cdot \frac{1}{L} \cdot \frac{1}{\left|\mathcal{A}\right|} \cdot \frac{1}{\tau} \cdot f(S_{i,t-1} \oplus a) = \frac{\gamma}{L\left|\mathcal{A}\right|} x_{t,a}^i + \delta^i$$

where $x_{t,a}^i = \frac{1}{\tau}\left(f(S_{i,t-1} \oplus a) - f(S_{i,t-1})\right)$ and $\delta^i = \frac{\gamma}{L|\mathcal{A}|\tau} f(S_{i,t-1})$. Thus, $\hat{x}_{t,a}^i$ is a biased estimate of the correct payoff, and Lemma 5 implies that $\mathbb{E}\left[r_t\right]$ is at most $\frac{L|\mathcal{A}|}{\gamma}$ times the worst-case expected regret of $\mathcal{E}_t$.

The performance of $\mathbf{OG^o}$ is summarized in the following theorem, which we prove in Appendix A.

Theorem 13. *Algorithm* $\mathbf{OG^o}$*, run with WMR as the subroutine experts algorithm, has* $\mathbb{E}[R_{coverage}] = O\left(T(|\mathcal{A}|\ln|\mathcal{A}|)^{\frac{1}{3}}n^{\frac{2}{3}}\right)$ *(when run with input* $L = T$*) and has* $\mathbb{E}[R_{cost}] = O\left((T\ln n)^2(|\mathcal{A}|\ln|\mathcal{A}|)^{\frac{1}{3}}n^{\frac{2}{3}}\right)$ *(when run with input* $L = T\ln n$*) in the opaque feedback model.*

2.4.5 Lower bounds on regret

In Appendix A we prove the following lower bounds on regret. The lower bounds apply to the online versions of two set-covering problems: MAX k-COVERAGE and MIN-SUM SET COVER. The offline versions of these two problems were defined in §2.1.4. The online versions are special cases of the online versions of BUDGETED MAXIMUM SUBMODULAR COVERAGE and MIN-SUM SUBMODULAR COVER, respectively. For a formal description of the online set covering problems, see the text leading up to the proofs of Theorems 14 and 15 in Appendix A.

It is worth pointing out that the lower bounds hold even in a *distributional* online setting in which the jobs $f_1, f_2, \ldots, f_n$ are drawn independently at random from a fixed distribution.

Theorem 14. *Any algorithm for online* MAX k-COVERAGE *has worst-case expected 1-regret* $\Omega\left(\sqrt{Tn\ln\frac{|\mathcal{V}|}{T}}\right)$*, where* $\mathcal{V}$ *is the collection of sets and* $T = k$ *is the number of sets selected by the online algorithm on each round.*

Theorem 15. *Any algorithm for online* MIN-SUM SET COVER *has worst-case expected 1-regret* $\Omega\left(T\sqrt{Tn\ln\frac{|\mathcal{V}|}{T}}\right)$*, where* $\mathcal{V}$ *is a collection of sets and* T *is the number of sets selected by the online algorithm on each round.*

In Appendix A we show that there exist exponential-time online algorithms for these online set covering problems whose regret matches the lower bounds in Theorem 14 (resp. Theorem 15) up to constant (resp. logarithmic) factors.

Note that the upper bounds in Theorem 8 match the lower bounds in Theorems 14 and 15 up to logarithmic factors, although the former apply to $(1-\frac{1}{e})$-regret and 4-regret rather than 1-regret.

2.4.6 Refining the online greedy algorithm

We now discuss two simple modifications to **OG** that do not improve its worst-case guarantees, but that often improve its performance in practice (we make use of both of these modifications in our experiments in Chapter 3).

Avoiding duplicate actions

In many practical applications, it is never worthwhile to perform the same action twice. As an example, suppose that an action $a = (v, \tau)$ represents performing a run of length τ of a deterministic algorithm v (and then removing the run from memory), and $f(S) = 1$ if performing the actions in S yields a solution to a problem instance, and $f(S) = 0$ otherwise. Clearly, performing a twice can never increase the value of f. In cases such as this, the online algorithm **OG** as currently defined may never "figure out" that it should avoid performing the same action twice, as the following example illustrates.

Example 2. Let $\mathcal{A} = \{a_1, a_2, \ldots, a_T\}$ be a set of T actions that each take unit time, and for all i, let $f_i(S)$ equal $\frac{1}{T}$ times the number of distinct actions that appear in S. Thus, the schedule $S^* = \langle a_1, a_2, \ldots, a_T \rangle$ has $f_i(S^*) = 1$ for all i, and is optimal in terms of coverage. Suppose we run **OG** on the sequence of jobs $\langle f_1, f_2, \ldots, f_n \rangle$. All actions yield equal payoff to $\mathcal{E}_1$. If $\mathcal{E}_1$ is a standard experts algorithm such as randomized weighted majority, it will choose actions uniformly at random. Given that $\mathcal{E}_1$ chooses actions uniformly at random, $\mathcal{E}_2$ will (asymptotically) choose actions uniformly at random as well. Inductively, all actions will be chosen at random. If so, the probability that any particular experts algorithm selects a unique action is $1 - (1 - \frac{1}{T})^T$ (which approaches $1 - \frac{1}{e}$ as $T \to \infty$). By linearity of expectation, the expected fraction of actions that are unique is exactly this quantity.

To improve performance on examples such as this one, we may force the online algorithm to return a schedule with no duplicate actions as follows. Just before job f_i arrives, obtain from each experts algorithm $\mathcal{E}_t$ a distribution over $\mathcal{A}$ (for experts algorithms such as randomized weighted majority, it is straightforward to obtain this distribution explicitly). We then sample from these distributions as follows. We first sample from $\mathcal{E}_1$ to obtain an action a_1^i. To obtain action a_t^i for $t > 1$, we repeatedly sample from the distribution returned by $\mathcal{E}_t$ until we obtain an action not in the set $\left\{a_1^i, a_2^i, \ldots, a_{t-1}^i\right\}$ (given the distribution, we can simulate this step without actually performing repeated sampling).

With this modification, **OG** always achieves coverage 1 for the job f described in example 2. Furthermore, this modification preserves the worst-case guarantees of the original version of **OG** (under the assumption performing the same action twice never

increases the value of any function f_i). Informally, this follows from the fact that the expected payoff received by sampling from the modified distribution can *never* be smaller than the expected payoff received by sampling from the original distribution (because the payoffs associated with the experts corresponding to actions already in the schedule are all zero). For this reason, this modification never increases the worst-case regret of the experts algorithms, and our previous analysis carries through unchanged.

Independent versus dependent probabilities

Recall that in the case of arbitrary-cost actions, when an experts algorithm selects an action (v, τ) we add this action to the schedule independently with probability $\frac{1}{\tau}$. The fact that this addition is performed *independently* of the actions that are already in the schedule can lead to undesirable behavior, as the following example illustrates.

Example 3. Let $\mathcal{V} = \{v\}$ consist of a single activity, let $f(S) = 1$ if S contains the action (v, T), and let $f(S) = 0$ otherwise. Thus, the schedule $S^* = \langle (v, T) \rangle$ maximizes $f(S_{\langle T \rangle})$. However, $\mathbb{E}[f(S)] \leq 1 - (1 - \frac{1}{T})^T$ if S is a schedule returned by **OG**. This is true because at most T experts algorithms can select the action (v, T), but in each case the action is only added to the schedule with probability $\frac{1}{T}$, so the probability that (v, T) is added to the schedule is at most $1 - (1 - \frac{1}{T})^T$, which approaches $1 - \frac{1}{e}$ as $T \to \infty$.

We can fix this problem as follows. When experts algorithm $\mathcal{E}_t$ selects an action $a_t = (v, \tau)$, we *increase* the probability that the action is in the schedule by $\frac{1}{\tau}$. In other words, if a_t has been picked by k experts algorithms so far but has still not been added to the schedule, then we add it to the schedule with probability $\frac{1}{\tau - k}$. Thus, if τ consecutive experts algorithms select the same action (v, τ), it will always be added to the schedule exactly once.

The schedules produced by this modified online algorithm still consume T time steps in expectation, and our previous analysis carries through to give same regret bounds on $R_{coverage}$ that were stated in Theorem 9. Unfortunately, the analysis for the bounds on R_{cost} stated in Theorem 10 depends critically on the use of independent probabilities, and does not carry through after having made this modification. Nevertheless, in our experiments in Chapter 3 we found that this modification was helpful in practice.

2.5 Open Problems

The results presented in this chapter suggest several open problems:

1. *Avoiding discretization.* As currently defined, our online algorithm can only handle finite set of actions $\mathcal{A}$. Thus, to apply this online algorithm to a problem in which the actions have real-valued durations between 0 and 1, one might discretize the durations to be in the set $\left\{\frac{1}{T}, \frac{2}{T}, \ldots, 1\right\}$. To achieve the best performance, one would like to set T as large as possible, but the time and space required by the online algorithm grow linearly with T. It would be desirable to avoid discretization altogether, perhaps after making additional smoothness assumptions about the jobs f_i. A possible approach would be to consider the limiting behavior of our algorithm as $T \to \infty$, for some particular choice of subroutine experts algorithm.

2. *Lower bounds on 4-regret and $1 - \frac{1}{e}$ regret.* The lower bounds proved in §2.4.5 apply only to 1-regret, whereas our online algorithms optimize either 4 regret (in the case of R_{cost}) or $1 - \frac{1}{e}$ regret (in the case of $R_{coverage}$). It would be interesting to prove lower bounds on R_{cost} and $R_{coverage}$. Such lower bounds would hold for online algorithms that make decisions in polynomial time, under the assumption that P $\neq$ NP.

3. *An online version of the refined greedy approximation algorithm G'.* Recall that in §2.3.2 we showed that the offline greedy approximation algorithm is sub-optimal for a simple job involving two activities, and then considered an alternative greedy approximation algorithm that produces an optimal schedule for this job. The online algorithm presented in §2.4 is based on the original greedy approximation algorithm, and thus it also performs sub-optimally on this simple example. Although it appears non-trivial to do so, it would be interesting to develop an online version of the alternative greedy approximation algorithm that performed optimally on such examples.

2.6 Conclusions

This chapter considered an online resource allocation problem that generalizes several previously-studied online problems, and that has applications to algorithm portfolio design and the optimization of query processing in databases. The main contribution of this chapter was an online version of a greedy approximation algorithm whose worst-case performance guarantees in the offline setting are the best possible assuming P $\neq$ NP. In the next chapter we evaluate the online greedy algorithm experimentally by using it to combine multiple problem-solving heuristics in an online setting.

Chapter 3

Combining Multiple Heuristics Online

3.1 Introduction

In this chapter we present black-box techniques that can be used to combine multiple problem-solving heuristics into a new heuristic with (potentially) improved average-case running time. In our model, a user is given a set of heuristics whose only observable behavior is their running time. Each heuristic can compute a solution to any problem instance, but its running time varies across instances. The user solves each instance by interleaving runs of the heuristics according to some schedule. If the heuristics are randomized, the user may also periodically restart them with a fresh random seed.

Building on the results of chapter 2, we present

1. exact and approximation algorithms for computing an optimal schedule offline,

2. sample complexity bounds for learning a schedule from training data, and

3. no-regret algorithms for learning a schedule on-the-fly while solving a sequence of problems.

In our experimental evaluation, we use data from recent solver competitions to show that these algorithms can be used to create improved versions of state-of-the-art solvers in a number of problem domains.

3.1.1 Motivations

Many important computational problems seem unlikely to admit algorithms with provably good worst-case performance, yet must be solved as a matter of practical necessity. Examples of such problems include Boolean satisfiability, A.I. planning, integer programming, and numerous scheduling and resource allocation problems. In each of these problem domains, heuristics[1] have been developed that perform much better in practice than a worst-case analysis would guarantee, and there is an active research community working to develop improved heuristics. Indeed, entire conferences are devoted to the study of particular problem domains (e.g., Boolean satisfiability, A.I. planning), and annual competitions are held in order to assess the state of the art and to promote the development of better heuristics.

A major drawback of using heuristics is that the behavior of a heuristic on a previously unseen problem instance can be difficult to predict in advance. The running time of a heuristic may vary by orders of magnitude across seemingly similar problem instances or, if the heuristic is randomized, across multiple runs on a single instance that use different random seeds [33, 38]. For this reason, after running a heuristic unsuccessfully for some time one might decide to suspend the execution of that heuristic and start running a different heuristic (or the same heuristic with a different random seed).

Previous work has shown that combining multiple heuristics into a *portfolio* can dramatically improve average-case running time [33, 38]. Table 3.1 illustrates a situation in which this is the case. The table shows the behavior of the top two solvers from the industrial track of the 2007 SAT competition on three of the competition benchmarks. On these three instances, simply running the two solvers in parallel (e.g., at equal strength on a single processor) would reduce the average-case running time by orders of magnitude.

Table 3.1: Behavior of two solvers on instances from the 2007 SAT competition.

Instance	`Rsat` **CPU (s)**	`picosat` **CPU (s)**
industrial/anbulagan/medium-sat/dated-10-13-s.cnf	45	28
industrial/babic/dspam/dspam_dump_vc1081.cnf	3	$\geq$ 10000
industrial/grieu/vmpc_31.cnf	$\geq$ 10000	238

[1] A heuristic is simply an algorithm. We use the term "heuristic" only to suggest worst-case exponential running time.

If the heuristics are randomized, additional performance improvements may be achieved by periodically restarting them with a fresh random seed. In particular, solvers based on chronological backtracking often exhibit heavy-tailed run length distributions, and restarts can yield order-of-magnitude improvements in performance [34, 35].

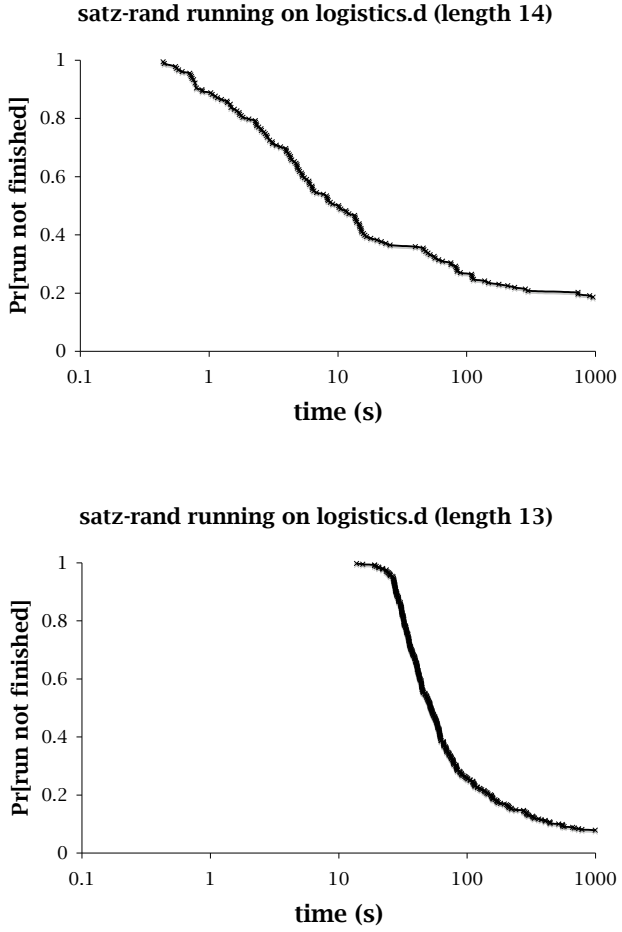

Figure 3.1: Run length distribution of `satz-rand` on two formulae created by `SATPLAN` in solving the logistics planning instance `logistics.d`. Each curve was estimated using 150 independent runs, and run lengths were capped at 1000 seconds.

Figure 1 shows the run length distribution of the SAT solver `satz-rand` on two Boolean formulae created by running a state-of-the-art planning algorithm, `SATPLAN`, on a logistics planning benchmark. To find a provably minimum-length plan, `SATPLAN` creates a sequence of Boolean formulae $\langle \sigma_1, \sigma_2, \ldots \rangle$, where σ_i is satisfiable if and only

if there exists a feasible plan of length $\leq i$. In this case the minimum plan length is 14. When run on the (satisfiable) formula σ_{14}, `satz-rand` exhibits a heavy-tailed run length distribution. There is about a 20% chance of solving the problem after running for 2 seconds, but also a 20% chance that a run will not terminate after having run for 1000 seconds. By restarting the solver every 2 seconds until it yields a solution, one can reduce the expected time required to find a solution by more than an order of magnitude. In contrast, when `satz-rand` is run on the (unsatisfiable) instance σ_{13}, over 99% of the runs take at least 19 seconds, so the same restart policy would be ineffective. Restarts are still beneficial on this instance, however; restarting every 45 seconds reduces the mean run length by at least a factor of 1.5. Of course, when using a randomized SAT solver to solve a given formula for the first time one does not know its run length distribution on that formula, and thus one must select a restart schedule based on experience with previously-solved formulae.

3.1.2 Formal setup

In our model, we are given a set $\mathcal{H}$ of heuristics, with $|\mathcal{H}| = k$, and a set $\mathcal{X}$ of instances of some decision problem. Heuristic h, when run on instance x, runs for $T(h, x)$ time units before returning a (provably correct) "yes" or "no" answer. In general, the heuristic h will be randomized, and $T(h, x)$ will be a random variable whose outcome depends on the sequence of random bits supplied as input to h.

As in Chapter 2, we consider schedules that are sequences of actions of the form $(h, \tau) \in \mathcal{H} \times \mathbb{R}_{>0}$. We use $\mathcal{S}$ to denote the set of schedules. In our setting, an action (h, τ) represents running heuristic h for (additional) time τ. We require that each heuristic $h \in \mathcal{H}$ be executed in one of two models: the *suspend-and-resume model* or the *restart model* (the choice of model need not be the same for all $h \in \mathcal{H}$).

- If h is executed in the suspend-and-resume model, then an action (h, τ) represents continuing a run of heuristic h for an additional τ time units. When the action is completed, the run of h is temporarily suspended and kept resident in memory, to be potentially resumed by a later action.

- If h is executed in the restart model, then an action (h, τ) represents running h from scratch for time τ, and then deleting the run from memory. If h is randomized, the run is performed with a fresh random seed.

As an example, suppose that h_1 is executed in the suspend-and-resume model and h_2

is executed in the restart model. Then the schedule

$$S = \langle (h_1, 5), (h_2, 5), (h_1, 10), (h_2, 10) \rangle$$

is interpreted as follows: "run h_1 for 5 time units; then suspend the run of h_1 and run h_2 for 5 time units; then discard the run of h_2 and continue the run of h_1 for an additional 10 time units; then suspend the run of h_1 and run h_2 from scratch (with a fresh random seed) for 10 time units."

This class of schedules includes *restart-schedules* [61] and *task-switching schedules* [73] as special cases.

- A *restart schedule* is a schedule for a set $\mathcal{H}$ that contains a single heuristic, executed in the restart model. A restart schedule can be written more concisely as a sequence $S = \langle \tau_1, \tau_2, \ldots \rangle$ of positive integers, whose meaning is "run h for τ_1 time units; if this does not yield a solution then restart h with a fresh sequence of random bits and run it for τ_2 time units, ...". When executing a restart schedule, only a single run needs to be kept in memory.

- A *task-switching schedule* is a schedule that runs all heuristics in the suspend-and-resume model. If all heuristics in $\mathcal{H}$ are deterministic, then the optimal schedule must be a task-switching schedule (assuming there is no overhead associated with keeping multiple runs in memory). When executing a task-switching schedule, up to k runs need to be kept in memory simultaneously.

We use $\mathcal{S}_{rs}$ and $\mathcal{S}_{ts}$ to denote the set of restart schedules and the set of task-switching schedules, respectively.

We measure the performance of a schedule S on a problem instance x in terms of the expected time required to solve x using S (where the expectation is over the random bits used in the runs that S performs). For any schedule S, let $p_x(S)$ denote the probability that performing the sequence of actions in S yields a solution to x. For example, if $S = \langle (h_1, \tau_1), (h_2, \tau_2), \ldots, (h_L, \tau_L) \rangle$, and all heuristics are executed in the restart model, then

$$p_x(S) = 1 - \prod_{l=1}^{L} \mathbb{P}\left[T\left(h_l, x\right) > \tau_l\right] \ .$$

In §3.3.1 we formally define $p_x(S)$ in the general case, when one or more heuristics may be executed in the suspend-and-resume model.

Overloading notation, let $T(S, x)$ denote the time required to solve instance x using schedule S. For any non-negative random variable X, we have $\mathbb{E}[X] = \int_{t=0}^{\infty} \mathbb{P}[X > t]\, dt$. Thus the expected time required to solve x using S can be written as

$$\mathbb{E}[T(S, x)] = \int_{t=0}^{\infty} 1 - p_x(S_{\langle t \rangle})\, dt \equiv c(p_x, S) \tag{3.1}$$

where, as in chapter 2, $S_{\langle t \rangle}$ is the schedule that results from truncating schedule S at time t. For example if $S = \langle (h_1, 3), (h_2, 3) \rangle$ then $S_{\langle 5 \rangle} = \langle (h_1, 3), (h_2, 2) \rangle$ (for a formal definition of $S_{\langle t \rangle}$, see §2.1.1).

In the special case when all heuristics are executed in the restart model, it can be shown that $p_x(S)$ satisfies the conditions required of a *job*, as defined in Chapter 2. However, when one or more heuristics are executed in the suspend-and-resume model, p_x is no longer submodular. Nevertheless, we show in §3.1.4 that the sufficient conditions described in Chapter 2 are satisfied even when some heuristics are executed in the suspend-and-resume model, so the results from Chapter 2 still apply.

By definition, the right hand side of (3.1) equals the cost $c(p_x, S)$ of the schedule S for the job p_x. This implies that the offline and online greedy approximation algorithms presented in Chapter 2 can be used to select schedules of the form considered in this chapter, so as to minimize $\mathbb{E}[T(S, x)]$.

As in Chapter 2, we use $\ell(S)$ to denote the sum of the durations of the actions in S. For example if $S = \langle (v_1, 3), (v_2, 3) \rangle$, then $\ell(S) = 6$.

3.1.3 Summary of results

We first consider the schedule selection problem in an offline setting. In the offline setting we are given a set of instances $\mathcal{X}$, and are given as input the distribution of $T(h, x)$ for all $h \in \mathcal{H}$ and $x \in \mathcal{X}$. Our goal is to compute the schedule with minimum total expected running time over the instances in $\mathcal{X}$, namely $S^* = \arg\min_{S \in \mathcal{S}} \sum_{x \in \mathcal{X}} \mathbb{E}[T(S, x)]$. In this setting, the greedy algorithm for MIN-SUM SUBMODULAR COVER from Chapter 2 gives a 4-approximation to the optimal schedule. We also show that, even in the special case where all heuristics are deterministic, this offline problem generalizes MIN-SUM SET COVER [26], implying that for any $\epsilon > 0$, computing an $4 - \epsilon$ approximation is NP-hard. We also give exact and approximation algorithms based on shortest path computations, whose running time is exponential as a function of k (where $k = |\mathcal{H}|$) but is polynomial for any fixed k.

We next consider a learning-theoretic setting in which we draw training instances independently at random from a distribution, compute an optimal schedule for the training

instances, and then use that schedule to solve additional test instances drawn from the same distribution. In this setting, we give bounds on the number of instances required to learn a schedule that is *probably approximately correct* [86].

We then consider an online setting in which we are fed a sequence $\mathcal{X} = \langle x_1, x_2, \ldots, x_n \rangle$ of problem instances one at a time and must obtain a solution to each instance (via some schedule) before moving on to the next instance. When selecting a schedule S_i to use to solve instance x_i, we have knowledge of the previous instances $x_1, x_2, \ldots, x_{i-1}$ but we have no knowledge of x_i itself or of any subsequent instances. In this setting, the online greedy algorithm for MIN-SUM SUBMODULAR COVER (from Chapter 2) converges to a 4-approximation to the best schedule, and requires decision-making time polynomial in k. We also present online shortest paths algorithms that converge to an α-approximation to the best schedule (for some desired $\alpha > 1$), but which requires decision-making time exponential in k.

We then discuss how our results in these three settings can be extended in two ways. First, we show that our algorithms can be applied in an interesting way to heuristics for optimization rather than decision problems. Second, we discuss how quickly-computable features of problem instances can be used to improve the schedule selection process.

Experimentally, we use data from recent solver competitions to show that task-switching schedules computed by our greedy approximation algorithm can be used to improve the performance of state-of-the-art solvers in several problem domains. Our experimental evaluation considers both optimization and decision problems, and makes use of instance features to improve the schedule selection process. We also show that that the greedy approximation algorithm can be used to construct a restart schedule that improves the performance of a randomized SAT solver on a set of logistics planning benchmarks.

The results in this chapter are based in part on two conference papers [78, 79].

3.1.4 Relationship to the framework of Chapter 2

Before moving on, we show how the problem considered in this chapter can be put into the framework of Chapter 2. As already mentioned, this fact allows us to apply the offline greedy approximation algorithm from Chapter 2, as well as its online counterpart, to the problem considered in this chapter.

As mentioned in §3.1.2, given an instance x, we will be interested in selecting a schedule S so as to minimize the expected running time $c(p_x, S) = \int_{t=0}^{\infty} 1 - p_x(S)\ dt = \mathbb{E}[T(S, x)]$. In the online setting, a sequence $\langle x_1, x_2, \ldots, x_n \rangle$ of instances arrive online. The following lemma shows that the sequence $\langle p_{x_1}, p_{x_2}, \ldots, p_{x_n} \rangle$ satisfies Condition 2

(defined in §2.1.2). As we discuss further in §3.6.3, this implies that the online greedy algorithm from Chapter 2 can be used to select schedules in such a way that the average running time of the schedules selected by the online algorithm is (asymptotically) at most 4 times that of the optimal fixed schedule for the given sequence of instances.

Lemma 6. *Let $\langle x_1, x_2, \ldots, x_n \rangle$ be a sequence of n problem instances, and let $p_i = p_{x_i}$. Then the sequence $\langle p_1, p_2, \ldots, p_n \rangle$ satisfies Condition 2 from §2.1.2. That is, for any sequence $S_1, S_2, \ldots, S_n$ of schedules and any schedule S,*

$$\frac{\sum_{i=1}^{n} p_i(S_i \oplus S) - p_i(S_i)}{\ell(S)} \leq \max_{(v,\tau) \in \mathcal{V} \times \mathbb{R}_{>0}} \left\{ \frac{\sum_{i=1}^{n} p_i(S_i \oplus \langle (v,\tau) \rangle) - p_i(S_i)}{\tau} \right\}. \quad (3.2)$$

Proof. Any schedule S can be rewritten as $S = \langle a_1, a_2, \ldots, a_L \rangle$, where each action a_l runs a *different* heuristic, without changing the value of $p(S)$ (i.e., multiple runs of a heuristic executed in the suspend-and-resume model can be consolidated into a single run). Fix some instance x_i, and let $p_i'(S) = p_i(S_i \oplus S) - p_i(S_i)$. Thus, $p_i'(S)$ is the probability that at least one action in S solves x_i after all actions in S_i have failed to solve x_i. Using the union bound, we have

$$p_i'(S) \leq \sum_{l=1}^{L} p_i'(\langle a_l \rangle).$$

Let $a_l = (h_l, \tau_l)$, and let r be the maximum in the right hand side of (3.2). Summing the inequality over all $x \in \mathcal{X}$ yields

$$\sum_{i=1}^{n} p_i(S_i \oplus S) \leq \sum_{l=1}^{L} \sum_{i=1}^{n} p(S_i \oplus \langle a_l \rangle) \leq \sum_{l=1}^{L} r \cdot \tau_l = r \cdot \ell(S)$$

which proves the lemma. □

In the offline setting, we are given a set of instances $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$, and wish to select a schedule S that minimizes $\frac{1}{n} \sum_{i=1}^{n} \mathbb{E}[T(S, x_i)]$. Using equation (3.1), this is the same as minimizing $\frac{1}{n} \sum_{i=1}^{n} c(p_{x_i}, S) = c(p, S)$, where $p(S) = \frac{1}{n} \sum_{i=1}^{n} p_{x_i}(S)$. The following corollary of Lemma 6 shows that the function p satisfies Condition 1 (defined in §2.1.2). As we discuss further in §3.4.2, this implies that the greedy approximation algorithm from Chapter 2 can be used to obtain a 4-approximation to the optimal schedule.

Corollary 1. *Let $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ be a set of problem instances, and let $p(S) = \frac{1}{n} \sum_{i=1}^{n} p_{x_i}(S)$. Then p satisfies Condition 1 from §2.1.2. That is, for any $S_1, S \in \mathcal{S}$, we have*

$$\frac{p(S_1 \oplus S) - p(S_1)}{\ell(S)} \leq \max_{(v,\tau) \in \mathcal{V} \times \mathbb{R}_{>0}} \left\{ \frac{p(S_1 \oplus \langle (v,\tau) \rangle) - p(S_1)}{\tau} \right\}.$$

3.2 Related Work

In this section we discuss two areas of related work: *algorithm portfolios* and *restart schedules*.

3.2.1 Algorithm portfolios

The work presented in this chapter is closely related to, and shares the same goals as, previous work on *algorithm portfolios* [33, 38]. An algorithm portfolio is a schedule for combining runs of various heuristics. The schedules considered in the original papers on algorithm portfolios simply run each heuristic in parallel at equal strength and assign each heuristic a fixed restart threshold. Gomes *et al.* [33] addressed the problem of constructing an optimal algorithm portfolio offline given knowledge of the run length distribution of each algorithm, under the assumption that each algorithm has the same run length distribution on all problem instances. Earlier work [43] considered the problem of devising a schedule for combining multiple heuristics that achieves an optimal competitive ratio on a single problem instance.

A recent paper by Sayag *et al.* [73] considered the problems of selecting task-switching schedules and resource-sharing schedules for multiple heuristics, both in the offline and learning-theoretic settings. A *resource-sharing schedule* $S : \mathcal{H} \rightarrow [0, 1]$ specifies that all heuristics in $\mathcal{H}$ are to be run in parallel, with each $h \in \mathcal{H}$ receiving a proportion $S(h)$ of the CPU time. The primary contribution of their paper was an offline algorithm that computes an optimal resource-sharing schedule in $O(n^{k-1})$ time. They also discuss an $O(n^{k+1})$ algorithm for computing optimal task-switching schedules offline. As proved by Sayag *et al.* (Lemma 1 of [73]), an optimal task-switching schedule always performs as well or better than an optimal resource-sharing schedule.

Independently, Petrik [67] and Petrik and Zilberstein [68] gave exact and approximation algorithms for computing optimal task-switching schedules and optimal resource-sharing schedules. Their algorithms are based on dynamic programming, and the running time is exponential in k.

The term algorithm portfolio has also been used to describe approaches that use features of instances to attempt to predict which algorithm will run the fastest on a given instance, and then simply run that algorithm exclusively [59, 90]. Note that in this approach there is no notion of a schedule *per se*. As already mentioned, we show how instance-specific features can be incorporated into our framework later in this chapter.

The works just described consider the problem of learning an algorithm portfolio from

training data. Recently, Gagliolo and Schmidhuber [30] presented an algorithm that, like our online algorithms, can be used to select algorithm portfolios on-the-fly while solving a sequence of problem instances. Their algorithm produces resource-sharing schedules, and uses one of a number of rules to select resource-sharing schedules based on statistical models of the behavior of the heuristics.

3.2.2 Restart schedules

There has been a considerable amount of work on devising restart schedules for Las Vegas algorithms. Letting $\mathcal{A}$ denote an arbitrary Las Vegas algorithm, such a schedule can be represented as a sequence $\langle t_1, t_2, \ldots \rangle$ of positive real numbers, whose meaning is "run $\mathcal{A}$ for t_1 time units; if this does not yield a solution then restart and run for t_2 time units, ...".

In the early 1990s, at least two papers studied the problem of selecting a restart schedule to use in solving a *single* problem instance, given no prior knowledge of the algorithm's run length distribution on that instance. The main results are summarized in the following theorem proved by Luby *et al.* [61].

Theorem 16 (Luby *et al.* , 1993)**.**

1. *For any instance x, the schedule S that minimizes $\mathbb{E}[T(S,x)]$ is a uniform restart schedule of the form $\langle t^*, t^*, \ldots, t^* \rangle$.*
2. *Let $\ell = \min_{S \in \mathcal{S}_{rs}} \mathbb{E}[T(S,x)]$. The universal restart schedule*[2]
$$S_{univ} = \langle 1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, \ldots \rangle$$
has $\mathbb{E}[T(S_{univ}, x)] = O(\ell \log \ell)$.
3. *For any schedule S, it is possible to define a distribution of $T(\mathcal{A}, x)$ such that $\mathbb{E}[T(S,x)] \geq \frac{1}{8}\ell \log_2 \ell$ (i.e., the worst-case performance of S_{univ} is optimal to within constant factors).*

Luby [61] also showed that other classes of restart schedules, for example suspend-and-resume and probabilistic schedules, are no more powerful than ordinary restart schedules (assuming the restart schedule's performance is measured on a single problem instance). Alt *et al.* [3] gave related results, with a focus on minimizing tail probabilities rather than expected running time.

[2]The universal schedule can be described as follows. All run lengths are powers of two, and as soon as two runs of the same length have been completed, a run of twice that length is immediately performed.

In the late 1990s, there was a renewed interest in restart schedules due to a paper by Gomes *et al.* [35], which demonstrated that (then) state-of-the-art solvers for Boolean satisfiability and constraint satisfaction could be dramatically improved by randomizing the solver's decision-making heuristics and running the randomized solver with an appropriate restart schedule. In one experiment, their paper took a deterministic SAT solver called `satz` and created a version called `satz-rand` with a randomized heuristic for selecting which variable to branch on at each node in the search tree. They found that, on certain problem instances, `satz-rand` exhibited a heavy-tailed run length distribution. By periodically restarting it they obtained order-of-magnitude improvements in running time over both `satz-rand` (without restarts) and `satz`, the original deterministic solver. Their paper also demonstrated the benefit of randomization and restart for a then state-of-the-art constraint solver.

One limitation of Theorem 16 is that it is "all or nothing": it either assumes complete knowledge of the run length distribution (in which case a uniform restart schedule is optimal) or no knowledge at all (in which case the universal schedule is optimal to within constant factors). Several papers have considered the case in which partial but not complete knowledge of the run length distribution is available. Ruan *et al.* [71] consider the case in which each run length distribution is one of m known distributions, and give a dynamic programming algorithm for computing an optimal restart schedule. The running time of their algorithm is exponential in m, and thus it is practical only when m is small (in the paper the algorithm is described for $m = 2$). Kautz *et al.* [45] considered the case in which, after running for some fixed amount of time, one observes a feature that gives the distribution of that run's length.

A paper by Gagliolo & Schmidhuber [31] considered the problem of selecting restart schedules online in order to solve a sequence of problem instances as quickly as possible. Their paper treats the schedule selection problem as a 2-armed bandit problem, where one of the arms runs Luby's universal schedule and the other arm runs a schedule designed to exploit the empirical run length distribution of the instances encountered so far. Their strategy was designed to work well in the case where each instance has a similar run length distribution.

3.2.3 Contributions of this chapter

The results in this chapter advance the state of the art in algorithm portfolio design in four important ways:

1. We consider a powerful class of schedules that *generalizes* both task-switching

schedules and restart schedules.

2. We provide *polynomial-time* approximation algorithms for computing schedules of this form, and state hardness-of-approximation results showing that no polynomial-time algorithm can provide better worst-guarantees assuming P $\neq$ NP. Note that no polynomial-time approximation algorithms were previously known for computing task-switching schedules or for computing restart schedules (in the general case of multiple problem instances with heterogeneous run length distributions).

3. We provide *online* algorithms with strong theoretical guarantees, which can be used to learn an appropriate schedule on-the-fly while solving a sequence of problems. The worst-case performance of our online algorithms is the same as that of the corresponding offline approximation algorithm, asymptotically as the number of problem instances goes to infinity.

4. We show how features of problem instances can be exploited by our online algorithms in a natural way, thus *unifying* the benefits of two previous approaches [38, 59] to algorithm portfolio design.

3.3 State Space Representation of Schedules

In this section we introduce a state-space representation of schedules that will be used extensively in later sections. We first discuss *profiles* and *states*, which provide a canonical way of representing the work done by a particular schedule at a particular time. We then discuss how the problem of computing an optimal schedule can be solved as a shortest path problem in a graph whose vertices are states. We use this shortest path formulation in §3.4 to obtain offline algorithms, then use it again in §3.5 to obtain sample complexity bounds for learning a schedule from training data. Lastly, we discuss α-regularity, which provides a way to drastically reduce the size of the state space at the cost of a constant factor performance degradation, leading in §3.4 to offline approximation algorithms and in §3.5 to additional sample complexity bounds.

3.3.1 Profiles and states

A *profile* $P = \langle \tau_1, \tau_2, \ldots, \tau_L \rangle$ is a non-increasing sequence of positive real numbers. For any heuristic h, we use $\mathcal{P}(S, h)$ to denote a profile that lists, in non-decreasing order, the

total lengths each run of h performed by S. If h is being executed in the suspend-and-resume model, then $\mathcal{P}(S,h)$ always contains a single number. If h is being executed in the restart model, then the number of values in $\mathcal{P}(S,h)$ equals the number of actions in S that refer to h. The *size* of a profile $P = \langle \tau_1, \tau_2, \ldots, \tau_L \rangle$ is defined to be $\sum_{l=1}^{L} \tau_l$.

As an example, let

$$S = \langle (h_1, 1), (h_2, 2), (h_1, 3), (h_2, 4) \rangle \ .$$

If h_1 is being executed in the suspend-and-resume model, then $\mathcal{P}(S,h_1) = \langle 4 \rangle$; otherwise $\mathcal{P}(S,h_1) = \langle 3, 1 \rangle$. Similarly, if h_2 is being executed in the suspend-and-resume model, then $\mathcal{P}(S,h_2) = \langle 6 \rangle$; otherwise $\mathcal{P}(S,h_2) = \langle 4, 2 \rangle$.

A *state* $Y = \langle P_1, P_2, \ldots, P_k \rangle$ is a k-tuple of profiles, where $k = |\mathcal{H}|$. We define the size of a state to be the sum of the sizes of the k profiles it contains. For any schedule S, we define a corresponding state

$$\mathcal{Y}(S) = \langle \mathcal{P}(S,h_1), \mathcal{P}(S,h_2), \ldots, \mathcal{P}(S,h_k) \rangle$$

where $\mathcal{H} = \{h_1, h_2, \ldots, h_k\}$. We use $Y^{\emptyset} = \langle \langle\rangle, \langle\rangle, \ldots, \langle\rangle \rangle$ to denote the empty state.

For any instance x, the value of $p_x(S)$ (the probability that performing the runs in S yields a solution to x) can be written as follows. Let $\mathcal{Y}(S) = \langle P_1, P_2, \ldots, P_k \rangle$, where $P_j = \langle \tau_1^j, \tau_2^j, \ldots, \tau_{L_j}^j \rangle$. Then

$$p_x(S) = 1 - \prod_{j=1}^{k} \prod_{l=1}^{L_j} \mathbb{P}\left[T(h_j, x) > \tau_l^j\right] \ . \tag{3.3}$$

The key property of states is given by the following lemma, which follows immediately from the definitions.

Lemma 7. *Let S_1 and S_2 be schedules such that $\mathcal{Y}(S_1) = \mathcal{Y}(S_2)$. Then*

1. *for any instance x, $p_x(S_1) = p_x(S_2)$, and*
2. *for any schedule S, $\mathcal{Y}(S_1 \oplus S) = \mathcal{Y}(S_2 \oplus S)$.*

Proof. The first statement is immediate from (3.3). To prove the second statement, assume for contradication that $\mathcal{P}(S_1 \oplus S, h) \neq \mathcal{P}(S_2 \oplus S, h)$, for some heuristic h. First, suppose h is executed in the suspend-and-resume model. Then $\mathcal{P}(S_1,h) = \mathcal{P}(S_2,h) = \langle \tau \rangle$, and $\mathcal{P}(S,h) = \langle \tau' \rangle$. It follows that $\mathcal{P}(S_1 \oplus S, h) = \mathcal{P}(S_2 \oplus S, h) = \langle \tau + \tau' \rangle$.

Now suppose h is executed in the restart model. Let $\mathcal{P}(S_1, h) = \mathcal{P}(S_2, h) = \langle \tau_1, \tau_2, \ldots, \tau_L \rangle$, and let $\mathcal{P}(S, h) = \langle \tau'_1, \tau'_2, \ldots, \tau'_M \rangle$. Then $\mathcal{P}(S_1 \oplus S, h)$ simply contains the $L + M$ values in $\mathcal{P}(S_1, h)$ and $\mathcal{P}(S, h)$, listed in non-decreasing order, and the same is true of $\mathcal{P}(S_2 \oplus S, h)$. □

3.3.2 State space graphs

A *state space graph* is a triple $G = \langle V, E, S_e \rangle$, where $\langle V, E \rangle$ is a directed acyclic graph whose vertices are states, and $S_e : E \to \mathcal{S}$ is a function that labels each edge $e \in E$ with a schedule $S_e(e)$. We require that a state-space graph satisfy the following conditions.

1. V contains the empty state $Y^\emptyset$, as well as a distinguished vertex v^* (v^* is not a state),
2. for every state $Y \in V$, there is a path from Y to v^*, and
3. if the edges $e_1, e_2, \ldots, e_L$ form a path from $Y^\emptyset$ to some state Y, then
$$\mathcal{Y}(S_e(e_1) \oplus S_e(e_2) \oplus \ldots \oplus S_e(e_L)) = Y \ .$$

We use $\mathcal{S}_G$ to denote the set of schedules that correspond to paths from $Y^\emptyset$ to v^* in G. In other words, $S \in \mathcal{S}_G$ if any only if there exists a path $e_1, e_2, \ldots, e_L$ from $Y^\emptyset$ to v^* such that $S = S_e(e_1) \oplus S_e(e_2) \oplus \ldots \oplus S_e(e_L)$.

We now describe how to assign weights to the edges of a state-space graph in such a way that an optimal schedule within the set $\mathcal{S}_G$ can be found by computing a shortest path from $Y^\emptyset$ to v^*. Let $e = \langle Y_1, Y_2 \rangle$ be a directed edge in the state space graph. Let S_1 be any schedule such that $\mathcal{Y}(S_1) = Y_1$, and let $S = S_e(e)$. For any instance x, define the weight $w(e, x)$ as
$$w(e, x) = \int_{t=\ell(S_1)}^{\ell(S_1 \oplus S)} 1 - p_x\left((S_1 \oplus S)_{\langle t \rangle}\right) dt \ . \tag{3.4}$$
Note that by Lemma 7, the value of the right hand side does not depend on the choice of S_1.

Let the edges $e_1, e_2, \ldots, e_L$ form a path from $Y^\emptyset$ to v^* in the state space graph. Let $S = S_e(e_1) \oplus S_e(e_2) \oplus \ldots \oplus S_e(e_L)$. By construction, the weight of the path is exactly
$$\sum_{i=1}^{L} w(e_i, x) = \int_{t=0}^{\ell(S)} 1 - p_x(S_{\langle t \rangle})\, dt = \mathbb{E}\left[\min\left\{\ell(S), T(S, x)\right\}\right] \ .$$

Thus we have the following lemma.

Lemma 8. *Given a state space graph G and a set of instances $\mathcal{X}$, the schedule*

$$S^* = \arg\min_{S \in \mathcal{S}_G} \sum_{x \in \mathcal{X}} \mathbb{E}\left[\min\left\{\ell\left(S\right), T\left(S, x\right)\right\}\right]$$

may be found by computing a shortest path from the empty state $Y^{\emptyset}$ to v^* in G, where each edge e is assigned weight $\sum_{x \in \mathcal{X}} w(e, x)$, where $w(e, x)$ is defined in equation (3.4)*.*

Note that this lemma provides a way to optimize $\mathbb{E}\left[\min\left\{\ell\left(S\right), T\left(S, x\right)\right\}\right]$, whereas in general we will be interested in optimizing $\mathbb{E}\left[T\left(S, x\right)\right]$. When making use of this lemma, we will set up our state space graph G so that $T\left(S, x\right) \leq \ell\left(S\right)$ for all $x \in \mathcal{X}$ and for all $S \in \mathcal{S}_G$, either with certainty or with high probability.

3.3.3 α-Regularity

In this section we discuss α-regularity, which provides a way to reduce the number of states that must be considered when using the shortest path formulation described in the previous section, at the cost of a constant factor performance degradation.

Special case: suspend-and-resume only

In the special case in which all heuristics are executed in the suspend-and-resume model, the definition of an α-regular schedule is simple: a schedule S is α-regular if, whenever S stops running a heuristic h and starts running a different heuristic instead, the total time invested so far in h is a power of α (i.e., it equals α^i for some integer i). For example, the schedule

$$S = \langle (h_1, 2), (h_2, 1), (h_1, 14), (h_2, 3) \rangle$$

is 2-regular. However, the schedule $S = \langle (h_1, 14), (h_2, 1), (h_1, 2), (h_2, 3) \rangle$ is not.

When all heuristics are executed in the suspend-and-resume model, it is not difficult to show that for any schedule S, there is an α-regular schedule S_α whose expected running time on any instance is at most α times that of S (see Lemma 9).

General case

In the general case, the definition of α-regularity takes into account both the lengths of the runs of each heuristic, as well as the number of runs of each particular length that have

been performed. Additionally, the performance overhead increases from a factor of α to a factor of α^2.

For any $\alpha > 1$, we say that a profile $P = \langle \tau_1, \tau_2, \ldots, \tau_L \rangle$ is α-regular if, for all l ($1 \leq l \leq L$),

1. τ_l is a power of α (i.e., $\tau_l = \alpha^i$ for some integer i), and

2. the number of occurrences of the value τ_l in P is the floor of a power of α (i.e., $|\{l' : \tau_{l'} = \tau_l\}| = \lfloor \alpha^i \rfloor$ for some integer i).

For example, the profiles $\langle 4, 1, 1 \rangle$ and $\langle 4, 2, 1, 1, 1, 1 \rangle$ are 2-regular, but the profile $\langle 3, 1 \rangle$ is not (because 3 is not a power of 2), and the profile $\langle 2, 1, 1, 1 \rangle$ is not (because there are three runs of length 1).

A state $Y = \langle P_1, P_2, \ldots, P_k \rangle$ is α-regular if each profile P_j is α-regular. A schedule S is α-regular if it can be written as $S = S_1 \oplus S_2 \oplus \ldots \oplus S_L$, where

1. for any l, all actions in S_l are identical

2. for any l, $\mathcal{Y}(S_1 \oplus S_2 \oplus \ldots \oplus S_l)$ is an α-regular state.

The following example illustrates these definitions.

Example 4. Let $a_\tau = (h, \tau)$. Then the geometric restart schedule $S = \langle a_1, a_2, a_4, a_8 \rangle$ is 2-regular, as can be seen by writing it as $S = \langle a_1 \rangle \oplus \langle a_2 \rangle \oplus \langle a_4 \rangle \oplus \langle a_8 \rangle$. However, the restart schedule

$$S = \langle a_1, a_2, a_1, a_2, a_1, a_2, a_1, a_2, \ldots \rangle$$

is not 2-regular. This is because, if we write the schedule as $S = \langle a_1 \rangle \oplus \langle a_2 \rangle \oplus \langle a_1 \rangle \oplus \langle a_2 \rangle \oplus \ldots$, then the state

$$\mathcal{Y}(\langle a_1 \rangle \oplus \langle a_2 \rangle \oplus \langle a_1 \rangle \oplus \langle a_2 \rangle \oplus \langle a_1 \rangle \oplus \langle a_2 \rangle) = \langle \langle 2, 2, 2, 1, 1, 1 \rangle \rangle$$

is not 2-regular. However, by permuting the order of the runs in S, we can obtain a schedule

$$S' = \langle a_1, a_2, a_1, a_2, a_1, a_1, a_2, a_2, a_1, a_1, a_1, a_1, a_2, a_2, a_2, a_2, \ldots \rangle$$

that is again 2-regular.

The key property of α-regular schedules is given by the following lemma, which shows that an arbitrary schedule can be "rounded up" to an α-regular schedule while introducing at most a factor α^2 performance overhead (in the special case where all heuristics are executed in the suspend-and-resume model, there is only a factor α overhead). The proof is given in Appendix A.

Lemma 9. *For any schedule S and any $\alpha > 1$, there exists an α-regular schedule S_α such that, for any instance x, $\mathbb{E}[T(S_\alpha, x)] \leq \alpha^2 \cdot \mathbb{E}[T(S, x)]$. In the special case where all heuristics are executed in the suspend-and-resume model, $\mathbb{E}[T(S_\alpha, x)] \leq \alpha \cdot \mathbb{E}[T(S, x)]$.*

3.4 Offline Algorithms

In the offline setting we are given as input a set of instances $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$, and are given the distribution of $T(h, x)$ for all $h \in \mathcal{H}$ and $x \in \mathcal{X}$. Our goal is to compute the schedule

$$S^* = \arg\min_{S \in \mathcal{S}} \sum_{i=1}^{n} \mathbb{E}[T(S, x_i)] \ .$$

This offline problem is of interest for two reasons. First, in the learning-theoretic setting one must solve the offline problem in order to compute an optimal schedule for the set of training instances. Second, the algorithms we develop for solving the offline problem will serve as a basis for our online algorithms.

3.4.1 Computational complexity

If $|\mathcal{H}|$ is arbitrary, it is NP-hard to compute even an approximately optimal schedule. This is true even in the special case in which each heuristic in $\mathcal{H}$ is deterministic (so that we only need to consider task-switching schedules).

To see this, consider the special case in which all heuristics are deterministic and, for each instance $x \in \mathcal{X}$ and each heuristic $h \in \mathcal{H}$, $T(h, x) \in \{1, \infty\}$. In this case, an optimal task-switching schedule can be represented simply as a permutation of the k heuristics (where the permutation corresponds to a schedule that runs each heuristic for one unit of time, in the order specified by the permutation). If we identify each heuristic h with the set of instances $\{x \in \mathcal{X} : T(h, x) = 1\}$, then our goal is to order these sets from left to right so as to minimize the sum, over all $x \in \mathcal{X}$, of the position of the leftmost set that contains x. This is exactly the MIN-SUM SET COVER problem introduced by

Feige, Lovász, and Tetali [26], who proved that for any $\epsilon > 0$, achieving a $4 - \epsilon$ ratio for MIN-SUM SET COVER is NP-hard. Thus we have the following theorem.

Theorem 17. *For any $\epsilon > 0$, obtaining a $4 - \epsilon$ approximation to the optimal schedule is NP-hard, even in the special case when each heuristic $h \in \mathcal{H}$ is deterministic and $T(h, x) \in \{1, \infty\}$ for all $h \in \mathcal{H}$ and all $x \in \mathcal{X}$.*

In light of previous work on restart schedules, it is natural to ask what happens in the special case when $\mathcal{H}$ contains a single randomized heuristic h. If, additionally, $\mathcal{X}$ contains a single problem instance (or if the distribution of $T(h, x)$ is the same for all $x \in \mathcal{X}$) then the problem of computing an optimal schedule is trivial, as shown in Theorem 16. However, in the general case in which $T(h, x)$ may have a different distribution for each $x \in \mathcal{X}$, the problem appears to be more complex. Although we have not been able to determine whether the offline optimization problem is NP-hard in this special case, we do believe that designing an algorithm to solve it is non-trivial. One simple idea that does *not* work is to compute an optimal restart schedule for the single distribution that results from averaging the distribution of $T(h, x)$ for each $x \in \mathcal{X}$. To see the flaw in this idea, consider the following example with two instances, x_1 and x_2: $T(h, x_1)$ equals 1 with probability $\frac{1}{2}$ and equals 1000 with probability $\frac{1}{2}$, while $T(h, x_2) = 1000$ with probability 1. The optimal restart schedule for the averaged distribution is the uniform schedule $\langle 1, 1, 1, \ldots \rangle$, however this schedule *never* solves x_2.

3.4.2 Greedy approximation algorithm

We first consider the greedy approximation algorithm from Chapter 2. As mentioned in §3.1.4, equation (3.1) shows that minimizing $\sum_{i=1}^{n} \mathbb{E}[T(S, x_i)]$ is equivalent to minimizing $c(p, S)$, where $p(S) = \frac{1}{n} \sum_{i=1}^{n} p_{x_i}(S)$.

For our purposes, the greedy schedule $G = \langle g_1, g_2, \ldots \rangle$ can be defined inductively as follows: $G_1 = \langle\rangle$, $G_j = \langle g_1, g_2, \ldots, g_{j-1} \rangle$ for $j > 1$, and

$$g_j = \operatorname*{arg\,max}_{(h,\tau) \in \mathcal{H} \times \mathbb{R}_{>0}} \left\{ \frac{p(G_j \oplus \langle (h, \tau) \rangle) - p(G_j)}{\tau} \right\}. \tag{3.5}$$

We may stop adding actions to G once we reach a j such that $p(G_j) = 1$ (or once we reach a j such that $p(G_j) \geq 1 - \delta$ for some small $\delta > 0$). The time required to compute each action g_j is not prohibitive in general. If all heuristics are deterministic and there are only n instances, then for each heuristic h we need to consider at most n action durations on each iteration of the greedy algorithm (because there are at most n values of τ where

$p\left(G_j \oplus \langle (h,\tau) \rangle\right)$ changes). Alternatively, it is not hard to show that requiring τ to be a power of some $\alpha > 1$ can increase the cost by at most a factor of α. If we restrict our attention to action durations that are powers of α between τ_{min} and τ_{max}, then at most $k\left(1 + \log_\alpha \frac{\tau_{max}}{\tau_{min}}\right)$ evaluations of p are necessary.

We showed in §3.1.4 that p satisfies conditions sufficient for the greedy algorithm's approximation guarantees to apply. Thus, we obtain the following as a corollary of Theorem 4.

Corollary 2.

$$\sum_{x \in \mathcal{X}} \mathbb{E}\left[T\left(G, x\right)\right] \leq 4 \min_{S \in \mathcal{S}} \sum_{x \in \mathcal{X}} \mathbb{E}\left[T\left(S, x\right)\right] \ .$$

Remark 1. Corollary 2 and the definition of G imply that, from the point of view of worst-case approximation guarantees, the suspend-and-resume model provides no advantage over the restart model. To see this, imagine that $\mathcal{H}$ contains two copies of each underlying heuristic: one executed in the suspend-and-resume model, and one executed in the restart model. If ties are broken appropriately, the action g_1 could use a heuristic executed in the restart model (because the choice of model does not affect $p(\langle a \rangle)$ for any single action a). Inductively, the entire schedule G could only use heuristics executed in the restart model, and still provide a 4 approximation. As shown in §3.4.1, achieving a $4 - \epsilon$ approximation (for any $\epsilon > 0$) is NP-hard, even when all heuristics are deterministic, and regardless of what model the heuristics are executed in. Thus, in terms of the worst-case approximation ratio, there is no penalty associated with keeping only a single run in memory at a time.

Recall from Chapter 2 that the greedy schedule also (approximately) maximizes $p(S_{\langle T \rangle})$, for certain values of T. In particular, we obtain the following as a corollary of Theorem 3. The corollary shows that, in addition to approximately minimizing average expected running time, the greedy schedule approximately maximizes the expected fraction of instances solved in time $\leq T$, for certain values of T.

Corollary 3. *Fix an integer L, and let $T = \sum_{j=1}^{L} \tau_j$, where $g_j = (h_j, \tau_j)$. Then*

$$p\left(G_{\langle T \rangle}\right) \geq \left(1 - \frac{1}{e}\right) \max_{S \in \mathcal{S}} \left\{p(S_{\langle T \rangle})\right\} \ .$$

Also recall from Chapter 2 that we defined an improved greedy schedule G', which has the same approximation guarantees as G in terms of minimizing cost. In the special case where $\mathcal{H}$ contains a single (possibly randomized) heuristic and $\mathcal{X}$ contains a single instance, the greedy schedule G' is *optimal*, as the following theorem shows. The proof is given in Appendix A.

Theorem 18. *If $\mathcal{H}$ contains a single (randomized) heuristic and $\mathcal{X} = \{x\}$ contains a single instance, then*

$$\mathbb{E}\left[T\left(G, x\right)\right] = \min_{S \in \mathcal{S}} \mathbb{E}\left[T\left(S, x\right)\right] .$$

3.4.3 Shortest path algorithms

Although the problem of computing an optimal schedule is NP-hard when $k = |\mathcal{H}|$ is part of the input, we might hope to find an algorithm that runs in polynomial time for any fixed k. As shown in §3.3.2, the optimal schedule within a restricted set of schedules can be found by computing a shortest path in an appropriate state-space graph. In this section we use this idea to obtain approximation algorithms for computing task-switching schedules and restart schedules. For the case of task-switching schedules, similar (though not identical) dynamic programming algorithms were given independently by Petrik [67] and by Sayag *et al.* [73].

Task-switching schedules

We first describe how to compute (approximately) optimal task-switching schedules. Recall that a task-switching schedule is a schedule for a set of *deterministic* heuristics $\mathcal{H}$, where each heuristic is executed in the suspend-and-resume model.

Let B be an artificial bound on the amount of time we are allowed to run any heuristic. We require that, for each instance $x \in \mathcal{X}$, there is always some $h \in \mathcal{H}$ such that $T\left(h, x\right) \leq B$. Also, we assume without loss of generality that $T\left(h, x\right) \geq 1$ for all $x \in \mathcal{X}$ and all $h \in \mathcal{H}$.

Fix some desired approximation ratio $\alpha > 1$. Define a state-space graph $G^{ts}_{\alpha,B} = \langle V, E, S_e \rangle$ inductively as follows. The vertex set contains the empty state $Y^{\emptyset}$. Let $Y = \langle \langle \tau_1 \rangle, \langle \tau_2 \rangle, \ldots, \langle \tau_k \rangle \rangle \in V$ be a state in the vertex set. For each j such that $\tau_j < B$, the vertex contains the state $Y' = \langle \langle \tau_1 \rangle, \langle \tau_2 \rangle, \ldots, \langle \tau_{j-1} \rangle, \langle \tau'_j \rangle, \langle \tau_{j+1} \rangle, \ldots, \langle \tau_k \rangle \rangle$, where $\tau'_j = \max\left\{1, \alpha\tau_j\right\}$. Additionally, the edge set contains the edge $e = \langle Y, Y' \rangle$, where $S_e(e)$ is the schedule containing the single action $(h_j, \tau'_j - \tau_j)$. Finally, if $\tau_j \geq B$ for all j, then there is an edge from Y to v^*, labeled with the empty schedule.

By construction, any α-regular task-switching schedule that runs each heuristic for time at most B corresponds to a path from $Y^{\emptyset}$ to v^* through this graph, and the weight of the path is $\sum_{x \in \mathcal{X}} T\left(S, x\right)$ (where the weights are assigned according to equation (3.4)). Thus, by Lemmas 8 and 9, an α-approximation to the optimal task-switching schedule can be obtained by computing a shortest path from $Y^{\emptyset}$ to v^* in the graph $G^{ts}_{\alpha,B}$.

To bound the time complexity, first note that there are at most $2 + \lceil \log_\alpha B \rceil$ choices[3] for each value τ_j in a state $Y = \langle \langle \tau_1 \rangle, \langle \tau_2 \rangle, \ldots, \langle \tau_k \rangle \rangle$ that appears in the vertex set, so $|V| \leq (2 + \lceil \log_\alpha B \rceil)^k$. Because each vertex has at most k outgoing edges, $|E| \leq k\,|V|$. The time required to compute a shortest path is dominated by the time required to assign weights to the edges. Each edge weight is a sum of $|\mathcal{X}|$ per-instance weights, each of which can be computed in time $O(1)$. Thus we have proved the following theorem.

Theorem 19. *For any approximation ratio $\alpha > 1$ and any budget B, an α-approximation to the optimal task-switching schedule for a set of instances $\mathcal{X}$ can be found by computing a shortest path in the state space graph $G^{ts}_{\alpha,B} = \langle V, E, S_e \rangle$, where $|V| \leq (2 + \lceil \log_\alpha B \rceil)^k$ and $|E| \leq k\,|V|$. The overall time complexity is $O\left(nk(2 + \lceil \log_\alpha B \rceil)^k\right)$, where $n = |\mathcal{X}|$.*

We now consider the problem of computing an optimal task-switching schedule. Given a set of instances $\mathcal{X}$, an optimal task-switching schedule can be found by computing a shortest path in a state-space graph $G_\mathcal{X}$ similar to the one just described. In the construction just described, a state whose j^{th} profile is $\langle \tau_j \rangle$ had an edge to a state that was identical except that the j^{th} profile is $\langle \tau'_j \rangle$, where τ'_j is the is the next largest power of α above τ_j. In $G_\mathcal{X}$, τ'_j will instead be the next largest member of the set $\{0\} \cup \{T(h_j, x) : x \in \mathcal{X}\}$. The set $\mathcal{S}_{G_\mathcal{X}}$ thus contains all schedules that will only stop running heuristic h_j if the time invested on h_j so far equals $T(h_j, x)$ for some instance x. Using an interchange argument, it can be shown that an optimal schedule must satisfy this condition. This fact, combined with the arguments leading up to the statement of Theorem 19, proves the following theorem, which was proved independently by Sayag *et al.* [73].

Theorem 20. *An optimal task-switching schedule for a set of instances $\mathcal{X}$ can be found by computing a shortest path in the state space graph $G_\mathcal{X} = \langle V, E, S_e \rangle$, where $|V| \leq (n+1)^k$ and $|E| \leq k\,|V|$, where $n = |\mathcal{X}|$. The overall time complexity is $O\left(nk(n+1)^k\right)$.*

Restart schedules

Lastly, we consider the case where $\mathcal{H}$ contains one or more (randomized) heuristics, each of which is executed in the restart model. In the special case $|\mathcal{H}| = 1$, the results in this section provide a way to compute approximately optimal restart schedules, however we allow $|\mathcal{H}|$ to be arbitrary.

As in the previous section, we assume $T(h, x) \geq 1$ (with probability 1) for all $h \in \mathcal{H}$ and all $x \in \mathcal{X}$, and we require an artificial bound B on the *total* time that any single heuristic can be run (note that this time may be spread across multiple runs). When using

[3]For example, if $\alpha = 2$ and $B = 4$, the possible choices are 0, 1, 2, and 4.

randomized heuristics, it may not be possible to solve a problem instance with certainty in any finite time. We will confine our attention to schedules that, for any $x \in \mathcal{X}$, solve x with probability at least $1 - \delta$, for some $\delta \in (0, 1)$. Letting $\mathcal{S}_\delta$ denote the set of schedules that have this property, our goal is to compute (an approximation to) the schedule

$$S^* = \arg\min_{S \in \mathcal{S}_\delta} \sum_{x \in \mathcal{X}} \mathbb{E}\left[\min Bk, T\left(S, x\right)\right] \ .$$

(In other words, we charge a schedule time Bk for instances it does not solve after running each heuristic for time B.)

In this setting, we obtain a quasi-polynomial time approximation scheme by combining Lemmas 8 and 9. Specifically, we obtain an α^2-approximation to the optimal restart schedule by computing a shortest path in a state space graph $G^{rs}_{\alpha,B}$ whose vertex set contains all the α-regular states in which each heuristic is run for time at most B. The construction is similar to the one described in the previous section, and is detailed in the proof of Theorem 21 in Appendix A. One difference from the construction in the previous section is that, to ensure that only schedules in $\mathcal{S}_\delta$ can form a shortest path from $Y^\emptyset$ to v^*, edges of the form $\langle Y, v^* \rangle$ are assigned infinite weight if performing the runs in Y does not solve each instance with probability at least $1 - \delta$.

The key to bounding the time complexity is to show that $|V|$, which equals the number of α-regular states in which each heuristic has been run for time at most B, is at most $B^{O(k \log_\alpha \log_\alpha B)}$, and that $|E| = O\left(\log_\alpha B \, |V|\right)$. We then show that, using precomputation, edge weights can be assigned in time $O\left(n\right)$ per edge.

Theorem 21. *Fix an approximation ratio $\alpha > 1$, a budget B, and an error tolerance $\delta > 0$. Then an α^2 approximation to schedule S^* may be found by computing a shortest path in a state-space graph $G^{rs}_{\alpha,B} = \langle V, E, S_e \rangle$, where $|V| = O\left(B^{O(k \log_\alpha \log_\alpha B)}\right)$ and $|E| = O\left(\log_\alpha B \, |V|\right)$. The overall running time is $O\left(n(\log_\alpha B) B^{O(k \log_\alpha \log_\alpha B)}\right)$, where $n = |\mathcal{X}|$.*

3.5 Generalization Bounds

To apply the offline algorithms just discussed, we might collect a set of problem instances to use as training data, compute an (approximately) optimal schedule for the training instances, and then use this schedule to solve additional test instances. Under the assumption that the training and test instances are drawn (independently) from a fixed probability distribution, we would then like to know how much training data is required so that (with

high probability) our schedule performs nearly as well on test data as it did on the training data. In our setting two distinct questions arise:

1. How many training instances do we need?
2. How many runs of each randomized heuristic h must we perform on each training instance x in order to estimate the distribution of $T(h, x)$ to sufficient accuracy?

We deal with each question separately in the following subsections.

In this section, we will confine our attention to schedules in the set $\mathcal{S}_G$, for some state-space graph $G = \langle V, E, S_e \rangle$ (although we will not necessarily find an optimal schedule for the training instances by computing a shortest path in G).

3.5.1 How many instances?

We first consider how many training instances are required to learn a schedule that is *probably approximately correct*, under the assumption that the run length distribution of each heuristic on each of the training instances is known exactly. In the next section, we show that, by running each heuristic on each training instance for a surprisingly small amount of time, we can obtain estimates of the length distribution that allow us to obtain the same guarantees as if we knew the run length distributions exactly.

Let $\{x_1, x_2, \ldots, x_m\}$ be a set of m training instances drawn independently at random from some distribution. For any edge $e \in E$ in the state-space graph $G = \langle V, E, S_e \rangle$, let $w(e, x)$ be defined as in equation (3.4), and let $\bar{\mu}(e) = \frac{1}{m} \sum_{i=1}^{m} w(e, x_i)$ be the sample mean value of $w(e, x)$ over the instances in the training set. Let $\mu(e) = \mathbb{E}[w(e, x)]$ be the expected edge weight for a random test instance x.

Recall from §3.3.2 that any schedule $S \in \mathcal{S}_G$ corresponds to a path $e_1, e_2, \ldots, e_L$ from $Y^{\emptyset}$ to v^* in G, such that for any instance x,

$$\mathbb{E}[\min\{\ell(S), T(S, x)\}] = W(S, x) \equiv \sum_{l=1}^{L} w(e_l, x) .$$

For any schedule S, let $\bar{\mu}(S) = \sum_{i=1}^{m} W(S, x_i)$ be the sample mean weight of the path corresponding to S and let $\mu(S) = \mathbb{E}[W(S, x)]$ be the expected value for a random test instance x.

The following lemma bounds errors in the estimates of $\mu(S)$ in terms of errors in the estimates of $\mu(e)$.

Lemma 10.

$$\max_{S \in \mathcal{S}_G} \left\{ \frac{\bar{\mu}(S) - \mu(S)}{\ell(S)} \right\} \leq \max_{e \in E} \left\{ \frac{|\bar{\mu}(e) - \mu(e)|}{\ell(S_e(e))} \right\} .$$

Proof. Fix an arbitrary schedule $S \in \mathcal{S}_G$, and let $e_1, e_2, \ldots, e_L$ be the corresponding path from $Y^{\emptyset}$ to v^*. Then $\bar{\mu}(S) = \sum_{l=1}^{L} \bar{\mu}(e_l)$ by construction, while $\mu(S) = \sum_{l=1}^{L} \mu(e_l)$ by linearity of expectation. Thus, letting r denote the maximum value of $\frac{|\bar{\mu}(e) - \mu(e)|}{\ell(S_e(e))}$, we have

$$|\bar{\mu}(S) - \mu(S)| \leq \sum_{l=1}^{L} |\bar{\mu}(e_l) - \mu(e_l)| \leq \sum_{l=1}^{L} r \cdot \ell(S_e(e_l)) = r \cdot \ell(S)$$

so $\frac{|\bar{\mu}(S) - \mu(S)|}{\ell(S)} \leq r$, as claimed. □

We now bound the probability that there exists an edge e such that $\frac{|\bar{\mu}(e) - \mu(e)|}{\ell(S_e(e))} > \epsilon$, for some $\epsilon > 0$. For any edge $e \in E$, $\bar{\mu}(e)$ is the average of m independent identically distributed random variables, each of which has range $[0, \ell(S_e(e))]$ and expected value $\mu(e)$. Thus by Hoeffding's inequality,

$$\mathbb{P}\left[\frac{|\bar{\mu}(e) - \mu(e)|}{\ell(S_e(e))} \geq \epsilon \right] \leq 2 \exp\left(-2m\epsilon^2\right) .$$

It follows that for any $\delta' > 0$, $m = O(\frac{1}{\epsilon^2} \ln \frac{1}{\delta'})$ training instances are required to ensure $\mathbb{P}\left[\frac{|\bar{\mu}(e) - \mu(e)|}{\ell(S_e(e))} \geq \epsilon \right] < \delta'$. Setting $\delta' = \frac{1}{|E|}$ and applying the union bound proves the following theorem.

Theorem 22. *Let $G = \langle V, E, S_e \rangle$ be a state-space graph. If the number of training instances m satisfies the inequality*

$$m \geq m_0(\epsilon, \delta, G) = O\left(\frac{1}{\epsilon^2} \ln \frac{|E|}{\delta} \right) \tag{3.6}$$

then the inequality

$$\max_{S \in \mathcal{S}_G} \left| \frac{\bar{\mu}(S) - \mu(S)}{\ell(S)} \right| \leq \epsilon$$

holds with probability at least $1 - \delta$.

Fix some $\bar{\epsilon} > 0$, and let $\ell_{max} = \max_{S \in \mathcal{S}_G} \ell(S)$. Plugging $\epsilon = \frac{\bar{\epsilon}}{\ell_{max}}$ into Theorem 22 shows that $O\left(\frac{\ell_{max}^2}{\bar{\epsilon}^2} \ln \frac{|E|}{\delta} \right)$ training instances suffice so that, with probability at least

$1-\delta$, every schedule's estimated expected cost is within $\bar{\epsilon}$ of its true mean. In particular, with probability at least $1-\delta$, the schedule in $\mathcal{S}_G$ that performs optimally on the training instances will have true expected cost at most $2\bar{\epsilon}$ worse than optimal.

Using Theorem 22, we can now obtain sample complexity bounds for any state-space graph of interest. In particular, putting Theorem 19 together with Theorem 22 yields the following corollary.

Corollary 4. *Let the function $m_0(\epsilon, \delta, G)$ be defined as in Theorem 22. Then*

$$m_0(\epsilon, \delta, G^{ts}_{B,\alpha}) = O\left(\frac{1}{\epsilon^2}\left(\ln\frac{1}{\delta} + k\log\log_\alpha B\right)\right).$$

Similarly, putting Theorem 21 together with Theorem 22 yields the following corollary.

Corollary 5. *Let the function $m_0(\epsilon, \delta, G)$ be defined as in Theorem 22. Then*

$$m_0(\epsilon, \delta, G^{ts}_{B,\alpha}) = O\left(\frac{1}{\epsilon^2}\left(\ln\frac{1}{\delta} + k\log B\log_\alpha\log_\alpha B\right)\right).$$

Corollaries 4 and 5 and Lemma 9 suggest the following procedure: compute an (approximately) optimal schedule for the m training instances (using any algorithm or heuristic whatsoever) then round the schedule up to an α-regular schedule, where α is chosen so that Corollary 4 or Corollary 5 applies for some desired ϵ and δ. The rounding step prevents overfitting, and introduces only a constant factor performance overhead (in particular, Lemma 9 shows that the rounding introduces at most a factor α overhead if all heuristics are executed in the suspend-and-resume model, and at most a factor α^2 overhead otherwise).

3.5.2 How many runs per instance?

We now consider how many runs of each heuristic h must be performed on each training instance x in order to estimate the distribution of the random variable $T(h, x)$ to sufficient accuracy. Let B denote the maximum time invested in any single heuristic by any schedule $S \in \mathcal{S}_G$. If h is deterministic, the distribution of $T(h, x)$ can be determined by performing a single run (for our purposes, this run can be performed with a time limit of B). Thus, our interest is in the case when h is randomized. Surprisingly, it will turn out that we need only to run each randomized heuristic for time $O(B\log B)$ in order to obtain sufficiently accurate estimates.

Given a training instance x, we will describe how to obtain a function $\bar{w} : E \times \mathcal{X} \rightarrow \mathbb{R}_{\geq 0}$ such that, for any edge e in the state space graph, $\mathbb{E}\left[\bar{w}(e,x)|x\right] = w(e,x)$. Note that because x is chosen at random, this implies

$$\mathbb{E}\left[\bar{w}(e,x)\right] = \mathbb{E}\left[\mathbb{E}\left[\bar{w}(e,x)|x\right]\right] = \mathbb{E}\left[w(e,x)\right] = \mu(e)$$

where $\mu(e)$ was defined in the previous section. Thus, for the purposes of proving Theorem 22, the estimates $\bar{w}(e,x)$ are as good as the true values $w(e,x)$.

We obtain the desired function $\bar{w}$ in two steps. First, we obtain unbiased estimates of the *failure probability* associated with each profile and heuristic. We then use the estimates of failure probabilities to obtain the desired estimates of edge weights.

For the purposes of this section, we will assume $T\left(h,x\right) \geq 1$ (with probability 1) for any heuristic h and any training instance x.

Estimating failure probabilities

Fix a training instance x and a heuristic h. Define the *failure probability* of a profile $P = \langle \tau_1, \tau_2, \ldots, \tau_L \rangle$ with respect to heuristic h as the probability that performing runs of lengths $\tau_1, \tau_2, \ldots, \tau_L$ of heuristic h does not yield a solution to x:

$$q_h(P) = \prod_{l=1}^{L} \mathbb{P}\left[T\left(h,x\right) > \tau_l\right] \ .$$

Let $\mathcal{P}_B$ denote the set of profiles of size $< B$, where each run is of length ≥ 1 (note that there can be at most B runs in such a profile). In this section our goal is to spend as little time as possible running h on x so as to obtain a function $\bar{q}_h : \mathcal{P}_B \rightarrow [0,1]$ with the following property: for any profile P, $\mathbb{E}\left[\bar{q}_h(P)\right] = q_h(P)$ (i.e., $\bar{q}_h(P)$ is an unbiased estimate of $q_h(P)$ for all $P \in \mathcal{P}_B$).

To obtain such a function, we perform B independent runs of h on x, where the i^{th} run is performed with a time limit of $\frac{B}{i}$. Note that the total time required for all runs of h is at most $\sum_{i=1}^{B} \frac{B}{i} = O(B \log B)$. Let T_i be the time the i^{th} run *would have taken* if it had been performed with no time limit (whereas we only have knowledge of $\min\left\{T_i, \frac{B}{i}\right\}$), and call the tuple $\mathcal{T} = \langle T_1, T_2, \ldots, T_B \rangle$ a *trace*. For any profile $P = \langle \tau_1, \tau_2, \ldots, \tau_L \rangle$, we say that $\mathcal{T}$ encloses P if $T_i > \tau_i$ for all i, $1 \leq i \leq L$ (see Figure 3.2). Our estimate is

$$\bar{q}(P) = \begin{cases} 1 & \text{if } \mathcal{T} \text{ encloses } P \\ 0 & \text{otherwise.} \end{cases}$$

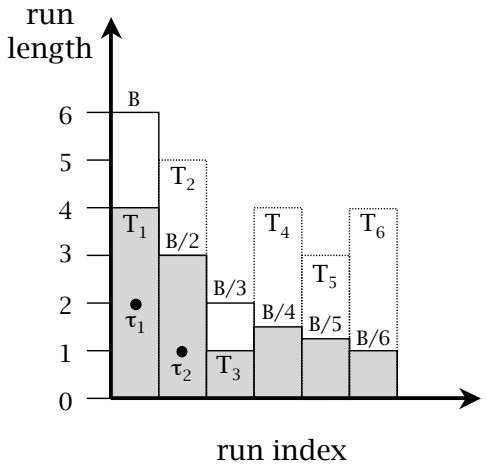

Figure 3.2: An illustration of our estimation procedure. The profile $P = \langle \tau_1, \tau_2 \rangle$ (dots) is enclosed by the trace $\langle T_1, T_2, T_3, T_4, T_5, T_6 \rangle$.

The estimate is unbiased (for profiles of size $< B$) because $\mathbb{E}\left[\bar{q}(P)\right] = \mathbb{P}\left[\bar{q}(P) = 1\right] = \prod_{i=1}^{B} \mathbb{P}\left[T\left(h, x\right) > \tau_i\right] = q(P)$. Furthermore, the estimate can be computed given only knowledge of $\min\{T_i, \frac{B}{i}\}$. This is true because if $P = \langle \tau_1, \tau_2, \ldots, \tau_L \rangle$ is a profile with size $< B$, then for each i we have $\tau_i < \frac{B}{i}$ (recall that the sequence $\langle \tau_1, \tau_2, \ldots, \tau_L \rangle$ is non-increasing by definition).

Thus we have proved the following lemma.

Lemma 11. *Given an instance x and a heuristic h, after running h for time $O\left(B \log B\right)$ (or time at most B, if h is deterministic), we can obtain a function $\bar{q}_h$ such that for any profile P, $\mathbb{E}\left[\bar{q}_h(P)\right] = q_h(P)$.*

Estimating schedule running times

Given the ability to obtain unbiased estimates of failure probabilities, we can readily obtain unbiased estimates of the running time of any schedule. Fix a training instance x. For any state $Y = \langle P_1, P_2, \ldots, P_k \rangle$, let $Q(Y)$ be the probability that none of the runs in Y yield a solution to x:

$$Q(Y) = \prod_{j=1}^{k} q_{h_j}(P_j)\ . \tag{3.7}$$

We can obtain an unbiased estimate of Q as follows. For each $h \in \mathcal{H}$, let $\bar{q}_h$ be an unbiased estimate of q_h, obtained (for example) using the procedure just described. Let $\bar{Q}$ be the function obtained by plugging $\bar{q}_{h_j}$ in for q_{h_j} in equation (3.7), for all j ($1 \leq j \leq k$). Given the instance x, the functions $\bar{q}_1, \bar{q}_2, \ldots, \bar{q}_k$ are independent. For any independent random variables A and B, $\mathbb{E}[AB] = \mathbb{E}[A] \cdot \mathbb{E}[B]$. Thus, for any state $Y = \langle P_1, P_2, \ldots, P_k \rangle$, we have

$$\mathbb{E}\left[\bar{Q}(Y)\right] = \prod_{j=1}^{k} \mathbb{E}\left[\bar{q}_{h_j}(P)\right] = \prod_{j=1}^{k} q_{h_j}(P) = Q(Y) \,. \tag{3.8}$$

Now fix an edge $e = (Y_1, Y_2)$. Let S_1 be a schedule such that $\mathcal{Y}(S_1) = Y_1$. Let $S = S_e(e)$, let $t_1 = \ell(S_1)$, and let $t_2 = \ell(S_1 \oplus S)$. As defined in equation (3.4),

$$w(e, x) = \int_{t=t_1}^{t_2} 1 - p_x((S_1 \oplus S)_{\langle t \rangle})\, dt = \int_{t=t_1}^{t_2} Q(\mathcal{Y}((S_1 \oplus S)_{\langle t \rangle}))\, dt \,.$$

Let $\bar{w}(e, x)$ be the estimate of $w(e, x)$ obtained by using $\bar{Q}$ in place of Q in this equation. We have

$$\begin{aligned}
\mathbb{E}[\bar{w}(e, x)] &= \mathbb{E}\left[\int_{t=t_1}^{t_2} \bar{Q}(\mathcal{Y}(S_1 \oplus S_{\langle t \rangle}))\, dt\right] \\
&= \int_{t=t_1}^{t_2} \mathbb{E}\left[\bar{Q}(\mathcal{Y}(S_1 \oplus S_{\langle t \rangle}))\right] dt && \text{(linearity of expectation)} \\
&= \int_{t=t_1}^{t_2} Q(\mathcal{Y}(S_1 \oplus S_{\langle t \rangle}))\, dt && \text{(equation (3.8))} \\
&= w(e, x) \,.
\end{aligned}$$

Thus, given an arbitrary instance x, after running each deterministic heuristic for time at most B and running each randomized heuristic for time at most $O(B \log B)$, we can obtain an unbiased estimate of $w(e, x)$ for any edge e the state space graph.

Finally, if S is a schedule corresponding to the path $e_1, e_2, \ldots, e_L$, then $\bar{W}(S, x) = \sum_{l=1}^{L} \bar{w}(e_l, x)$ is an unbiased estimate of $W(S, x)$ by linearity of expectation. Thus, if we redefine $\bar{\mu}(S) = \sum_{i=1}^{m} \bar{W}(S, x)$ to be the average *estimated* weight assigned to S over the m training instances, the following theorem follows by the same argument used to prove Theorem 22.

Theorem 23. *After running each deterministic heuristic for time at most B per training instance, and running each randomized heuristic for time at most $O(B \log B)$ per training instance, we can obtain a function $\bar{\mu}(S)$ for which Theorem 22 holds. That is, if $G =$*

$\langle V, E, S_e \rangle$ is a state-space graph and the number of training instances m satisfies the inequality

$$m \geq m_0(\epsilon, \delta, G) = O\left(\frac{1}{\epsilon^2} \ln \frac{|E|}{\delta}\right) \tag{3.9}$$

then the inequality

$$\max_{S \in \mathcal{S}_G} \left| \frac{\bar{\mu}(S) - \mu(S)}{\ell(S)} \right| \leq \epsilon$$

holds with probability at least $1 - \delta$.

Theorem 23 is significant for two reasons. First, it implies that if we optimize over the training instances in order to find a schedule with minimum *estimated* average expected running time on the training instances, we will obtain the same guarantees as if we had (somehow) found a schedule with minimum *actual* average expected running time on the training instances. Second, and perhaps more importantly, being able to obtain unbiased estimates of schedules' running times will be critical for making our algorithms work in the online setting.

Refining the estimation procedure

Although the procedure for estimating failure probabilities described in Lemma 11 is sufficient to obtain all our theoretical results, the estimate is somewhat crude in that the estimate of $q(P)$ (which is a probability) is always 0 or 1. The following lemma (proved in Appendix A) gives a more refined unbiased estimate which we will use in our experimental evaluation. As before, computing the estimate only requires knowledge of $\min\left\{T_i, \frac{B}{i}\right\}$ for each i.

Lemma 12. *For any profile $P = \langle \tau_1, \tau_2, \ldots, \tau_K \rangle$ of size $< B$, define $L_i(P) = \{i' : 1 \leq i' \leq B, \frac{B}{i'} > \tau_i\}$. Then the quantity*

$$\bar{q}_h(P) = \prod_{i=1}^{K} \frac{|\{i' \in L_i(P) : T_{i'} > \tau_i\}| - i + 1}{|L_i(P)| - i + 1}$$

is an unbiased estimate of $q_h(P)$ (i.e., $\mathbb{E}\left[\bar{q}_h(P)\right] = q_h(P)$).

3.6 Online Algorithms

One weakness of the results of §3.5 is that they assume we can draw training (and test) instances independently at random from a fixed probability distribution. In practice, the

distribution might change over time and successive instances might not be independent. For example, a SAT solver might be used to solve a set of problem instances derived from a particular application domain, and then be used to solve another set of instances from a different domain. To further illustrate this point, consider again the example from §3.1.1 of selecting a restart schedule for the SAT solver used by SATPLAN to solve one or more planning problems. To solve a particular planning problem, SATPLAN generates a sequence $\langle \sigma_1, \sigma_2, \ldots \rangle$ of Boolean formulae that are not at all independent. To optimize SATPLAN's performance, we would like to learn a restart schedule for the underlying SAT solver on-the-fly, without making strong assumptions about the sequence of formulae that are fed to it.

In this section we consider the problem of selecting schedules in a worst-case online setting. In this setting we are fed, one at a time, a sequence $\langle x_1, x_2, \ldots, x_n \rangle$ of problem instances to solve. Prior to receiving instance x_i, we must select a schedule $S_i \in \mathcal{S}$. As in previous sections, we confine our attention to schedules that run each heuristic for time at most B, for some budget $B > 0$. We then use S_i to solve x_i and incur cost C_i equal to the CPU time we spend running the heuristics in $\mathcal{H}$ on x_i, where the running time is artificially truncated at time Bk,so

$$\mathbb{E}\left[C_i\right] = \mathbb{E}\left[\min Bk, T\left(S_i, x_i\right)\right] \ .$$

As in Chapter 2, some form of truncation is necessary, for otherwise our algorithm might be forced to spend an arbitrarily large amount of time on one particular instance and we could not prove any meaningful bounds on its performance. After solving S_i we know the CPU time that was required, but nothing more (in particular, we do not know the distribution of $T\left(h, x_i\right)$ for each $h \in \mathcal{H}$). Our α-*regret*[4] after having received n instances is equal to

$$\mathbb{E}\left[\sum_{i=1}^{n} C_i\right] - \alpha \cdot \min_{S \in \mathcal{S}_0} \left\{ \sum_{i=1}^{n} \mathbb{E}\left[T\left(S, x_i\right)\right] \right\} \tag{3.10}$$

where $\mathcal{S}_0 \subseteq \mathcal{S}$ is some set of schedules, and where the expectation is over two sources of randomness: the random bits supplied to the heuristics in $\mathcal{H}$, but also any random bits used by our schedule-selection strategy. A strategy's worst-case regret is the maximum value of (3.10) over all instance sequences of length n.

[4] α-regret is unrelated to α-regularity.

3.6.1 Handling a small pool of schedules

Assume for the moment that we are given a set $\mathcal{S}_0$ of schedules to select from, where $|\mathcal{S}_0|$ is small enough that we would not mind using $O(|\mathcal{S}_0|)$ time or space for decision making. In this case one option is to treat our online problem as an instance of the "nonstochastic multiarmed bandit problem" and use the **Exp3** algorithm of Auer *et al.* [5] to obtain regret $O\left(Bk\sqrt{\frac{1}{n}|\mathcal{S}_0|}\right)$.

To obtain regret bounds with a better dependence on $|\mathcal{S}_0|$, we use a version of the "label-efficient forecaster" of Cesa-Bianchi *et al.* [17]. Applied to our online problem, this strategy behaves as follows. Given an instance x, with probability γ the strategy *explores* by computing an unbiased estimate of $\mathbb{E}\left[\min\left\{Bk, T\left(S, x\right)\right\}\right]$ for each $S \in \mathcal{S}_0$, using the estimation procedure described in §3.5.2. Recall that this estimation procedure requires us to spend time at most B running each deterministic heuristic, and time at most $\sum_{l=1}^{B} \frac{B}{l} = O\left(B \log B\right)$ running each randomized heuristic. We denote by F the total (maximum) running time over all $h \in \mathcal{H}$.

With probability $1-\gamma$, the strategy *exploits* by selecting a schedule at random from a distribution in which schedule S is assigned probability proportional to $\exp(-\eta\bar{c}(S))$, where $\bar{c}(S)$ is an unbiased estimate of $\sum_{l=1}^{i-1} \mathbb{E}\left[\min\left\{Bk, T\left(S, x_l\right)\right\}\right]$ (obtained by summing the estimates from each exploration round, and multiplying by $\frac{1}{\gamma}$) and η is a learning rate parameter. By Theorem 1 of Cesa-Bianchi *et al.* [17], the regret is at most

$$Bk\left(\frac{\ln|\mathcal{S}_0|}{\eta} + \frac{\eta \cdot n}{2\gamma}\right) + \gamma n F\,.$$

Optimizing γ and η yields the following theorem.

Theorem 24. *The label-efficient forecaster with learning rate* $\eta = \left(\frac{\ln|\mathcal{S}_0|}{n}\sqrt{\frac{2F}{Bk}}\right)^{2/3}$ *and exploration probability* $\gamma = \sqrt{\frac{\eta Bk}{2F}}$ *has 1-regret at most* $2Bkn^{\frac{2}{3}}\left(2\ln|\mathcal{S}_0|\frac{F}{Bk}\right)^{1/3}$.

3.6.2 Online shortest path algorithms

In the previous section we described how the label-efficient forecaster of Cesa-Bianchi *et al.* [17] can be used to converge to an optimal schedule within a set $\mathcal{S}_0$, but the time and space required for decision-making were both $O\left(|\mathcal{S}_0|\right)$. In this section, we describe how the label-efficient forecaster can be implemented more efficiently for certain sets $\mathcal{S}_0$ by exploiting the shortest path formulation discussed in §3.3.2.

Let $G = \langle V, E, S_e \rangle$ be a state-space graph, and let $\mathcal{S}_0 = \mathcal{S}_G$. Using the dynamic programming approach described by György *et al.* [36], we can maintain the distribution over schedules (i.e., paths) used by the label-efficient forecaster implicitly, by maintaining a weight for each edge in our graph. The space required, as well as the time required for each exploitation step, then become $O(|E|)$ rather than $O(|\mathcal{S}_0|)$, a potentially dramatic improvement.

The total decision-making time required by this approach is $O(n|E|)$. By using a "lazy" implementation of the exploitation steps, we can reduce the total decision-making time to $O(m|E|)$, where m is the number of exploration rounds (note that this is comparable to the amount of time required to solve the offline shortest path problem on the m training instances). The idea of this approach is to only resample a schedule whenever the distribution over schedules changes (i.e., sample once after each exploration step). On any particular round, this variant of the algorithm has the same expected behavior as the original version, and thus by linearity of expectation the overall worst-case regret bounds are unchanged. This approach has been used in other online algorithms (e.g., see [42]).

3.6.3 Online greedy algorithm

In §3.1.4 we showed that the online problem considered in this chapter satisfies the sufficient conditions required by the online greedy algorithm for MIN-SUM SUBMODULAR COVER from Chapter 2. In this section we flesh out the guarantees that the online greedy algorithm provides for the online problem considered in this chapter.

We will confine our attention to schedules in which each action duration is an integer between 1 and B, and in which each heuristic is run for total time at most B (where this time may be spread across multiple runs). We use $\mathcal{S}_B$ to denote the set of such schedules.

We first consider the online greedy algorithm $\mathbf{OG^P}$, defined in §2.4.4. Recall that this algorithm $\mathbf{OG^P}$ can be run in a *priced* feedback model in which, to receive access to the function p_x after solving instance x, we must pay a price that is then added to the regret. If all heuristics are deterministic, then after running each heuristic on instance x with a time limit of B, we are able to the determine value of $p_x(S)$ for any $S \in \mathcal{S}_B$. If one or more heuristics are randomized, then after running each randomized heuristic for time $O(B \log B)$, we can obtain an unbiased estimate of $p_x(S)$ for any $S \in \mathcal{S}_B$, as described in §3.5.2. As far as the analysis of $\mathbf{OG^P}$ is concerned, unbiased estimates of $p_x(S)$ are as good as the true values. Thus, we obtain the following bound as a corollary of Theorem 11. Here, as in §3.6.1, we use F to denote the maximum CPU time required by the estimation procedure described in §3.5.2.

Corollary 6. *Algorithm* $\mathbf{OG^P}$*, run with WMR as the subroutine experts algorithm, has 4-regret* $O\left((Bk\ln n)^{\frac{5}{3}}(\frac{F}{Bk}\ln|\mathcal{A}|)^{\frac{1}{3}}(n)^{\frac{2}{3}}\right)$ *with respect to the set* $\mathcal{S}_0 = \mathcal{S}_B$.

We next consider the *partially transparent* feedback model. Recall that in this model, after selecting a schedule S_i to use on instance x_i, one learns the value of $p_{x_i}(S_{i\langle t\rangle})$ for all $t > 0$. If all heuristics are deterministic, then the information revealed in this model is always available: $p_{x_i}(S_{i\langle t\rangle})$ equals 1 if $t \geq T(S_i, x_i)$, and equals 0 otherwise. If some heuristics are randomized, then knowing $T(S_i, x_i)$ does not reveal the exact value of $p_{x_i}(S_{i\langle t\rangle})$, but for any $t > 0$ it provides an unbiased estimate. Again, for the purposes of the analysis given in §2.4.4 for the partially transparent feedback model, these unbiased estimates are as good as the true values. Thus, we obtain the following bound as a corollary of Theorem 12.

Corollary 7. *Algorithm* $\mathbf{OG}$*, run with* **Exp3** *as the subroutine experts algorithm, has 4-regret* $O\left((Bk\ln n)^2\sqrt{nBk\ln Bk}\right)$ *with respect to the set* $\mathcal{S}_0 = \mathcal{S}_B$.

Given these two corollaries, it is natural to consider hybrid algorithms that always make use of the feedback provided in the partially transparent model, and occasionally pay the exploration cost F in order to obtain the information made available in the priced feedback model. We have analyzed such algorithms and consider them promising from a practical point of view. However, they do not yield regret bounds that improve on the minimum of the two bounds just stated (as a function of n, k, B, and F) by more than a constant factor.

3.7 Handling Optimization Problems

In this section we describe how the results of this chapter can be applied to optimization problems, as opposed to decision problems. In this context, we will assume that instead of simply returning a "yes" or "no" answer, our heuristics are anytime algorithms that return solutions of increasing quality over time. When constructing a schedule to use in combining such heuristics, the "cost" of a schedule should depend on how solution quality changes as a function of time (and should not, for example, depend only on the time required to find a *provably* optimal solution).

To better understand the motivation for the results in this section, consider Figure 3.3, which depicts the behavior of four solvers on an instance from the 2006 pseudo-Boolean evaluation (pseudo-Boolean optimization is another name for zero-one integer programming). On this instance, `bsolo` is the first solver to generate a feasible solution, while

`MiniSat 1.14` is the first to find an optimal solution and also the first to prove optimality. On the other hand, `SAT4J Heur.` generates a near-optimal solution very quickly but is unable to prove optimality (within the half hour time limit). By combining such heuristics using a task-switching schedule, we might hope to take advantage of their different strengths.

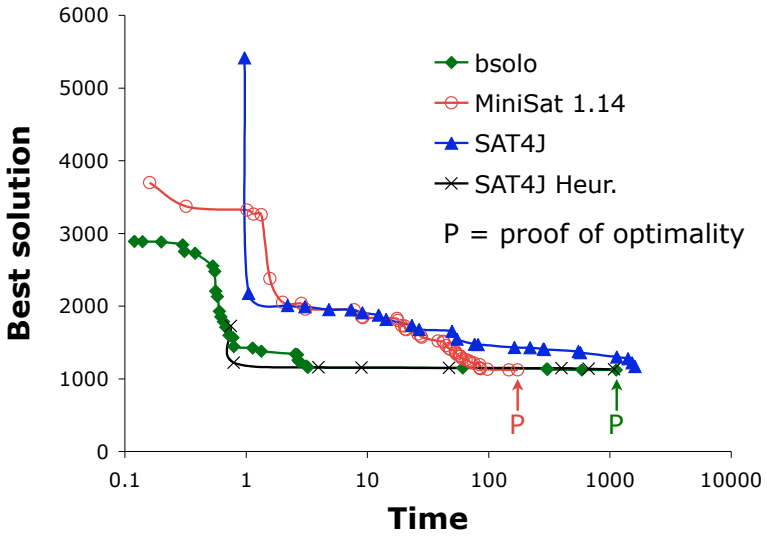

Figure 3.3: Behavior of four solvers on instance "normalized-mps-v2-20-10-lseu.opb" from the 2006 pseudo-Boolean evaluation.

We now describe a simple way to extend the results of this chapter to cost functions that account for solution quality. Let us define, for each instance, a set of objectives to achieve. For example, we might want to find a feasible solution to an optimization problem, and also to find a (provably) optimal solution. Then, for each instance $x \in \mathcal{X}$, we create a new set of fictitious instances $\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_k$, one for each of the k objectives. For each heuristic $h \in \mathcal{H}$, we define $T(h, \tilde{x}_i)$ to be the time that h requires to achieve the i^{th} objective. Thus, the average time a schedule or heuristic takes to "solve" the fictitious instances is simply the average time it takes to achieve each of the k objectives on the original instances. If some objectives are more important than others, we can assign different weights to the fictitious instances corresponding to each different objective. All the results of this chapter readily extend to weighted sets of instances.

We evaluate this approach experimentally in §3.9.5.

3.8 Exploiting Features of Instances

So far in this chapter we have imagined that we must select a schedule to use in solving instance x based *only* on our experience with previously-encountered instances, and not on any properties of the instance x itself. In practice, there may be quickly-computable features that distinguish one instance from another and suggest the use of different heuristics.

In this section, we describe how existing techniques for solving the so-called *sleeping experts* problem can be used to exploit features of instances in an attractive way. We will suppose that each problem instance is labeled with the values of M Boolean features. Given an instance, we may examine the values of the features (at zero cost) before selecting a schedule. Roughly speaking, the sleeping experts algorithms allow us to create variants of our online algorithms such that, even if regret is calculated using *only* the instances for which a particular feature is true, our online algorithm's usual regret bounds will still hold (this is true *simultaneously* for all features). For example, if each instance is labeled as either "random" or "industrial" and also labeled as either "small" or "large", then our online algorithm's usual regret bounds will hold, and they will also hold even if, when calculating regret, we *only* consider the "random" instances or only consider the "large" instances.

Background: the sleeping experts problem

The problem of combining different sources of expert advice was discussed in §2.4. Recall that in this problem, one has access to a set of M experts, each of whom gives out a piece of advice every day. On each day i, one must select an expert e_i whose advice to follow. Following the advice of expert j on day i yields a reward r_j^i. At the end of day i, the value of the reward r_j^i for each expert j is made public, and can be used as the basis for making choices on subsequent days. One's *regret* at the end of n days is equal to

$$\max_{1 \leq j \leq M} \left\{ \sum_{i=1}^{n} r_j^i \right\} - \sum_{i=1}^{n} r_{e_i}^i \ . \tag{3.11}$$

Note that the rewards assigned to an expert on the first $i - 1$ days do not necessarily imply anything about its reward on day i. Nevertheless, the randomized weighted majority algorithm [60] can be used to achieve worst-case regret $O\left(\sqrt{n \ln M}\right)$. Thus, as $n \to \infty$, the randomized weighted majority algorithm's average reward approaches (or exceeds) the maximum reward that could be received by following the advice of a single expert on all n days.

Now suppose that, on any particular day i, a given expert j may abstain from making a prediction (in this case $r_j^i = 0$). When choosing an expert on day i, one must pick an expert that made a prediction on that day (assume that at least one expert makes a prediction every day). Let $w_j^i = 1$ if expert j makes a prediction on day i; otherwise let $w_j^i = 0$. Define the "j regret" of an experts algorithm at the end of n days in analogy to (3.11), but only considering the days on which expert j made a prediction:

$$\max_{1 \leq l \leq M} \left\{ \sum_{i=1}^{n} w_j^i r_l^i \right\} - \sum_{i=1}^{n} w_j^i r_{e_i}^i \ . \tag{3.12}$$

Generalizing previous work by Freund *et al.* [29], Blum and Mansour [12] presented an algorithm for selecting experts in this setting whose j regret is $O\left(\sqrt{n \log M} + \log M\right)$, simultaneously for each j. In fact, the algorithm of Blum and Mansour is more general in that it allows for real-valued weights $w_j^i \in [0, 1]$, as opposed to the binary weights $w_j^i \in \{0, 1\}$ allowed in the original sleeping experts setting.

Exploiting features when selecting schedules

Our online schedule-selection algorithms can be combined with sleeping experts algorithms in a natural way. For each feature j, we create a copy $\mathcal{A}_j$ of the online schedule-selection algorithm that is only used for instances where feature j is true. We then treat each $\mathcal{A}_j$ as a sleeping expert whose loss equals the cost of the schedule that $\mathcal{A}_j$ selects (note that the sleeping experts problem can be defined equally well in terms of minimizing loss, rather than maximizing reward).

The code for algorithm **SE** illustrates this approach, which is well-known. As defined in this code, **SE** can only be run in the full-information feedback model, where after solving instance x_i we receive complete knowledge of the function p_{x_i}. However, as we discuss later in this section, **SE** can be modified to run in the priced feedback model (where receiving access to p_{x_i} entails paying a cost).

Algorithm SE
Input: sleeping experts algorithm $\mathcal{E}$, online schedule-selection algorithms $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_M$.

For i from 1 to n:

1. For each feature j that is true for instance x_i, use $\mathcal{A}_j$ to select a schedule $S_{i,j}$.
2. Use $\mathcal{E}$ to select a sleeping expert j, and select the schedule $S_i = S_{i,j}$.
3. Receive access to the function p_{x_i}.
4. For each feature j that is true for instance x_i:
   (a) feed back the function p_{x_i} to $\mathcal{A}_j$, and
   (b) feed back the cost of schedule $S_{i,j}$ to $\mathcal{E}$ as the loss it would have incurred had it selected sleeping expert j.

The performance of **SE** is summarized by the following fact, which follows immediately from the definition of α-regret together with the regret bound of the sleeping experts algorithm of Blum and Mansour [12].

Fact 2. *Consider running* **SE** *with the sleeping experts algorithm of Blum and Mansour [12] as input, and with a set of subroutine schedule-selection algorithms that each have worst-case expected α-regret Δ on a sequence of n instances. Then, simultaneously for each feature j, it holds that* **SE** *has worst-case α-regret at most $\Delta + O\left(\sqrt{n \log M} + \log M\right)$ if regret is calculated using only the instances for which feature j is true.*

As defined, **SE** receives complete knowledge of the function p_{x_i} after selecting schedule S_i, and then uses this function to give feedback to each $\mathcal{A}_j$. Alternatively, **SE** can be run in the priced feedback model by only performing this step with some small exploration probability γ, as described in §3.6.3. In this setting, Fact 2 can be used in conjunction with Corollary 6 and Lemma 5 to obtain bounds on 4-regret that hold (simultaneously for all j) when regret is calculated using only the instances that have feature j.

Table 3.2: Solver competitions.

Competition	Venue	Domain
CASC-J3	CADE 2007	theorem proving
SMT-COMP'07	CAV'07	satisfiability modulo theories
SAT 2007	SAT 2007	Boolean satisfiability
MaxSAT-2007	SAT 2007	maximum satisfiability
PB'07	SAT 2007	zero-one integer programming
QBFEVAL'07	SAT 2007	quantified Boolean formulae
CPAI'06	CP 2006	constraint satisfaction
IPC-5	ICAPS 2006	A.I. planning

3.9 Experimental Evaluation

In this section we present an experimental evaluation of the offline and online algorithms described in this chapter. The bulk of our experimental evaluation consists of using data from recent solver competitions to determine how task-switching schedules constructed by our offline and online algorithms would have fared had they been entered in the competitions. In these experiments we consider both optimization and decision problems, and we exploit features of problem instances in order to improve performance. At the end of this section, we present experiments in which we construct restart schedules for a randomized heuristic.

3.9.1 Solver competitions

Each year, various computer science conferences hold solver competitions designed to assess the state of the art in some problem domain. In these competitions, each submitted solver is run on a sequence of problem instances, subject to some per-instance time limit. Solvers are awarded points based on the instances they solve, and prizes are awarded to the highest-scoring solvers. Many competitions are divided into tracks corresponding to different categories of instances.

In this section we describe experiments performed using data from the eight solver competitions listed in Table 3.2. Each of these competitions is held either annually or bi-annually, and the competitions listed in Table 3.2 are the most recent competitions that had taken place at the time of writing. We now provide a brief description of each of the

competitions and problem domains.

1. **CASC-J3.** Theorem proving is the task of finding a proof that a given theorem follows from a given set of axioms, or refuting the theorem. The annual CASC theorem prover competition evaluates the performance of theorem provers over various logics.

2. **SMT-COMP'07.** Satisfiability modulo theories is the task of determining whether a logical formula is true with respect to a background theory expressed in classical first-order logic with equality. SMT solvers are generally used to solve hardware and software verification problems, where typical background theories include the theories of real and integer arithmetic, and the theories of various data structures such as arrays and fixed size bit vectors.

3. **SAT 2007.** Boolean satisfiability is the task of determining whether there exists an assignment of truth values to a set of Boolean variables that satisfies each clause (disjunction) in set of clauses. SAT solvers are used as subroutines in state-of-the-art algorithms for hardware and software verification and A.I. planning. The SAT 2007 competition included industrial, random, and hand-crafted benchmarks.

4. **Max-SAT 2007.** Maximum satisfiability is the optimization problem of finding an assignment of truth values to a set of Boolean variables that maximizes the number of satisfied clauses in a given set of clauses. The 2007 Max-SAT evaluation contained weighted and unweighted Max-SAT instances that encoded various optimization problems, including graph-theoretic problems and constraint satisfaction problems.

5. **PB'07.** Pseudo-Boolean optimization is the task of minimizing a function of zero-one variables subject to algebraic constraints, also known as zero-one integer programming. On many benchmarks, pseudo-Boolean optimizers (which are usually based on SAT solvers) outperform general integer programming packages such as CPLEX [2]. The PB'07 evaluation included both optimization and decision (feasibility) problems from a large number of domains, including formal verification and logic synthesis, as well as various numerical and graph-theoretic problems.

6. **QBFEVAL'07.** Determining whether a quantified Boolean formula (QBF) is true or false is the canonical PSPACE-complete problem. The 2007 QBF solver evaluation included instances derived from A.I. planning and formal verification problems, as well as various other problems.

7. **CPAI'06.** Constraint satisfaction problems entail finding an assignment of values to a set of discrete variables so as to satisfy a set of arbitrary discrete constraints. In the decision version of the problem, the goal is to determine whether there exists an assignment that satisfies all the constraints, whereas in the optimization version of the problem, the goal is to find an assignment that satisfies as many constraints as possible. The CPAI'06 competition included both decision and optimization problems, the bulk of which were generated randomly from various distributions.

8. **IPC-5.** A.I. planning is the problem of finding a sequence of actions (called a plan) that leads from a starting state to a desired goal state, according to some formally-specified model of how actions affect the state of the world. The makespan of a plan is the number of steps in the plan, treating actions that can be performed simultaneously as a single step. In the *optimal planning* track of IPC-5, the model of the world is specified in the STRIPS language and the goal is to find a plan with (provably) minimum makespan. The optimal planning benchmarks of IPC-5 require solving tasks such as finding a sequence of biochemical reactions that produce a desired set of substances, moving packages between locations subject to time and spatial constraints, and scheduling manufacturing operations.

3.9.2 Experimental procedures

Our experiments for each solver competition followed a common procedure:

1. We determined the value of $T(h, x)$ for each heuristic h and benchmark instance x using data available on the competition web site (we did not actually run any of the solvers). Note that the solvers considered in these competitions are deterministic (or randomized, but run with a fixed random seed), so $T(h, x)$ is simply a single numeric value. For optimization problems, we define $T(h, x)$ to be the time required to obtain a *provably* optimal solution (or to prove that the problem is infeasible). If a solver did not finish within the competition time limit, then $T(h, x)$ is undefined.

2. We discarded any instances that none of the solvers could solve within the time limit. (Clearly, no task-switching schedule could solve any such instance within the time limit either.)

Given a schedule S and instance x, we will not generally be able to determine the true value of $T(S, x)$, due to the fact that $T(h, x)$ is undefined for some heuristics $h \in \mathcal{H}$. Instead, we will measure the performance of S on x in terms of upper and lower bounds

on the true value of $T(S, x)$, computed as follows. The lower bound is simply the value

$$\min\{B, T(S, x)\}$$

where B is the competition time limit. Given knowledge of $\min\{B, T(h, x)\}$ for each $h \in \mathcal{H}$, we can determine the value of $\min\{B, T(S, x)\}$ exactly. The upper bound is the value of $T(S, x)$, computed after artificially setting $T(h, x) = \infty$ for heuristics that did not solve x within the competition time limit.

When evaluating offline schedule selection algorithms such as the greedy approximation algorithm from §3.4, we are concerned about the possibility of overfitting the solver competition data. To address this possibility, we use leave-one-out cross-validation. Leave-one-out cross-validation is performed as follows: for each instance x, we remove x from the data set and then run the offline algorithm on the remaining data to obtain a schedule to use in solving x. We then measure the average performance of these schedules over all $x \in \mathcal{X}$.

In these experiments, we consider schedules that execute all heuristics in the suspend-and-resume model, as well as schedules that execute all heuristics in the restart model (when the model is not mentioned explicitly, we use the suspend-and-resume model). Note that even if all available heuristics are deterministic, the restart model may be useful due to memory limitations. Recall from §3.4.2 that the greedy approximation algorithm can be used to produce a schedule optimized for execution under either model. Also recall that, by Remark 1, there is no loss associated with the restart model from the point of view of worst-case approximation guarantees.

3.9.3 Experiments with the shortest path algorithm

In this section we present experiments in which the number of heuristics is small enough that we can compute an optimal task-switching schedule using the shortest path algorithm described in Theorem 20 of §3.4.3. This allows us to evaluate the potential benefits of task-switching schedules, and also to determine how close the schedules returned by the greedy approximation algorithm are to optimality. The experiments described in this section use the data from the SAT 2005 competition. We used the 2005 data because the data from the SAT 2007 competition was not yet available at the time these experiments were performed (the SAT competition was not held in 2006).

For each of the three instance categories from the SAT 2005 competition (industrial, random, and hand-crafted), we computed an optimal task-switching schedule for the two

solvers that won first prize in the satisfiable and unsatisfiable subsets of that category.[5] We use only the top two solvers, because we found that computing an optimal task-switching schedule for three or more solvers was too computationally expensive to be practical (for the sets of benchmark instances considered here). When evaluating the top two solvers within each category we consider only the instances that belong to that category, and, as already mentioned, we discarded instances that neither of the solvers could solve within the time limit.[6]

Table 3.3: Results for the SAT 2005 competition.

Category (#Instances)	**Solver**	**Avg. CPU (s)** [lower,upper]	**Num. solved**
Industrial (268)	*Optimal schedule*	[793,793]	268
	Greedy schedule	[794,794]	268
	Greedy schedule (CV)	[810,810]	268
	`SatELiteGTI`	[958,∞]	267
	Parallel schedule	[1222,1264]	265
	`MiniSat 1.13`	[1759,∞]	250
Random (284)	*Optimal schedule*	[1015,1173]	261
	Greedy schedule	[1015,1173]	261
	Greedy schedule (CV)	[1050,1221]	260
	Parallel schedule	[1081,1325]	257
	`ranov`	[2026,∞]	209
	`kcnfs-2004`	[2874,∞]	167
Hand-crafted (403)	*Optimal schedule*	[483,538]	391
	Greedy schedule	[483,540]	391
	Parallel schedule	[542,643]	388
	Greedy schedule (CV)	[585,655]	386
	`Vallst`	[1095,∞]	343
	`SatELiteGTI`	[1214,∞]	350

[5]In the industrial category, the solver `SatELiteGTI` won first prize for both the satisfiable and unsatisfiable subsets, so we instead combined it with one of the second-place solvers.

[6]The time limit for the second stage of the SAT 2005 competition was 200 minutes for industrial instances and 100 minutes for random and hand-crafted instances.

Table 3.3 displays the upper and lower bounds on average CPU time, as well as the number of instances solved within the competition time limit, for various solvers: the top two solvers in each category, a schedule that simply runs these solvers in parallel, the optimal task-switching schedule, and the task-switching schedule returned by the greedy approximation algorithm. (As described in §3.9.2, we cannot determine the average CPU time required by each solver and schedule exactly, but we can compute upper and lower bounds.)

In all three categories, the optimal schedule improves on the original solvers in terms of average CPU time and in terms of the number of instances solved within the time limit. In terms of the upper bound on average CPU time, the improvement is unbounded. In terms of the lower bound on average CPU time, the improvement is by factors of 1.21, 2.00, and 2.27 for the industrial, random, and hand-crafted categories, respectively. The optimal schedule also improves on the naïve parallel schedule, both in terms of average CPU time and in terms of the number of instances solved within the time limit. Another interesting feature of Table 3.3 is that the schedules returned by the greedy approximation algorithm are very close to optimal (the average CPU time is within 0.2% of optimal in all three cases).

Some of these performance improvements are not surprising. In the random category, one of the two solvers (`kcnfs-2004`) is a complete solver, whereas the other (`ranov`) is based on local search and can only solve satisfiable formulae (in the other two categories, both solvers are complete). It thus seems natural that hybridizing `ranov` and `kcnfs-2004` could yield improved performance on a mixture of satisfiable and unsatisfiable instances. What is perhaps surprising is that the performance can be improved simply by interleaving the execution of the two solvers according to an appropriate schedule.

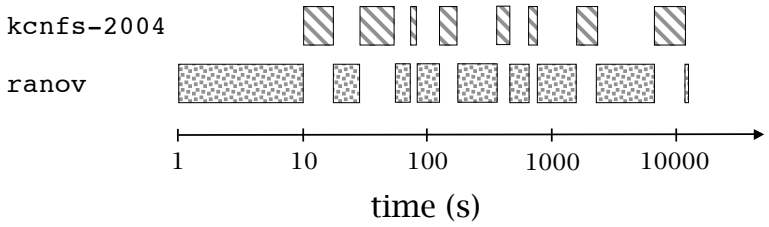

Figure 3.4: The optimal task-switching schedule for interleaving `kcnfs-2004` and `ranov`, the top two solvers in the *random* instance category.

Figure 3.4 illustrates the optimal task-switching schedule for interleaving the top two

solvers in the random instance category. As illustrated in the figure, the optimal schedule first runs `ranov` for about 10 seconds, then runs `kcnfs-2004` for about 7 seconds (note logarithmic scale), and so on.

To address the possibility of overfitting, we repeated the experiments with the greedy approximation algorithm using leave-one-out cross-validation, as described in §3.9.2. Under cross-validation, the lower bound on greedy schedule's average CPU time increased by about 2% in the industrial category, and by about 3.5% and 21% in the random and hand-crafted categories, respectively.

3.9.4 Experiments with a larger number of heuristics

In the previous section we saw that the performance of the greedy approximation algorithm was very close to optimal when using it to combine the top two solvers from the industrial, random, and hand-crafted tracks of the SAT 2005 competition. In this section, we present experiments involving a larger number of heuristics, using data from the IPC-5 A.I. planning competition.

Six planners were entered in the optimal planning track of the Fifth International Planning Competition (IPC-5). Each planner was run on 240 instances, with a time limit of 30 minutes per instance. On 110 of the instances, at least one of the six planners was able to find a (provably) optimal plan. As described in §3.9.2, we used the greedy algorithm to construct an approximately optimal task-switching schedule, given as input the completion times of each of the six planners on each of these 110 instances.

Table 3.4: Results for the optimal planning track of IPC-5.

Solver	**Avg. CPU** [lower,upper]	**Num. solved**
Greedy schedule	[307,358]	98
Greedy schedule (CV)	[315,434]	97
Greedy schedule (restart)	[332,426]	96
Greedy schedule (restart, CV)	[368,551]	95
Parallel schedule	[456,1244]	89
`SATPLAN`	[507,∞]	83
Parallel schedule (restart)	[527,2145]	89
`Maxplan`	[641,∞]	88

continued on next page...

Table 3.4 (continued from previous page)

Solver	**Avg. CPU** [lower,upper]	**Num. solved**
MIPS-BDD	[946,∞]	54
CPT2	[969,∞]	53
FDP	[1079,∞]	46
IPPLAN-1SC	[1437,∞]	23

Table 3.4 presents the results. In this table, the schedules marked with "(restart)" indicate schedules executed in the restart model, and the schedules marked with "(CV)" indicate the results of leave-one-out cross-validation. The schedule "Parallel (restart)" indicates a schedule that runs each of the k heuristics for T time units each, starting with $T = 1$ and repeatedly doubling T.

As Table 3.4 shows, the greedy schedules outperform the naïve parallel schedule (which simply runs all six planners in parallel) as well as each of the six individual planners, both in terms of (lower and upper bounds on) average CPU time and in terms of the number of instances solved within the 30 minute time limit. Note that, in contrast to the experiments in the previous section, the greedy schedules now outperform the naïve parallel schedule by a substantial factor, particularly in terms of the upper bound on average CPU time: the upper bound on the parallel schedule's average CPU time is about 3.5 times that of the greedy schedule in the suspend-and-resume model, and about 5 times that of the greedy schedule in the restart model.

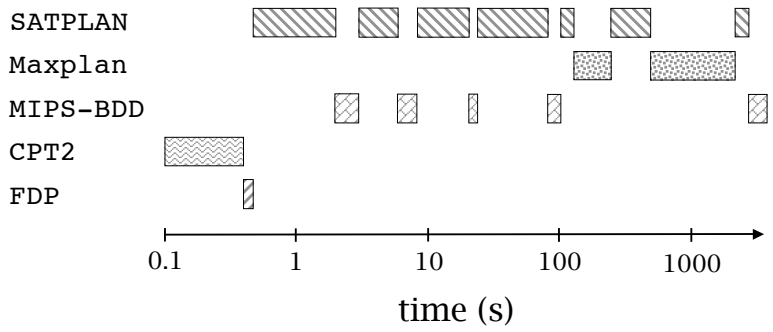

Figure 3.5: Greedy task-switching schedule for interleaving solvers from IPC-5.

Figure 3.5 shows the task-switching schedule constructed by the greedy approximation algorithm (the solver IPPLAN-1SC is not shown because it did not appear in the schedule). As indicated in the figure, the greedy schedule spends the majority of its time running SATPLAN, the solver that performed best in the competition. However, the first two solvers that the greedy schedule runs are CPT2 and FDP. Although these solvers did not perform as well as SATPLAN in the competition, there are some instances that they are able to solve very quickly, making it beneficial to perform short runs of these solvers initially.

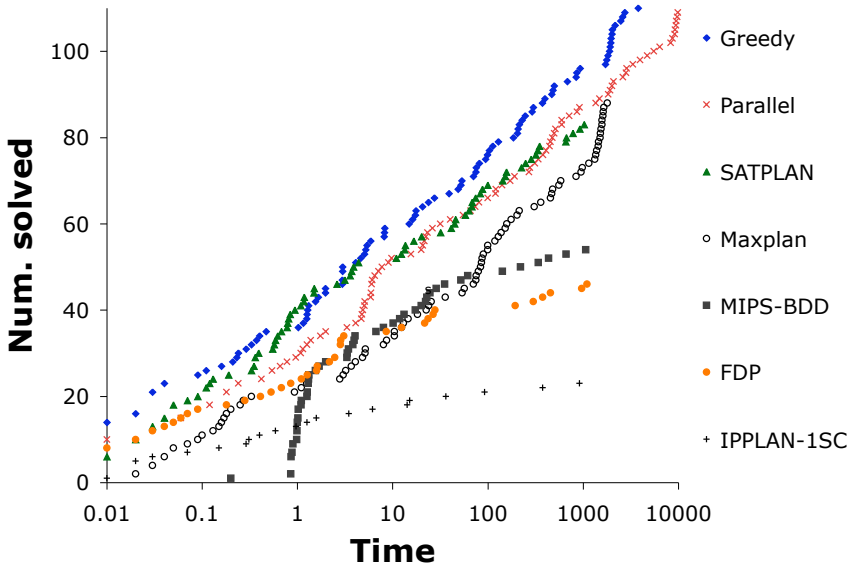

Figure 3.6: Number of benchmark instances from the IPC-5 A.I. planning competition solved by various solvers and schedules, as a function of time.

Figure 3.6 shows the number of instances solved by various solvers as a function of time: the six individual solvers, as well as the greedy and parallel schedules, executed in the suspend-and-resume model. As indicated in the figure, the greedy schedule not only outperforms the other schedules and solvers in terms of average CPU time, but outperforms them in terms of the number of instances solved within time T, for almost all choices of T.

3.9.5 Experiments with optimization heuristics

As described in §3.7, the results of this chapter can be applied to optimization as well as decision problems. Recall from §3.7 that the idea of this approach was to redefine the "cost" of a schedule to reflect how solution quality changes as a function of time, and that all our results carry over to this more general notion of schedule cost.

In this section, we demonstrate the power of this idea using data from the optimization tracks of the 2007 pseudo-Boolean evaluation and the CPAI'06 constraint programming competition.[7] Our experimental procedure is identical to the one used in the previous section, except that we now define the "cost" of a schedule to be the average of three quantities:

1. the time the schedule takes to find a feasible solution,
2. the time the schedule takes to find an optimal solution, and
3. the time the schedules takes to prove optimality (or to prove that the problem is infeasible).[8]

As discussed in §3.9.2, we discarded instances where none of the solvers were able to find a provably optimal solution (for these instances, we would not be able to evaluate the last two of the three quantities just listed).

We present detailed results for instances in the "small integers, linear constraints" category of the PB'07 evaluation, then summarize the results for the other categories and competitions. In many cases, our experiments yield schedules that outperform each of the solvers entered in the competition *simultaneously* in terms of each of the three objectives just discussed.

Table 3.5 shows the behavior of various solvers in terms of each of these three objectives, for instances in the "small integers, linear constraints" category of the PB'07 evaluation. As indicated in the table, no one solver is the best in terms of all three objectives: `bsolo3.0.17` is best in terms of the average time required to find an optimal solution and the average time to prove optimality, while `sat4jPseudoCP` is about 1.5 times slower at proving optimality but is about 1.6 times faster at finding a feasible solution.

[7] We do not consider optimization problems from the Max-SAT evaluation because the necessary data showing solver solution quality as a function of time was not collected as part of the Max-SAT evaluation.

[8] The time required to find a provably optimal solution is equal to the time required to find an optimal solution plus the time required to prove that no better solution exists. Note that the time required to prove optimality is often substantially larger than the time required to simply discover an optimal solution.

Table 3.5: Results for PB'07, optimization problems with small integers and linear constraints.

Solver	**Avg. CPU (s) to prove optimality (or infeasibility)**	**Avg. CPU (s) to find optimal solution**	**Avg. CPU (s) to find feasible solution**
Greedy	[218,345]	[164,243]	[32.54,65.09]
Greedy (CV)	[251,738]	[200,491]	[44.73,187]
Greedy (restart)	[262,506]	[190,326]	[38.79,79.00]
Greedy (restart, CV)	[295,1234]	[236,783]	[53.34,340]
Parallel	[381,1360]	[288,949]	[56.94,275]
Parallel (restart)	[474,2537]	[369,1749]	[87.29,517]
`bsolo3.0.17`	[629,∞]	[577,∞]	[270,∞]
`bsolo3.0.16`	[664,∞]	[608,∞]	[270,∞]
`minisat+1.14`	[756,∞]	[714,∞]	[250,∞]
`Pueblo1.4`	[890,∞]	[816,∞]	[229,∞]
`sat4jPseudoCP`	[961,∞]	[842,∞]	[164,∞]
`sat4jPseudoCPCls.`	[971,∞]	[856,∞]	[167,∞]
`sat4jPseudoRes.`	[972,∞]	[880,∞]	[270,∞]
`glpPB0.2`	[735,∞]	[735,∞]	[735,∞]
`PBS4_v2`	[1086,∞]	[974,∞]	[280,∞]
`PBS4`	[1086,∞]	[975,∞]	[281,∞]
`PB-clasp04-10`	[1025,∞]	[931,∞]	[410,∞]
`PB-clasp03-23`	[1168,∞]	[1117,∞]	[633,∞]
`oree0.1.2 alpha`	[1431,∞]	[1360,∞]	[614,∞]
`absconPseudo102`	[1399,∞]	[1281,∞]	[812,∞]
`wildcat-skc`	[1795,∞]	[1109,∞]	[593,∞]
`wildcat-rnp`	[1795,∞]	[1210,∞]	[702,∞]

As shown in Table 3.5, the greedy schedule significantly outperforms each of the solvers entered in the competition, simultaneously in terms of all three objectives. In fairness, we should also note that simply running all the solvers in parallel also outperforms each of the original solvers in terms of all three objectives, although by a smaller margin. However, the parallel schedule is inefficient, in part because many of the solvers have very similar behavior (e.g., the two versions of `bsolo` and `sat4JPseudoCP`). Accordingly, the performance of the parallel schedule is significantly worse than that of the

greedy schedule (even when evaluated under leave-one-out cross-validation) in terms of all three objectives, particularly in terms of the upper bounds on average CPU time.

Table 3.6 summarizes the results of all our optimization experiments. For each set of instances and for each of the three objectives, we define a *speedup factor* equal to the (lower bound on) average CPU time required by the fastest individual solver to achieve that objective, divided by the corresponding quantity for the greedy schedule, where the performance of the greedy schedule is evaluated under leave-one-out cross-validation. Note that in general, the three different speedup factors listed for each competition represent a comparison against three *different* solvers.

Table 3.6: Speedup factors for experiments with optimization heuristics.

Competition	**Category**	**Speedup factor (proving optimality)**	**Speedup factor (finding optimal solution)**	**Speedup factor (finding feasible solution)**
PB'07	Opt. small integers	2.50	2.89	3.67
	Opt. small integers, non-linear	1.60	1.30	1.44
	Opt. big integers	1.18	1.47	1.36
CPAI'06	Opt. binary constraints in extension	1.60	3.31	0.96
	Opt. n-ary constraints in extension	1.13	1.33	0.98

As the table shows, our strongest results were for the PB'07 evaluation: in all three categories, we were able to generate a schedule that simultaneously outperformed each of the original solvers in terms of each of the three objectives we considered: average time to find a feasible solution, average time to find an optimal solution, and average time required to prove optimality. Our results for the optimization tracks of the CPAI'06 competition are qualitatively similar, though not quite as strong. In each of the two categories considered in these experiments, we obtain a schedule that simultaneously outperforms each of the original solvers in terms of the time required to find an optimal solution and

the time required to prove optimality, and simultaneously performs almost as well as the best individual solver in terms of the time required to find a feasible solution.

3.9.6 Experiments with online algorithms

In this section we compare the performance of various online schedule-selection algorithms by using them to combine solvers from the three tracks of the 2007 SAT competition: industrial, random, and hand-crafted. Within each instance category, we compared the performance of the online algorithms to the offline greedy schedule, to the individual solver with the lowest (lower bound on) average CPU Time, and to a schedule that ran each solver in parallel at equal strength. All the schedules considered in these experiments are executed in the suspend-and-resume model.

We compare the performance of four online algorithms:

1. *Online greedy (WMR):* the online greedy algorithm **OG** from Chapter 2, run in the full-information feedback model (i.e., after solving an instance, the times required by all solvers are revealed). When running in the full-information feedback model, we use the self-tuning randomized weighted majority algorithm of Auer & Gentille [6] as the subroutine experts algorithm used by **OG**.

2. *Online greedy (Exp3):* the online greedy algorithm **OG**, run in the opaque feedback model (i.e., after using a schedule to solve an instance, we only learn the CPU time required by that schedule to solve that instance). When running in this feedback model, we use a self-tuning version of the **Exp3** algorithm [5] as the subroutine experts algorithm used by **OG**. Recall from §3.6.3 that, when all heuristics are deterministic, the opaque feedback model and the partially transparent feedback model are equivalent, and thus the information needed to compute the payoffs to **Exp3** is available.

3. *WMR:* An online algorithm that uses the self-tuning randomized weighted majority algorithm of Auer & Gentille [6] to select a *single* heuristic to use in solving each problem instance. Specifically, we treat each heuristic as an expert whose loss (negative payoff) equals the time required by the heuristic to solve that instance (capped at the competition time limit).

4. *Exp3:* An online algorithm identical to the one just described, except that the self-tuning version of **Exp3** is used as the experts algorithm.

We ran the online greedy algorithm with parameter $L = 25$. The time units used by the online greedy algorithm are the competition time limit divided by L. In implementing the algorithm, we made use of the two modifications described in §2.4.6, namely (*i*) ruling out actions that have already been performed when sampling from the distribution returned by each experts algorithm and (*ii*) using dependent rather than independent probabilities when converting the experts algorithms' choices into a schedule.

Tables 3.7 summarizes the results of these experiments. The rows labeled *Online greedy w/features* and *Offline greedy w/features (CV)* refer to experiments described in the next section in which we make use of instance features.

In each category, the offline greedy schedule (evaluated under leave-one-out cross-validation) outperforms each individual solver as well as the naïve parallel schedule, both in terms of the number of instances solved within the time limit and in terms of (upper and lower bounds on) average CPU time. The same is true of the online greedy schedule in the full information setting. Under the opaque feedback model, the performance of the online greedy algorithm is not as strong. In some cases it does not outperform the best individual solver, while in other cases it does not outperform the naïve parallel schedule. In the section that follows, we evaluate how the various online algorithms behave asymptotically, as the number of instances grows large.

Table 3.7: Results for the SAT 2007 competition.

Category (#Instances)	**Solver**	**Avg. CPU (s)** [lower,upper]	**Num. solved**
Industrial (166)	*Offline greedy w/features (CV)*	[1872,3180]	151
	Online greedy w/features	[2014,4336]	149
	Online greedy (WMR)	[2215,4196]	149
	`Fastest solver`	[2438,∞]	139
	Offline greedy (CV)	[2464,4271]	148
	WMR	[2617,∞]	139
	Online greedy (Exp3)	[2765,6858]	134
	Parallel	[3176,7003]	132
	Exp3	[3574,∞]	120
Random (411)	*Offline greedy w/features (CV)*	[963,2204]	380
	Online greedy w/features	[1044,3262]	365
	Online greedy (WMR)	[1304,4261]	347
	Offline greedy (CV)	[1337,3252]	344

continued on next page...

Table 3.7 (continued from previous page)

Category (#Instances)	**Solver**	**Avg. CPU (s)** [lower,upper]	**Num. solved**
	Parallel	[1775,7571]	302
	Online greedy (Exp3)	[2050,8127]	294
	`Fastest solver`	[2157,∞]	252
	WMR	[2184,∞]	255
	Exp3	[2835,∞]	191
Hand-crafted (129)	*Offline greedy w/features (CV)*	[1237,2518]	113
	Offline greedy (CV)	[1344,2715]	110
	Online greedy w/features	[1430,2947]	108
	Online greedy (WMR)	[1513,3452]	107
	`Fastest solver`	[1847,∞]	98
	Parallel	[1855,4866]	95
	WMR	[1903,∞]	96
	Exp3	[2012,∞]	96
	Online greedy (Exp3)	[2041,5148]	92

Asymptotic behavior of online algorithms

To more thoroughly evaluate the behavior of the online algorithms in the limited-feedback setting, we performed experiments involving a much larger number of benchmark instances. To do so, we sampled 100,000 benchmark instances independently (with replacement) from the set of 411 benchmarks from the *random* category of the SAT 2007 competition. We then ran each of the online algorithms on this sequence of 100,000 benchmarks.

It is worth pointing out that, if we *knew* that each instance in the sequence was being drawn independently from a fixed distribution (as is the case in these experiments), we could design simpler algorithms and achieve better performance (e.g., by using the first m instances as training data, for some appropriate value of m, and then using the schedule that performs best on the training instances to solve the remaining instances). However, the intent of these experiments it to evaluate the behavior of the online algorithms on long sequences of instances, which will not in general have this property.

In addition to evaluating the online algorithms considered in the previous section, we

evaluated the variant of the online greedy algorithm designed for operation in the priced feedback model, as described in §3.6.3. Recall that this model works as follows: when given an instance, the online algorithm may either select a schedule, or may choose to *explore* by running all the solvers until they find a solution (or until the competition time limit expires). As described in §3.6.3, the online greedy algorithm $\mathbf{OG}^{\mathbf{P}}$ can be applied in this setting, and simply explores with a fixed probability on each instance. We experimented with three values of the exploration probability: 0.1, 0.01, and 0.001.

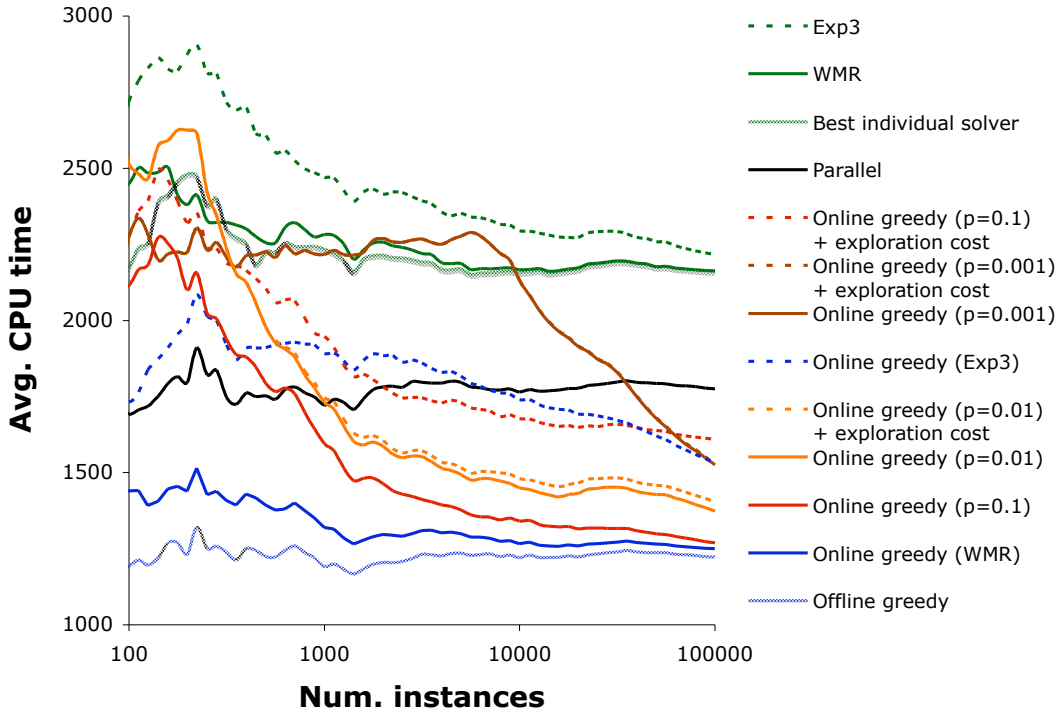

Figure 3.7: Performance of various online algorithms on instances drawn at random from the set of SAT 2007 benchmarks instances in the *random* category.

Figure 3.7 shows the (running) average CPU time for various online algorithms and offline schedules as a function of the number of instances encountered. The algorithms labeled "Online greedy ($p = \gamma$)" refer to the online greedy algorithm, run with exploration probability γ. For each of the three values of γ we display two curves: one for the average CPU time on the non-exploration rounds, and the other (marked "Online greedy ($p = \gamma$) + exploration cost") the overall average CPU time, including the time required to run each solver on the exploration rounds.

As Figure 3.7 shows, all the online greedy algorithms eventually outperform both the best individual solver and the parallel schedule in terms of average CPU time. In the full-information setting, this happens after a few dozen instances, while in the limited feedback settings it takes longer. In the limited feedback settings, each of the online greedy algorithms outperforms the best individual solver after less than 1000 instances, while the number of instances required to overtake the parallel schedule ranges from about 700 (when the exploration probability is 0.1, and exploration costs are ignored) to 35,000 (when the exploration probability is 0.001).

Not surprisingly, the online algorithms that select a single solver to run on each instance do not outperform the best individual solver, although they approach its performance as the number of instances grows large.

3.9.7 Experiments with instance features

In §3.8, we discussed how instance-specific features may be exploited to make a better choice of schedule to use in solving a particular instance. In this section, we present experiments that demonstrate the additional speedups that can be obtained using this approach.

Recall from §3.8 that in this approach we create, for each feature, a separate copy of our online schedule-selection algorithm that is only run on instances that have that feature. We then use a "sleeping experts algorithm" to select among the schedules returned by the various copies. We use $\mathbf{OG}^{se}$ to denote the online algorithm that results from composing the sleeping experts algorithm of Blum and Mansour [12] with the online greedy algorithm $\mathbf{OG}$ in this way. In other words, $\mathbf{OG}^{se}$ is the algorithm $\mathbf{SE}$ from §3.8, where the algorithm of Blum and Mansour [12] is the subroutine sleeping experts algorithm and $\mathbf{OG}$ is the subroutine online schedule selection algorithm.

Features used

Selecting a set of informative features for each of the eight problem domains considered in this chapter would be a challenging research project in itself. For this reason, take a very simple and domain-independent approach to feature selection. We made use of two types of features:

1. *Features based on competition benchmark directory structure.* We compute a number of features of each instance x based on the directory in which x is stored. Specifically, for each directory we create a Boolean feature that is true if and only if the

instance resides somewhere within that directory's subtree (so if x is stored d levels deep there will be d features that evaluate to true for x). Note that these features are potentially quite useful; for example the instances stored in the directory `GRAPHS/WMAXCUT/SPINGLASS` seem likely to have common features which some heuristic might be able to exploit. In an attempt to make our experiments fair, we manually removed any directory names that we felt would give away too much information (e.g., we would remove a directory called `HARD_INSTANCES`).

2. *Features based on instance annotations.* In some cases, the instances contained specific annotations that described the problem. Specifically, the instances used in the CASC-J3 theorem proving competition specified the field of mathematics that the theorem came from (e.g., general algebra, geometry, theory of computation).

Additionally, we include a Boolean feature that evaluates to true for all instances. This ensures that the online algorithm $\mathbf{OG}^{se}$ maintains regret bounds that hold for all instances, in addition to its per-feature regret bounds.

Note that in general there will be many Boolean features that are true for a particular instance, and so the online algorithm must discover which features to pay attention to.

Cross-validation

In addition to evaluating the online algorithm $\mathbf{OG}^{se}$, we perform experiments in which we evaluate the use of features under leave-one-out cross-validation. Unlike in our previous experiments, it is not immediately obvious how to perform leave-one-out cross-validation in the presence of features. One possible approach would be the following: when leaving out each instance x, run a copy of $\mathbf{OG}^{se}$ on the instances one at a time, with instance x presented last. Unfortunately, the computation time required by this approach is prohibitive for some solver competitions (some of which have thousands of instances within a single category). Instead, we adopt a simpler approach that is designed to achieve roughly the same effect.

Our approach is as follows. For each instance x, we remove x from the set of instances and use the remaining instances as training data. For each feature j, we use the subset of training instances that have feature j as input to the offline greedy approximation algorithm, producing a schedule S_j. Then, for each feature j, we create a "sleeping expert" that recommends schedule S_j on instances that have feature j. We then run the sleeping experts algorithm of Blum and Mansour [12] on all n instances, with x presented last, to obtain a schedule to use in solving x. Note that in the degenerate case where we have only

a single feature and it is true for all instances, this cross-validation procedure is identical to the one used in previous sections.

Results

Table 3.7 (in the previous section) summarizes the results of our experiments for the industrial, random, and hand-crafted categories of the SAT 2007 competition. The rows labeled *Online greedy w/features* refer to the online algorithm $\mathbf{OG}^{se}$. The rows labeled *Offline greedy w/features (CV)* refer to the cross-validation procedure just described. As the table shows, the use of features consistently improves performance, both in the online setting and under leave-one-out cross-validation. In the *random* category, for example, using features improves (the lower bound on) the average CPU time of the offline greedy algorithm (evaluated under leave-one-out cross-validation) by a factor of 1.43, and improves the average CPU time of the online greedy algorithm by a factor of 1.25.

Table 3.8: Average CPU time (lower bounds) required by different schedules and heuristics to solve instances from the *random* category of the 2007 SAT competition. Bold numbers indicate the (strictly) smallest value in a row.

Feature (#Instances)	**Best heuristic for feature**	**Greedy (CV)**	**Greedy (CV) w/features**
`2+p` (105)	1100 (*March KS*)	918	**885**
`2+p/p0.7` (36)	1607 (*SATzilla*)	**1286**	1368
`2+p/p0.8` (36)	847 (*March KS*)	850	**728**
`2+p/p0.9` (33)	678 (*March KS*)	590	**531**
`LargeSize` (130)	2276 (*adaptg2wsat+*)	2641	**1571**
`LargeSize/3SAT` (42)	1016 (*gnovelty+*)	2588	1016
`LargeSize/5SAT` (57)	**921** (*adaptg2wsat+*)	2215	1164
`LargeSize/7SAT` (31)	**2532** (*ranov*)	3496	3069
`OnThreshold` (176)	764 (*SATzilla*)	625	**435**
`OnThreshold/3SAT` (55)	404 (*SATzilla*)	606	**397**
`OnThreshold/5SAT` (60)	601 (*SATzilla*)	624	**523**
`OnThreshold/7SAT` (61)	1034 (*March KS*)	643	**382**

Table 3.8 illustrates in greater detail the power of using features on instances from the *random* category. The first column lists the feature along with the number of instances for

which the feature was true. The second column lists, for each feature, the minimum average CPU time required by any single heuristic, where the average is computed only over instances that have that feature. The third and fourth columns list the average CPU time for the greedy schedule (evaluated under cross-validation), with and without the benefit of features, respectively. Bold numbers indicate the minimum average CPU time within a row. As the table shows, the use of features substantially improves the performance of the greedy schedule in many cases. In nine of the twelve cases, the greedy schedule with features outperforms the best solver for instances that had that feature. In contrast, for the greedy schedule constructed without the use of features, this is only true in five out of twelve cases.

3.9.8 Summary of experimental evaluation

In this section we summarize our experimental results for the eight solver competitions described in §3.9.1. For each category of each competition, we compare the performance of the offline greedy schedule (evaluated under leave-one-out cross-validation) to that of the solver that performed best in terms of average CPU time. We quantify the performance improvement achieved by the greedy schedule by calculating two "speedup factors": the first equals the ratio of (a lower bound on) the average CPU time of the best individual solver to that of the schedule produced by the greedy algorithm, while the second equals the ratio of the median CPU time of the best individual solver to that of the greedy schedule (both speedup factors compare the greedy schedule to the same individual solver, namely the one that performed best in terms of average CPU time). We run the greedy algorithm both with and without the use of features, as described in §3.9.7. All CPU times for the offline greedy algorithm are calculated using leave-one-out cross-validation, to avoid results that are misleading due to overfitting.

Table 3.9 shows the results. In 30 out of 44 cases, the greedy schedule (evaluated under cross-validation) outperforms the best individual solver in terms of average CPU time, while in 14 cases, the greedy schedule performs worse than the best individual solver, due to overfitting. Generally speaking, overfitting occurs for categories in which the number of instances is relatively small. In terms of average CPU time, the performance improvements are less than a factor of 10 in all but one case. In terms of median CPU time, the performance improvements are more dramatic: the greedy schedule outperforms the best individual solver by more than a factor of 10 in several cases. This difference is not all that surprising, given that the "best" individual solver was defined as the one with minimum average CPU time, and not the one with minimum median CPU time.

The use of features usually but not always improved performance. In 30 of the 44 cases

listed in Table 3.9, the use of features led to a larger speedup in average CPU time; in 10 cases it was harmful; and in 4 cases it had no effect. In cases where the use of features is harmful, the harm is again due to overfitting. Again, overfitting occurs primarily in cases where the number of instances is relatively small. Again, note that the performance improvement in terms of median CPU time is generally larger than the performance in terms of average CPU time, and is larger than a factor of 10 in several cases.

Table 3.9: Speedup factors for various solver competitions.

Competition	**Category** (#Instances)	**Speedup w/features**		**Speedup**	
		Mean	*Median*	Mean	*Median*
CASC-21	CNF (191)	1.24	*0.97*	1.45	*1.46*
	EPR (98)	0.58	*1.00*	0.56	*0.98*
	FNT (100)	3.78	*2.94*	3.47	*2.94*
	FOF (295)	2.06	*90.0*	2.15	*90.0*
	SAT (100)	4.83	*0.98*	5.49	*0.98*
	UEQ (93)	0.99	*1.00*	0.99	*1.00*
CPAI'06	Binary ext. (1140)	1.39	*1.06*	1.37	*1.00*
	Binary int. (698)	3.03	*2.36*	1.97	*1.66*
	Global (127)	0.28	*1.00*	0.28	*0.94*
	Opt. binary ext. (619)	2.06	*1.36*	1.57	*1.11*
	Opt. n-ary ext. (97)	1.55	*1.64*	1.23	*0.94*
	N-ary ext. (312)	1.36	*18.6*	1.19	*18.6*
	N-ary int. (736)	2.60	*41.8*	2.10	*29.5*
IPC-5	Optimal planning (110)	1.78	*2.89*	1.61	*2.50*
MaxSAT-2007	Max-SAT (790)	0.99	*0.97*	0.98	*1.00*
	Partial Max-SAT (647)	1.68	*0.89*	1.31	*0.94*
	Weighted Max-SAT (308)	1.15	*1.55*	0.82	*1.76*
	Weighted Partial (702)	1.49	*1.58*	1.15	*0.81*
PB'07	Opt. big ints. (124)	1.11	*1.04*	1.05	*0.95*
	Opt. small ints. (396)	3.09	*6.77*	2.71	*4.08*
	Opt. small ints. non-lin. (280)	2.32	*1.01*	2.10	*0.96*
	Pure satisfiability (88)	1.24	*1.09*	0.98	*0.97*
	Small ints. (216)	3.19	*69.2*	2.56	*36.7*
QBFEVAL'07	Formal verification (728)	1.91	*3.04*	1.52	*2.36*
	Horn clause formulas (287)	1.06	*1.00*	1.06	*1.00*

continued on next page...

Table 3.9 (continued from previous page)

Competition	**Category** (#Instances)	**Speedup w/features**		**Speedup**	
		Mean	*Median*	Mean	*Median*
	Miscellanea (67)	2.19	*3.72*	2.19	*3.72*
	Non_prenex_non_cnf (81)	0.81	*0.76*	0.81	*0.79*
	Planning (80)	1.37	*0.92*	1.28	*1.00*
SAT 2007	And-Inverter Graphs (263)	1.26	*1.00*	1.11	*1.00*
	Hand-crafted (129)	1.49	*3.24*	1.37	*3.24*
	Industrial (166)	1.30	*1.42*	0.99	*1.18*
	Random (411)	2.24	*7.27*	1.61	*5.31*
SMT-COMP'07	AUFLIA (192)	2.62	*1.00*	2.64	*0.99*
	AUFLIRA (193)	15.1	*1.00*	15.1	*1.00*
	QF_AUFBV (187)	1.00	*1.00*	1.00	*1.00*
	QF_AUFLIA (206)	1.32	*1.00*	1.05	*1.00*
	QF_BV (200)	1.94	*1.00*	1.97	*1.00*
	QF_IDL (186)	1.01	*1.00*	1.00	*0.98*
	QF_LIA (186)	0.33	*10.0*	0.95	*10.0*
	QF_LRA (202)	0.92	*1.00*	0.82	*1.00*
	QF_RDL (168)	0.90	*1.00*	0.70	*1.00*
	QF_UF (199)	2.17	*1.98*	2.29	*1.98*
	QF_UFIDL (201)	0.85	*0.98*	0.85	*0.98*
	QF_UFLIA (110)	0.25	*1.00*	0.25	*1.00*

3.9.9 Experiments with restart schedules

In our experiments so far, we have only considered deterministic heuristics. In this section we consider randomized heuristics. Specifically, we consider the problem of constructing a single restart schedule to use to solve a set of problem instances via a single Las Vegas algorithm.

Following Gagliolo & Schmidhuber [31], we evaluate our algorithm for constructing restart schedules using the SAT solver `satz-rand`. We note that `satz-rand` is at this point a relatively old SAT solver. However it has the following key feature: successive runs of `satz-rand` on the same problem instance are *independent*, as required by our

theoretical results. More modern solvers (e.g., `MiniSat`) also make use of restarts but maintain a repository of conflict clauses that is shared among successive runs on the same instance, violating this independence assumption.

To generate a set of benchmark formulae, we use the instance generator supplied with *blackbox* [46] to generate 80 random logistics planning problems, using the same parameters that were used to generate the instance `logistics.d` from the paper by Gomes *et al.* [35].[9] We then used `SATPLAN` to find an optimal plan for each instance, and saved the Boolean formulae it generated. This yielded a total of 242 Boolean formulae.[10] We then performed $B = 1000$ runs of `satz-rand` on each formula, where the i^{th} run was performed with a time limit of $\frac{B}{i}$ as per the discussion in §3.5.2.

We evaluated several different restart schedules:

1. the schedule returned by the offline greedy approximation algorithm,
2. uniform schedules of the form $\langle t, t, t, \ldots \rangle$ for each $t \in \{1, 2, \ldots, B\}$,
3. geometric restart schedules of the form $\langle \beta^0, \beta^1, \beta^2, \ldots \rangle$ for each $\beta \in \{1.1^k : 1 \leq k \leq \lceil \log_{1.1} B \rceil\}$, and
4. Luby's universal restart schedule.

We estimated the expected CPU time required by each schedule using the "refined estimation procedure" described in §3.5.2.

Table 3.10 gives the average CPU time required by each of the schedules we evaluated. Because run lengths were capped at 1000 seconds, the values in this table are lower bounds on the (estimated) expected running time of a schedule on problem instances drawn from the distribution used in these experiments. In terms of these lower bounds, the schedule returned by the greedy approximation algorithm had the smallest mean running time. The greedy schedule was 1.7 times faster than Luby's universal schedule, 1.5 times faster than the best uniform schedule (which used threshold $t = 85$), and 1.1 times faster than the best geometric schedule (which set $\beta \approx 1.6$). The average CPU time for a schedule that performed no restarts was about 3.4 times that of the greedy schedule in terms of the lower bounds on average CPU time, but is likely to be much worse in terms of actual expected running time (it is likely that some runs would take *much* longer than 1000 seconds if all the runs were allowed to finish).

[9] The parameters are: 9 packages, 5 cities, 2 planes, 3 locations per city, 1 truck per city, and 9 goals.

[10] The number of generated formulae is less than the sum of the minimum plan lengths, because `SATPLAN` can trivially reject some plan lengths without invoking a SAT solver.

Table 3.10: Performance of various restart schedules for running `satz-rand` on a set of Boolean formulae derived from random logistics planning benchmarks.

Restart schedule	**Avg. CPU (s)**
Greedy schedule	21.9
Greedy schedule (CV)	22.8
Best geometric schedule	23.9
Best uniform schedule	33.9
Luby's universal schedule	37.2
No restarts	74.1

Examining Table 3.10, one may be concerned that the greedy approximation algorithm was run using the same estimated run length distributions that were later used to estimate its expected CPU time. To address the possibility of overfitting, we also evaluated the greedy algorithm using leave-one-out cross-validation. The estimated average CPU time increased by about 4% under leave-one-out cross-validation.

3.9.10 Ways to improve our experimental results

The results of the experiments presented in this chapter could potentially be improved in at least two ways:

1. *Sharing information among heuristics.* In the experiments performed in this chapter, each heuristic executes independently. In practice, during the process of solving an instance, one heuristic may discover information that could be useful to share with other heuristics. For example, when solving optimization problems, the heuristics could share upper and lower bounds on the optimal objective function value. When solving decision problems such as Boolean satisfiability or constraint satisfaction, the runs could maintain a common repository of learned conflict clauses.

2. *Monitoring progress of heuristics.* The schedules considered in this chapter simply run a heuristic for a certain amount of time, without monitoring the heuristic to see whether it appears to be close to producing an answer. In practice, it may be possible to predict a heuristic's remaining running time based on its current state. For example, if the heuristic makes use of chronological backtracking one could examine how much of the search tree has already been pruned. One could also leverage

existing techniques for deliberation control (e.g., [58, 72]). Exploiting information of this sort is an interesting prospect, both from an experimental and a theoretical point of view.

3.10 Conclusions

This chapter presented algorithms for combining multiple heuristics in offline and online settings. Experimentally, we used data from recent solver competitions to show that, by combining heuristics, we can improve the performance of state-of-the-art solvers in several problem domains. Our experimental evaluation considered heuristics for optimization as well as decision problems, as well as a randomized heuristic, and showed that instance-specific features can be exploited to obtain additional performance improvements.

Chapter 4

Using Decision Procedures Efficiently for Optimization

4.1 Introduction

Optimization problems are often solved by making repeated calls to a decision procedure that answers questions of the form "Does there exist a solution with cost at most k?". Each query to the decision procedure can be represented as a pair $\langle k, t \rangle$, where t is a bound on the CPU time the decision procedure may consume in answering the question. The result of a query is either a (provably correct) "yes" or "no" answer or a timeout. A *query strategy* is a rule for determining the next query $\langle k, t \rangle$ as a function of the responses to previous queries.

The performance of a query strategy can be measured in several ways. Given a fixed query strategy and a fixed minimization problem, let $u(T)$ denote the upper bound (i.e., the smallest k that elicited a "yes" response) obtained by running the query strategy for a total of T time units; and let $l(T)$ be the corresponding lower bound. A natural goal is for $u(T)$ to decrease as quickly as possible. Alternatively, we might want to achieve $u(T) \leq \alpha \cdot l(T)$ in the minimum possible time for some desired approximation ratio $\alpha \geq 1$.

In this chapter we study the problem of designing query strategies. Our goal is to devise strategies that do well with respect to natural performance criteria such as the ones just described, when applied to decision procedures whose behavior (i.e., how the required CPU time varies as a function of k) is typical of the procedures used in practice.

The results in this chapter are based on a conference paper [83].

4.1.1 Motivations

A.I. planning is the problem of finding a sequence of actions (called a plan) that leads from a starting state to a desired goal state, according to some formally-specified model of how actions affect the state of the world. The makespan of a plan is the number of steps in the plan, treating actions that can be performed simultaneously as a single step. In *optimal planning*, the goal is to find a plan with (provably) minimum makespan.

The two winners from the optimal track of last year's International Planning Competition were `SATPLAN` [47] and `Maxplan` [89]. Both planners find a minimum-makespan plan by making a series of calls to a SAT solver, where each call determines whether there exists a feasible plan of makespan $\leq k$ (where the value of k varies across calls). One of the differences between the two planners is that `SATPLAN` uses the *ramp-up* query strategy (in which the i^{th} query is $\langle i, \infty \rangle$), whereas `Maxplan` uses the *ramp-down* strategy (in which the i^{th} query is $\langle U - i, \infty \rangle$, where U is an upper bound obtained using heuristics).

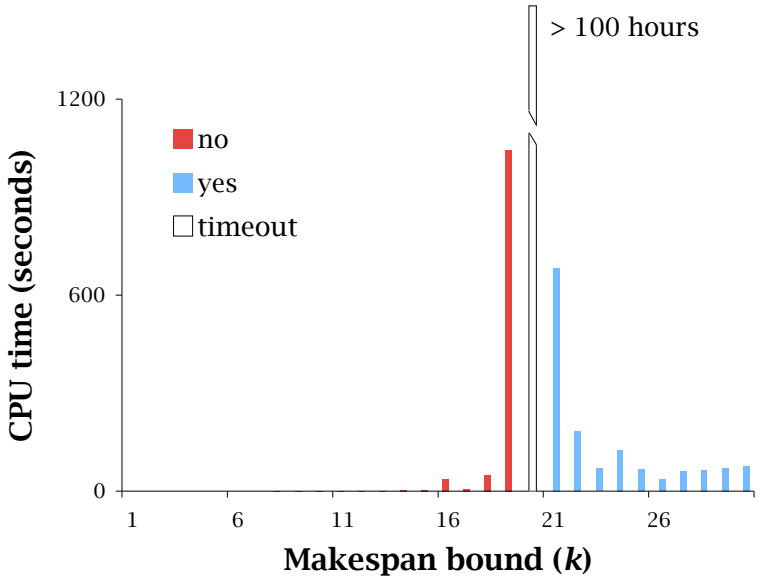

Figure 4.1: Behavior of the SAT solver `siege` running on formulae generated by `SATPLAN` to solve instance `p17` from the `pathways` domain of the 2006 International Planning Competition.

To appreciate the importance of query strategies, consider Figure 4.1, which shows the CPU time required by `siege` (the SAT solver used by `SATPLAN`) as a function of the makespan bound k, on a benchmark instance from the competition. For most values of

k, the solver terminates in under one minute; for $k = 19$ and $k = 21$, the solver requires 10-20 minutes; and for $k = 20$, the solver was run for over 100 hours without returning an answer. Because only the queries with $k \geq 21$ return a "yes" answer, the ramp-up query strategy (used by SATPLAN) does not find a feasible plan after running for 100 hours, while the ramp-down strategy returns a feasible plan but does not yield any non-trivial lower bounds on the optimum makespan. In this example, the time required by any query strategy to obtain a *provably optimal* plan is dominated by the time required to run the decision procedure with input $k = 20$. On the other hand, executing the queries $\langle 18, \infty \rangle$ and $\langle 23, \infty \rangle$ takes less than two minutes and yields a plan whose makespan is provably at most $\frac{23}{18+1} \approx 1.21$ times optimal. Thus, the choice of query strategy has a dramatic effect on the time required to obtain a *provably approximately optimal* solution. For planning problems where provably optimal plans are currently out of reach, obtaining provably approximately optimal plans quickly is a natural goal.

4.1.2 Summary of results

In this chapter we consider the problem of devising query strategies in two settings. In the *single-instance* setting, we are confronted with a single optimization problem, and wish to obtain an (approximately) optimal solution as quickly as possible. In this setting we provide a simple query strategy S_2, and analyze its performance in terms of a parameter that is intended to capture the unpredictability of the decision procedure's behavior. We then show that our performance guarantee is optimal up to a constant factor.

In the *multiple-instance* setting, we use the same decision procedure to solve a number of optimization problems, and our goal is to learn from experience in order to improve performance. In this setting, we prove that computing an optimal query strategy is NP-hard, and discuss how algorithms from machine learning theory can be used to learn a good query strategy on-the-fly while solving a sequence of optimization problems.

In the experimental section of this chapter, we demonstrate that query strategy S_2 can be used to create improved versions of state-of-the-art algorithms for planning and job shop scheduling. In the course of the latter experiments we develop a simple method for applying query strategies to branch and bound algorithms, which seems likely to be useful in other domains besides job shop scheduling.

4.1.3 Related work

The ramp-up strategy was used in the original GraphPlan algorithm [13] for A.I. planning, and is conceptually similar to iterative deepening [52].

In the A.I. planning community, alternatives to the ramp-up strategy were investigated by Rintanen [69], who proposed two algorithms. Algorithm A runs the decision procedure on the first n decision problems in parallel, each at equal strength, where n is a parameter. Algorithm B runs the decision procedure on all decision problems simultaneously, with the i^{th} problem receiving a fraction of the CPU time proportional to γ^i, where $\gamma \in (0, 1)$ is a parameter. Rintanen showed that Algorithm B yields dramatic performance improvements over the ramp-up strategy on a variety of A.I. planning benchmarks.

Our query strategy S_2 exploits binary search and is quite different from the three strategies just discussed. In the experimental section of this chapter, we compare S_2 to the ramp-up strategy and to a geometric strategy based on a Rintanen's Algorithm B.

4.2 Preliminaries

In this chapter we are interested in solving minimization problems of the form

$$OPT = \min_{x \in \mathcal{X}} c(x)$$

where $\mathcal{X}$ is an arbitrary set and $c : \mathcal{X} \to \mathbb{Z}_+$ is a function assigning a positive integer cost to each $x \in \mathcal{X}$. We will solve such a minimization problem by making a series of calls to a decision procedure that, given as input an integer k, determines whether there exists an $x \in \mathcal{X}$ with $c(x) \leq k$. When given input k, the decision procedure runs for $\tau(k)$ time units before returning a (provably correct) "yes" or "no" answer. Thus from our point of view, a minimization problem is completely specified by the integer OPT and the function τ.

Definition (instance). *An* instance *of a minimization problem is a pair $\langle OPT, \tau \rangle$, where OPT is the smallest input for which the decision procedure answers "yes" and $\tau(k)$ is the CPU time required by the decision procedure when it is run with input k.*

A *query* is a pair $\langle k, t \rangle$. To execute this query, one runs the decision procedure with input k subject to a time limit t. Executing query $q = \langle k, t \rangle$ on instance $I = \langle OPT, \tau \rangle$ requires CPU time $\min\{t, \tau(k)\}$ and elicits the response

$$\text{response}(I, q) = \begin{cases} \text{yes} & \text{if } t \geq \tau(k) \text{ and } k \geq OPT \\ \text{no} & \text{if } t \geq \tau(k) \text{ and } k < OPT \\ \text{timeout} & \text{if } t < \tau(k) \text{ .} \end{cases}$$

We say that a query q *eliminates* an integer k if executing q determines what side of OPT that k is on.

Definition (elimination). *A query $q = \langle k_0, t \rangle$ eliminates a value k if the response to q is "yes" and $k \geq k_0$, or if the response is "no" and $k \leq k_0$.*

Definition (query strategy). *A* query strategy S *is a function that takes as input the sequence $\langle r_1, r_2, \ldots, r_i \rangle$ of responses to the first i queries, and returns as output a new query $\langle k, t \rangle$.*

When executing queries according to some query strategy, we maintain upper and lower bounds on OPT. Initially $l = 1$ and $u = \infty$. If query $\langle k, t \rangle$ elicits a "no" response we set $l \leftarrow k + 1$; and if it elicits a "yes" reponse we set $u \leftarrow k$. Thus, any $k \notin [l, u - 1]$ has been eliminated by the query strategy.

4.2.1 Performance of query strategies

In the single-instance setting, we will evaluate a query strategy according to the following competitive ratio.

Definition (competitive ratio). *The* competitive ratio *of a query strategy S on an instance $I = \langle OPT, \tau \rangle$ is defined by*

$$\text{ratio}(S, I) = \max_k \left\{ \frac{T_{elim}(S, k)}{\tau(k)} \right\} .$$

where $T_{elim}(S, k)$ is CPU time required to eliminate k when executing queries according to strategy S.

As an example, consider running the ramp-up query strategy on the instance $I = \langle OPT, \tau \rangle$, where $\tau(k) = 2^{k-1}$ for all k. The ramp-up strategy must be run for CPU time $1 + 2 + 4 + 8 + \ldots + 2^{OPT-1} = 2^{OPT} - 1$ in order to eliminate the value OPT, and OPT is the last k value to be eliminated. The the ramp-up strategy has competitive ratio $\frac{2^{OPT}-1}{2^{OPT-1}} < 2$ on the instance I.

4.2.2 Behavior of τ

The performance of our query strategies will depend on the behavior of the function τ. For most decision procedures used in practice, we expect $\tau(k)$ to be an increasing function

for $k \leq OPT$ and a decreasing function for $k \geq OPT$. Previous work [75, 85] has shown that this behavior is prevalent in planning domains (e.g., see the behavior of `siege` illustrated in Figure 4.1), and our query strategies are designed to take advantage of it. More specifically, our query strategies are designed to work well when τ is close to its *hull*.

Definition (hull). *The* hull *of* τ *is the function*

$$\text{hull}^{\tau}(k) = \min \left\{ \max_{k_0 \leq k} \tau(k_0), \max_{k_1 \geq k} \tau(k_1) \right\} .$$

Figure 4.2 gives an example of a function τ (gray bars) and its hull (dots). Note that the region under the curve $\text{hull}^{\tau}(k)$ is *not* (in general) the convex hull of the points $(k, \tau(k))$. Also note that the functions τ and hull^{τ} are identical if τ is monotonically increasing (or monotonically decreasing), or if there exists an x such that τ is monotonically increasing for $k \leq x$ and monotonically decreasing for $k > x$.

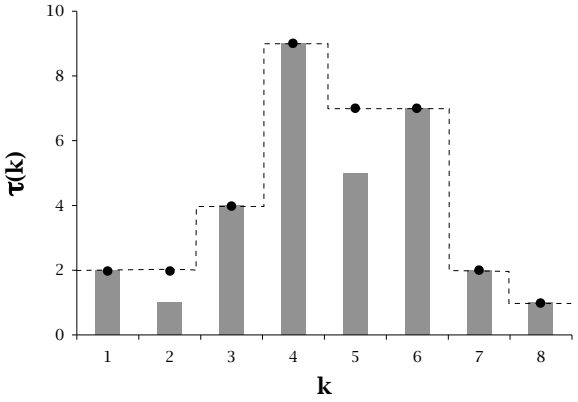

Figure 4.2: A function τ (gray bars) and its hull (dots).

We measure the discrepancy between τ and its hull in terms of the *stretch* of an instance.

Definition (stretch). *The* stretch *of an instance* $I = \langle OPT, \tau \rangle$ *is defined by*

$$\text{stretch}(I) = \max_{k} \frac{\text{hull}^{\tau}(k)}{\tau(k)} .$$

The instance depicted in Figure 4.2 has a stretch of 2 because $\tau(2) = 1$ while $\text{hull}^{\tau}(2) = 2$.

4.3 The Single-Instance Setting

We first consider the case in which we wish to design a query strategy for use in solving a single instance $I = \langle OPT, \tau \rangle$ (where OPT and τ are of course unknown to us). Our goal is to devise a query strategy that minimizes the value of $\mathrm{ratio}(S, I)$ for the worst case instance I. We assume $OPT \in \{1, 2, \ldots, U\}$ for some known upper bound U. For simplicity, we also assume $\tau(k) \geq 1$ for all k.

4.3.1 Arbitrary instances

In the case where the function τ is arbitrary, the following simple query strategy S_1 achieves a competitive ratio that is optimal (to within constant factors). S_1 and its analysis are similar to those of Algorithm A of Rintanen [69]. We do not advocate the use of S_1. Rather, its analysis indicates the limits imposed by making no assumptions about τ.

Query strategy S_1

1. Initialize $T \leftarrow 1$, $l \leftarrow 1$, and $u \leftarrow U$.
2. While $l < u$:
   (a) For each $k \in \{l, l+1, \ldots, u-1\}$, execute the query $\langle k, T \rangle$, and update l and u appropriately (if the response is "yes" then set $u \leftarrow k$, and if the response is "no" then set $l \leftarrow k+1$).
   (b) Set $T \leftarrow 2T$.

The analysis of S_1 is straightforward. Consider some fixed k, and let $T_k = 2^{\lceil \log_2 \tau(k) \rceil}$ be the smallest power of two that is $\geq \tau(k)$. Each iteration of the loop consumes time at most TU, and on the iteration where $T = T_k$, k will be eliminated. Thus the total time it takes to eliminate k is at most

$$U + 2U + 4U + \ldots + T_k U < 2T_k U < 4\tau(k)U \ .$$

Because k we arbitrary, it follows that $\mathrm{ratio}(S_1, I) \leq 4U$.

To obtain a matching lower bound, suppose that $\tau(k) = 1$ if $k = k^*$, and $\tau(k) = \infty$ otherwise. For any query strategy S, there is some choice of k^* that forces S to consume time at least U before executing a successful query[1], which implies $\mathrm{ratio}(S, I) \geq U$.

[1]We are only considering deterministic query strategies. For randomized query strategies, there must be some choice of k^* that forces S to consume *expected* time at least $\frac{U}{2}$ before executing a successful query.

These observations are summarized in the following theorem.

Theorem 25. *For any instance I, $\mathrm{ratio}(S_1, I) = O(U)$. Furthermore, for any strategy S, there exists an instance I such that $\mathrm{ratio}(S, I) = \Omega(U)$.*

4.3.2 Instances with low stretch

In practice we do not expect τ to be as pathological as the function used to prove the lower bound in Theorem 25. Indeed, as already mentioned, in practice we expect instances to have low stretch, whereas the instance used to prove the lower bound has infinite stretch. We now describe a query strategy S_2 whose competitive ratio is $O(\mathrm{stretch}(I) \cdot \log U)$, a dramatic improvement over Theorem 25 for instances with low stretch.

Like S_1, strategy S_2 maintains an interval $[l, u]$ that is guaranteed to contain OPT, and maintains a value T that is periodically doubled. S_2 also maintains a "timeout interval" $[t_l, t_u]$ with the property that the queries $\langle t_l, T\rangle$ and $\langle t_u, T\rangle$ have both been executed and returned a timeout response.

Query strategy S_2

1. Initialize $T \leftarrow 2$, $l \leftarrow 1$, $u \leftarrow U$, $t_l \leftarrow \infty$, and $t_u \leftarrow -\infty$.
2. While $l < u$:
   (a) If $t_l \neq \infty$ and $[l, u-1] \subseteq [t_l, t_u]$ then set $T \leftarrow 2T$, set $t_l \leftarrow \infty$, and set $t_u \leftarrow -\infty$.
   (b) Let $u' = u - 1$. Define

$$k = \begin{cases} \left\lfloor \frac{l+u'}{2} \right\rfloor & \text{if } [l, u'] \text{ and } [t_l, t_u] \text{ are disjoint or } t_l = \infty \\ \left\lfloor \frac{l+t_l-1}{2} \right\rfloor & \text{if } [l, u'] \text{ and } [t_l, t_u] \text{ intersect and } t_l - l > u' - t_u \\ \left\lfloor \frac{t_u+1+u'}{2} \right\rfloor & \text{otherwise.} \end{cases}$$

   (c) Execute the query $\langle k, T\rangle$. If the result is "yes" set $u \leftarrow k$; if the result is "no" set $l \leftarrow k + 1$; and if the result is "timeout" set $t_l \leftarrow \min\{t_l, k\}$ and set $t_u \leftarrow \max\{t_u, k\}$.

Each query executed by S_2 is of the form $\langle k, T\rangle$, where $k \in [l, u-1]$ but $k \notin [t_l, t_u]$. We say that such a k value is *eligible*. The queries are selected in such a way that the number

of eligible k values decreases exponentially. This is accomplished using what could be described as a "two-sided" binary search. Once there are no more eligible k values, T is doubled and $[t_l, t_u]$ is reset to the empty interval (so each $k \in [l, u-1]$ becomes eligible again).

To analyze S_2, we first bound the number of queries that can be executed in between updates to T. As already mentioned, the k value defined in step 2(b) belongs to the interval $[l, u-1]$ but not to the interval $[t_l, t_u]$. By examining each case, we find that the number of k values that have this property goes down by a factor of at least $\frac{1}{4}$ every query, except for the very first query that causes a timeout. It follows that the number of queries in between updates to T is $O(\log U)$.

To complete the analysis, first note that whenever $t_l \neq \infty$ and $t_u \neq -\infty$, it holds that $\tau(t_l) > T$ and $\tau(t_u) > T$. For any $k \in [t_l, t_u]$, this implies $\mathrm{hull}^\tau(k) > T$ (by definition of hull) and thus $\tau(k) > \frac{T}{\mathrm{stretch}(I)}$ (by definition of stretch). Now consider some arbitrary k. Once $T \geq \mathrm{stretch}(I) \cdot \tau(k)$ it cannot be that $k \in [t_l, t_u]$, so we must have $k \notin [l, u-1]$ before T can be doubled again. Because there can be at most $O(\log U)$ queries in between updates to T, it follows that we have to wait $O(\mathrm{stretch}(I) \cdot \tau(k) \cdot \log U)$ time before $k \notin [l, u-1]$. Because this holds for all k, it follows that $\mathrm{ratio}(S_1, I) = O(\mathrm{stretch}(I) \cdot \log U)$.

We now use a simple information-theoretic argument to prove a matching lower bound. Fix some query strategy S. Let $\tau(k) = 1$ for all k (clearly, $\mathrm{stretch}(I) = 1$). Assume without loss of generality that S only executes queries of the form $\langle k, 1\rangle$. For each $OPT \in \{1, 2, \ldots, U\}$, S must elicit a unique sequence of "yes" or "no" answers, one of which must have length $\geq \lfloor \log_2 U \rfloor$. Thus for some choice of OPT, $\mathrm{ratio}(S, I) \geq \frac{\lfloor \log_2 U \rfloor}{2} = \Omega(\mathrm{stretch}(I) \cdot \log U)$. Thus we have proved the following theorem.

Theorem 26. *For any instance I,* $\mathrm{ratio}(S_2, I) = O(\mathrm{stretch}(I) \cdot \log U)$. *For any strategy S, there exists an instance I such that* $\mathrm{ratio}(S, I) = \Omega(\mathrm{stretch}(I) \cdot \log U)$.

4.3.3 Generalizing S_2

Although the performance of query strategy S_2 (as summarized in Theorem 26) is optimal to within constant factors, in practice one might want to adjust the behavior of S_2 so as to obtain better performance on a particular set of optimization problems. Toward this end, we generalize S_2 by introducing three parameters: β controls the value of k; γ controls the rate at which the time limit T is increased; and ρ controls the balance between the time the strategy spends working to improve its lower bound versus the time it spends working to improve the upper bound. Each parameter takes on a value between 0 and 1. The parameters were chosen so as to include several natural query strategies in the parameter

space. The original strategy S_2 is recovered by setting $\beta = \gamma = \rho = \frac{1}{2}$. When $\beta = \gamma = 0$ and $\rho = 0$, S_3 is equivalent to the ramp-up query strategy (in which the i^{th} query is $\langle i, \infty \rangle$). When $\beta = \gamma = 0$ and $\rho = 1$, S_3 is equivalent to the ramp-down query strategy (in which the i^{th} query is $\langle U - i, \infty \rangle$).

The analysis of S_3 follows along exactly the same lines as that of S_2. Retracing the argument leading up to Theorem 26 and working out the appropriate constant factors yields the following theorem, which shows that the class $S_3(\beta, \gamma, \rho)$ includes a wide variety of query strategies with performance guarantees similar to that of S_2 (note that the theorem provides no guarantees when $\beta = 0$ or $\gamma = 0$, as in the ramp-up and ramp-down strategies).

Theorem 27. *Let $S = S_3(\beta, \gamma, \rho)$, where $0 < \beta \leq \frac{1}{2}$, $0 < \gamma < 1$, and $0 < \rho < 1$. Then for any instance I, $\mathrm{ratio}(S, I) = O(\frac{1}{\beta\gamma} \cdot \mathrm{stretch}(I) \cdot \log U)$.*

Query strategy $S_3(\beta, \gamma, \rho)$

1. Initialize $T \leftarrow \frac{1}{\gamma}$, $l \leftarrow 1$, $u \leftarrow U$, $t_l \leftarrow \infty$, and $t_u \leftarrow -\infty$.
2. While $l < u$:
   (a) If $t_l \neq \infty$ and $[l, u-1] \subseteq [t_l, t_u]$ then set $T \leftarrow \frac{T}{\gamma}$, set $t_l \leftarrow \infty$, and set $t_u \leftarrow -\infty$.
   (b) Let $u' = u - 1$. If $[l, u']$ and $[t_l, t_u]$ are disjoint (or $t_l = \infty$) then define

$$k = \begin{cases} \lfloor (1-\beta)l + \beta u' \rfloor & \text{if } (1-\rho)l > \rho(U - u') \\ \lfloor \beta l + (1-\beta)u' \rfloor & \text{otherwise;} \end{cases}$$

   else define

$$k = \begin{cases} \lfloor (1-\beta)l + \beta(t_l - 1) \rfloor & \text{if } (1-\rho)(t_l - l) \\ & > \rho(u' - t_u) \\ \lfloor (1-\beta)u' + \beta(t_u + 1) \rfloor & \text{otherwise.} \end{cases}$$

   (c) Execute the query $\langle k, T \rangle$. If the result is "yes" set $u \leftarrow k$; if the result is "no" set $l \leftarrow k + 1$; and if the result is "timeout" set $t_l \leftarrow \min\{t_l, k\}$ and set $t_u \leftarrow \max\{t_u, k\}$.

4.4 The Multiple-Instance Setting

We now consider the case in which the same decision procedure is used to solve a sequence $\langle x_1, x_2, \ldots, x_n \rangle$ of instances of some optimization problem. In this case, it is natural to attempt to learn something about the instance sequence and select query strategies accordingly.

Let $\mathcal{S}$ be some set of query strategies, and for any $S \in \mathcal{S}$, let $c_i(S)$ denote the CPU time required to obtain an acceptable solution to instance $x_i = \langle OPT_i, \tau_i \rangle$ using query strategy S (e.g., $c_i(S)$ could be the time required to obtain a solution whose cost is provably at most a factor α times optimal, for some $\alpha \geq 1$). We consider the problem of selecting query strategies in two settings: offline and online.

4.4.1 Computing an optimal query strategy offline

In the offline setting we are given as input the values of $\tau_i(k)$ for all i and k, and wish to compute the query strategy

$$S^* = \arg\min_{S \in \mathcal{S}} \sum_{i=1}^{n} c_i(S) \; .$$

This offline optimization problem arises in practice when the instances $\langle x_1, x_2, \ldots, x_n \rangle$ have been collected for use as training data, and we wish to compute the strategy S^* that performs optimally on the training data.

Unfortunately, if $\mathcal{S}$ contains all possible query strategies then computing S^* is NP-hard. To see this, suppose that our goal is to obtain an approximation ratio $\alpha = U - 1$. To obtain this ratio, we simply need to execute a single query that returns a non-timeout response. Consider the special case that $\tau_i(k) \in \{1, \infty\}$ for all i and k, and without loss of generality consider only query strategies that issue queries of the form $\langle k, 1 \rangle$. For our purposes, such a query strategy is just a permutation of the k values in the set $\{1, 2, \ldots, U\}$. For each k, let $A_k = \{x_i : \tau_i(k) = 1\}$. To find an optimal query strategy, we must order the sets $A_1, A_2, \ldots, A_U$ from left to right so as to minimize the sum, over all instances x_i, of the position of the leftmost set that contains x_i. This is exactly MIN-SUM SET COVER. For any $\epsilon > 0$, obtaining a $4 - \epsilon$ approximation to MIN-SUM SET COVER is NP-hard [26]. Thus we have the following theorem.

Theorem 28. *For any $\epsilon > 0$, obtaining a $4 - \epsilon$ approximation to the optimal query strategy is NP-hard.*

Certain special cases of the offline problem are tractable. For example, suppose all queries take the same time, say $\tau_i(k) = t$ for all i and k. In this case we need only consider queries of the form $\langle k, t\rangle$, and any such query elicits a non-timeout response. A query strategy can then be specified as a binary search tree over the key set $\{1, 2, \ldots, U\}$. The optimal query strategy is simply the optimum binary search tree for the access sequence $\langle OPT_1, OPT_2, \ldots, OPT_n\rangle$, which can be computed in $O(U^2)$ time using dynamic programming [51]. Similarly, if we consider arbitrary τ_i but restrict ourselves to queries of the form $\langle k, \infty\rangle$ (so that again all queries succeed), dynamic programming can be used to compute an optimal query strategy. Finally, the offline problem is tractable if $\mathcal{S}$ is small enough for us to search through it by brute force. Based on the results of the previous section, a natural choice would be for $\mathcal{S}$ to include $S_3(\beta, \gamma, \rho)$ for various values of the three parameters.

As discussed in Chapter 2, a simple greedy algorithm achieves the optimal approximation ratio of 4 for MIN-SUM SET COVER, and it is natural to wonder whether this greedy algorithm can be generalized to obtain a 4-approximation to the optimal query strategy. We are not aware of any straightforward way of accomplishing this. Note that in general, the optimal query strategy will adapt its choice of later queries based on the results of earlier queries, whereas the natural generalization of the greedy algorithm for MIN-SUM SET COVER would produce a static list of queries.

4.4.2 Selecting query strategies online

We now consider the problem of selecting query strategies in an online setting, assuming that $|\mathcal{S}|$ is small enough that we would not mind using $O(|\mathcal{S}|)$ time or space for decision-making. In the online setting we are fed, one at a time, a sequence $\langle x_1, x_2, \ldots, x_n\rangle$ of problem instances to solve. Prior to receiving instance x_i, we must select a query strategy $S_i \in \mathcal{S}$. We then use S_i to solve x_i and incur cost $c_i(S_i)$. Our *regret* at the end of n rounds is equal to

$$\frac{1}{n} \cdot \left(\mathbb{E}\left[\sum_{i=1}^{n} c_i(S_i) \right] - \min_{S \in \mathcal{S}} \sum_{i=1}^{n} c_i(S) \right) \tag{4.1}$$

where the expectation is over any random bits used by our strategy-selection algorithm. That is, regret is $\frac{1}{n}$ times the difference between the expected total cost incurred by our online algorithm and that of the optimal query strategy for the (unknown) set of n instances. An online algorithm's worst-case regret is the maximum value of (4.1) over all instance sequences of length n. A *no-regret algorithm* has worst-case regret that is $o(1)$ as a function of n.

We now describe how two existing algorithms can be applied to the problem of selecting query strategies. Let M be an upper bound on $c_i(S)$, and let T be an upper bound on $\tau_i(k)$. Viewing our online problem as an instance of the "nonstochastic multiarmed bandit problem" and using the **Exp3** algorithm of Auer *et al.* [5] yields regret $O\left(M\sqrt{\frac{1}{n}|\mathcal{S}|}\right) = o(1)$. The second algorithm makes use of the fact that on any particular instance x_i, we can obtain enough information to determine the value of $c_i(S)$ for all $S \in \mathcal{S}$ by executing the query $\langle k, T\rangle$ for each $k \in \{1, 2, \ldots, U\}$. This requires CPU time at most TU. We can then use the "label-efficient forecaster" of Cesa-Bianchi et al. [17] to select query strategies. Theorem 1 of that paper shows that the regret is at most $M\left(\ln\frac{|\mathcal{S}|}{\eta} + n\frac{\eta}{2\varepsilon}\right) + \varepsilon nTU$, where η and ε are parameters. Optimizing η and ε yields regret $O\left(M\left(\frac{TU\ln|\mathcal{S}|}{Mn}\right)^{\frac{1}{3}}\right) = o(1)$. Given n as input, one can choose whichever of the two algorithms yields the smaller regret bound.

4.5 Experimental Evaluation

In this section we evaluate query strategy S_2 experimentally by using it to create modified versions of state-of-the-art solvers in two domains: classical A.I. planning and job shop scheduling. In both of these domains, we found that the number of standard benchmark instances was too small for the online algorithms discussed in the previous section to be effective. Accordingly, our experimental evaluation focuses on the techniques developed for the single-instance setting.

4.5.1 Planning

The planners entered in the 2006 International Planning Competition were divided into two categories: *optimal* planners always return a plan of provably minimum makespan, whereas *satisficing* planners simply return a feasible plan quickly. In this section we pursue a different goal: obtaining a *provably* near-optimal plan as quickly as possible.

As already mentioned, `SATPLAN` finds a minimum-makespan plan by making a sequence of calls to a SAT solver that answers questions of the form "Does there exist a plan of makespan $\leq k$?". The original version of `SATPLAN` tries k values in an increasing sequence starting from $k = 1$, stopping as soon as it obtains a "yes" answer. We compare the original version to a modified version that instead uses query strategy S_2. When using S_2 we do not share any work (e.g., intermediate result files) among queries with the same

k value, although doing so could improve performance.

We ran each of these two versions of SATPLAN on benchmark instances from the 2006 International Planning Competition, with a one hour time limit per instance, and recorded the upper and lower bounds we obtained. To obtain an initial upper bound, we ran the satisficing planner SGPlan [37] with a one minute time limit. We chose SGPlan because it won first prize in the *satisficing planning* track of last year's competition. If SGPlan found a feasible plan within the one minute time limit, we used the number of actions in that plan as an upper bound on the optimum makespan; otherwise we artificially set the upper bound to 100.

Table 4.1: Performance of two query strategies on benchmark instances from the pathways domain of the 2006 International Planning Competition. Bold numbers indicate the (strictly) best upper/lower bound we obtained.

Instance	SATPLAN (S_2) [lower,upper]	SATPLAN (S_g) [lower,upper]	SATPLAN **(original)** [lower,upper]
p01	[5,5]	[5,5]	[5,5]
p02	[7,7]	[7,7]	[7,7]
p03	[8,8]	[8,8]	[8,8]
p04	[8,8]	[8,8]	[8,8]
p05	[9,9]	[9,9]	[9,9]
p06	[12,12]	[12,12]	[12,12]
p07	[13,13]	[13,13]	[13,13]
p08	[15,17]	[16,17]	[16,∞]
p09	[15,17]	[15,17]	[15,∞]
p10	[15,15]	[15,15]	[15,15]
p11	[16,17]	[16,17]	[16,∞]
p12	[16,19]	[17,19]	[17,∞]
p13	[16,18]	[17,18]	[17,∞]
p14	[14,20]	[15,**19**]	[15,∞]
p15	[18,18]	[18,18]	[18,18]
p16	[17,**21**]	[19,22]	[19,∞]
p17	[19,**21**]	[20,22]	[20,∞]
p18	[19,**22**]	[19,23]	[19,∞]
p19	[17,**22**]	[18,24]	[18,∞]

continued on next page...

Table 4.1 (continued from previous page)

Instance	SATPLAN (S_2) [lower,upper]	SATPLAN (S_g) [lower,upper]	SATPLAN **(original)** [lower,upper]
p20	[17,28]	[18,**27**]	[**19**,∞]
p21	[20,25]	[21,25]	[**22**,∞]
p22	[17,**23**]	[18,26]	[**19**,∞]
p23	[17,25]	[17,25]	[**18**,∞]
p24	[21,**27**]	[21,28]	[**22**,∞]
p25	[20,**27**]	[20,∞]	[**21**,∞]
p26	[19,**27**]	[20,31]	[**21**,∞]
p27	[19,34]	[20,**31**]	[20,∞]
p28	[19,**27**]	[20,∞]	[**21**,∞]
p29	[**19**,29]	[18,29]	[18,∞]
p30	[20,**60**]	[21,∞]	[21,∞]

Table 4.1 presents our results for 30 instances from the pathways domain. Numbers in bold indicate an upper or lower bound obtained by one query strategy that was strictly better than the bound obtained by any other query strategy. Not surprisingly, S_2 always obtains upper bounds that are as good or better than those obtained by the ramp-up strategy. Interestingly, the lower bounds obtained by S_2 are only slightly worse, differing by at most two parallel steps from the lower bound obtained by the ramp-up strategy. Examining the ratio of the upper and lower bounds obtained by S_2, we see that for 26 out of the 30 instances it finds a plan whose makespan is (provably) at most 1.5 times optimal, and for all but one instance it obtains a plan whose makespan is at most two times optimal. In contrast, the ramp-up strategy does not find a feasible plan for 21 of the 30 instances. Thus on the pathways domain, the modified version of SATPLAN using query strategy S_2 gives behavior that is in many ways better than that of the original.

To better understand the performance of S_2, we also compared it to a geometric query strategy S_g inspired by Algorithm B of Rintanen [69]. This query strategy behaves as follows. It initializes T to 1. If l and u are the initial lower and upper bounds, it then executes the queries $\langle k, T\gamma^{k-l}\rangle$ for each $k = \{l, l+1, \ldots, u-1\}$, where $\gamma \in (0,1)$ is a parameter. It then updates l and u, doubles T, and repeats. Based on the results of Rintanen [69] we set $\gamma = 0.8$. We do not compare to Rintanen's Algorithm B directly because it requires many runs of the SAT solver to be performed in parallel, which requires

an impractically large amount of memory for some of the benchmark instances considered in our evaluation.

The results for S_g are shown in the second column of Table 4.1. Like S_2, S_g always obtains upper bounds that are as good or better than those of the ramp-up strategy. Compared to S_2, S_g generally obtains slightly better lower bounds and slightly worse upper bounds. Unlike S_2, S_g does not obtain any non-trivial upper bound for three of the 30 instances.

Similar tables for the remaining six problem domains are available online at `http://www.cs.cmu.edu/~matts/icaps07/appendixA.pdf`. For the `storage`, `rovers`, and `trucks` domains, our results are similar to the ones presented in Table 4.1: S_2 achieved significantly better upper bounds than ramp-up and slightly worse lower bounds, while S_g achieved slightly better lower bounds than S_2 and slightly worse upper bounds. For the `openstacks`, `TPP`, and `pipesworld` domains, our results were qualitatively different: most instances in these domains were either easy enough that all three query strategies found a provably optimal plan, or so difficult that no strategy found a feasible plan, with the ramp-up strategy yielding the best lower bounds.

To gain more insight into these results, we plotted the function $\tau(k)$ for various instances. Broadly speaking, we encountered two types of behavior: either $\tau(k)$ increased as a function of k for $k < OPT$ but decreased as a function of k for $k \geq OPT$, or $\tau(k)$ increased as a function of k for all k. Figure 4.3 (A) and (B) give prototypical examples of these two behaviors. The gross behavior of τ on a particular instance was largely determined by the problem domain. For instances from the `pathways`, `storage`, `trucks`, and `rovers` domains τ tended to be increasing-then-decreasing, while for instances from the `TPP` and `pipesworld` domain τ tended to be monotonically increasing, explaining the qualitative difference between our results in these two sets of domains. For most instances in the `openstacks` domain we found no k values that elicited a "yes" answer in reasonable time; hence we cannot characterize the typical behavior of τ.

4.5.2 Job shop scheduling

In this section, we use query strategy S_2 to create a modified version of a branch and bound algorithm for job shop scheduling. We chose the algorithm of Brucker *et al.* [15] (henceforth referred to as `Brucker`) because it is one of the state-of-the-art branch and bound algorithms for job shop scheduling, and because code for it is freely available online.

Given a branch and bound algorithm, one can always create a decision procedure that answers the question "Does there exist a solution with cost at most k?" as follows: initialize the global upper bound to $k + 1$ (here we are assuming the objective function is

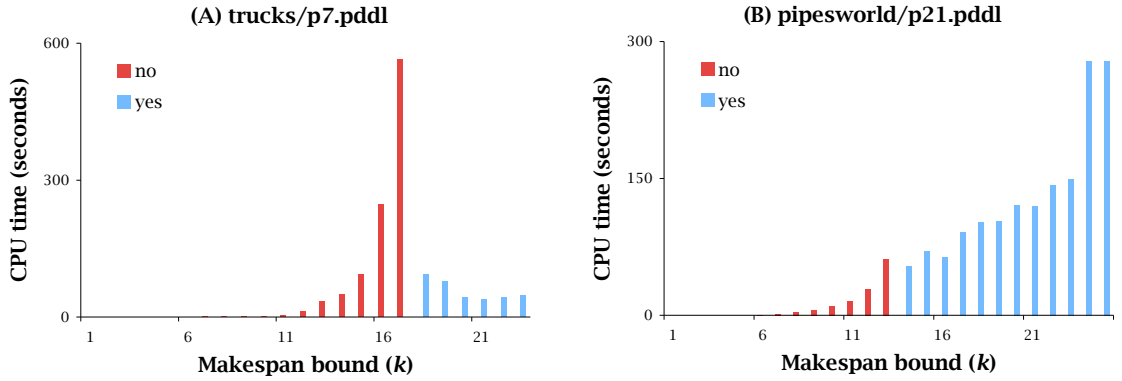

Figure 4.3: Behavior of the SAT solver `siege` running on formulae generated by `SATPLAN` to solve (A) instance `p7` from the `trucks` domain and (B) instance `p21` from the `pipesworld` domain of the 2006 International Planning Competition.

integer-valued), and run the algorithm until either a solution with cost $\leq k$ is discovered (in which case the result of the query is "yes") or the algorithm terminates without finding such a solution (in which case the result is "no"). Note that the decision procedure returns the correct answer independent of whether $k + 1$ is a valid upper bound. A query strategy can be used in conjunction with this decision procedure to find optimal or approximately optimal solutions to the original minimization problem.

We evaluate two versions of `Brucker`: the original and a modified version that uses S_2. We ran both versions on the instances in the OR library [10] with a one hour time limit per instance, and recorded the upper and lower bounds obtained. We do not evaluate the ramp-up strategy or S_g in this context, because they were not intended to work well on problems such as job shop scheduling, where the number of possible k values is very large.

On 50 of the benchmark instances, both query strategies found a (provably) optimal solution within the time limit. Table 4.2 presents the results for the remaining instances. As in Table 4.1, bold numbers indicate an upper or lower bound that was strictly better than the one obtained by the competing algorithm.

With the exception of just one instance (`la25`), the modified algorithm using query strategy S_2 obtains better lower bounds than the original branch and bound algorithm. This is not surprising, because the lower bound obtained by running the original branch and bound algorithm is simply the value obtained by solving the relaxed subproblem at the root node of the search tree, and is not updated as the search progresses. What is

Table 4.2: Performance of two query strategies on benchmark instances from the OR library. Bold numbers indicate the (strictly) best upper/lower bound we obtained.

Instance	**Brucker (S_2)** [lower,upper]	**Brucker (original)** [lower,upper]
abz7	[650,**712**]	[650,726]
abz8	[**622**,**725**]	[597,767]
abz9	[**644**,**728**]	[616,820]
ft20	[**1165**,**1165**]	[1164,1179]
la21	[**1038**,1070]	[995,**1057**]
la25	[971,979]	[**977**,**977**]
la26	[1218,1227]	[1218,**1218**]
la27	[1235,1270]	[1235,1270]
la28	[1216,**1221**]	[1216,1273]
la29	[**1118**,1228]	[1114,**1202**]
la38	[**1176**,1232]	[1077,**1228**]
la40	[**1211**,1243]	[1170,**1226**]
swv01	[**1391**,**1531**]	[1366,1588]
swv02	[1475,**1479**]	[1475,1719]
swv03	[**1373**,1629]	[1328,**1617**]
swv04	[**1410**,**1632**]	[1393,1734]
swv05	[**1414**,**1554**]	[1411,1733]
swv06	[**1572**,**1943**]	[1513,2043]
swv07	[**1432**,**1877**]	[1394,1932]
swv08	[**1614**,**2120**]	[1586,2307]
swv09	[1594,**1899**]	[1594,2013]
swv10	[**1603**,**2096**]	[1560,2104]
swv11	[2983,**3407**]	[2983,3731]
swv12	[**2971**,**3455**]	[2955,3565]
swv13	[3104,**3503**]	[3104,3893]
swv14	[2968,**3350**]	[2968,3487]
swv15	[2885,**3279**]	[2885,3583]
yn1	[**813**,**987**]	[763,992]
yn2	[**835**,**1004**]	[795,1037]
yn3	[**812**,**982**]	[793,1013]
yn4	[**899**,**1158**]	[871,1178]

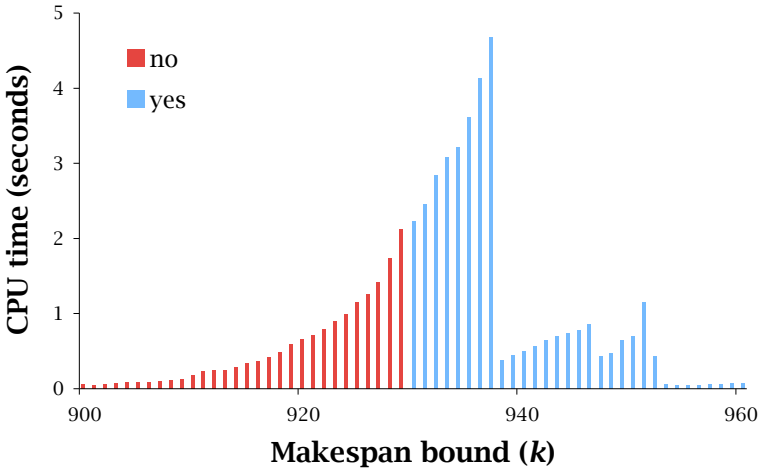

Figure 4.4: Behavior of `Brucker` running on OR library instance `ft10`.

more surprising is that the upper bounds obtained by S_2 are also, in the majority of cases, substantially better than those obtained by the original algorithm. This indicates that the speculative upper bounds created by S_2's queries are effective in pruning away irrelevant regions of the search space and forcing the branch and bound algorithm to find low-cost schedules more quickly. These results are especially promising given that the technique used to obtain them is domain-independent and could be applied to other branch and bound algorithms. In related work, Streeter & Smith [81] improved the performance of `Brucker` by using an iterated local search algorithm for job shop scheduling to obtain *valid* upper bounds and also to refine the branch ordering heuristic.

To better understand these results, we manually examined the function $\tau(k)$ for a number of instances from the OR library. In all cases, we found that $\tau(k)$ increased smoothly up to a point and then rapidly decreased in a jagged fashion. Figure 4.4 illustrates this behavior. The smooth increase of $\tau(k)$ as a function of k for $k < OPT$ reflects the fact that proving that no schedule of makespan $\leq k$ exists becomes more difficult as k gets closer to OPT. The jaggedness of $\tau(k)$ for $k \geq OPT$ can be seen as an interaction between two factors: for $k \geq OPT$, increasing k leads to less pruning (increasing $\tau(k)$) but also to a weaker termination criterion (reducing it). In spite of this, the curve has low stretch overall, and thus its shape can be exploited by query strategies such as S_2.

4.6 Conclusions

Optimization problems are often solved using an algorithm for the corresponding decision problem as a subroutine. In this chapter, we considered the problem of choosing which queries to submit to the decision procedure so as to obtain an (approximately) optimal solution as quickly as possible. Our main contribution was a new query strategy S_2 that has attractive theoretical guarantees and appears to perform well in practice. Experimentally, we showed that S_2 can be used to create improved versions of state-of-the-art algorithms for classical A.I. planning and job shop scheduling. Given the success of our experiments with a branch and bound algorithm for job shop scheduling, an interesting direction for future work would be to apply S_2 in other domains where branch and bound algorithms work well, for example integer programming or resource-constrained project scheduling.

Chapter 5

The Max k-Armed Bandit Problem

5.1 Introduction

In the classical k-armed bandit problem one is faced with a set of k slot machines, each of which has an arm that, when pulled, yields a payoff drawn independently at random from a fixed (but unknown) distribution. The goal is to allocate trials to the arms so as to maximize the cumulative payoff received over a series of n trials. Solving the problem entails striking a balance between exploration (determining which arm yields the highest mean payoff) and exploitation (repeatedly pulling this arm).

In the max variant of the k-armed bandit problem, the goal is to maximize the *maximum* (rather than cumulative) payoff. This version of the problem arises in practice when tackling combinatorial optimization problems for which a number of randomized search heuristics exist: given k heuristics, each yielding a stochastic outcome when applied to some particular problem instance, we wish to allocate trials to the heuristics so as to maximize the maximum payoff (e.g., the maximum number of clauses satisfied by any sampled variable assignment, the maximum quality of any sampled schedule). Cicirello and Smith [21] show that a max k-armed bandit approach yields good performance on the resource-constrained project scheduling problem with maximal time lags (RCPSP/max), a difficult real-world scheduling problem.

Formally, an instance $I = \langle G_1, G_2, \ldots, G_k \rangle$ of the max k-armed bandit problem is a k-tuple of probability distributions, each thought of as an arm on a slot machine. The i^{th} arm, when pulled, returns a payoff drawn independently at random from distribution G_i. A *strategy* $\mathcal{S}$ is a rule for determining which arm to pull next, as a function of the results of previous pulls. For any strategy $\mathcal{S}$, instance I, and positive integer n, let $M(I, \mathcal{S}, n)$

denote the maximum payoff obtained by following strategy $\mathcal{S}$ on instance I for n pulls. The *regret* of strategy $\mathcal{S}$ on instance I after n pulls is equal to

$$\max_{1 \leq i \leq k} \left\{ \mathbb{E}\left[M_n^i\right] \right\} - \mathbb{E}\left[M(I, \mathcal{S}, n)\right] \tag{5.1}$$

where M_n^i denotes maximum of n independent draws from G_i (i.e., the maximum payoff obtained by pulling arm i every time). Note that, in contrast to the classical k-armed bandit problem (where the goal is to maximize cumulative payoff), the optimal strategy for a particular instance I does not necessarily consist of pulling a single arm for all n pulls.[1] Thus, it is possible for a strategy to have negative regret on some instances.

The worst-case regret of strategy $\mathcal{S}$ is the maximum value of (5.1) as a function of k and n. We say that $\mathcal{S}$ is a *no-regret strategy* if, for any fixed k, the worst-case regret is $o(1)$ as a function of n. Note that this is stronger than simply requiring that, for any *particular* instance, the regret approaches zero as $n \to \infty$. Indeed, as long as payoffs are bounded then simply sampling the arms in round-robin order meets the latter requirement. However, as Theorem 29 in the next section shows, round-robin sampling is not a no-regret strategy.

In this chapter, we present two strategies for the max k-armed bandit problem. We first discuss a simple strategy called Threshold Ascent which is designed to work well for a wide variety of payoff distributions encountered in practice (Theorem 29 shows that no strategy can be expected to work well for all payoff distributions). We then discuss a second strategy that has strong theoretical guarantees in the special case where each payoff distribution is a generalized extreme value (GEV) distribution (defined in §5.3.1). The motivation for studying this special case is the Extremal Types Theorem [23], which singles out the GEV as the limiting distribution of the maximum of a large number of independent identically distributed (i.i.d.) random variables. Roughly speaking, one can think of the Extremal Types Theorem as an analogue of the Central Limit Theorem. Just as the Central Limit Theorem states that the sum of a large number of i.i.d. random variables converges in distribution to a Gaussian, the Extremal Types Theorem states that the maximum of a large number of i.i.d. random variables converges in distribution to a GEV.

The results presented in this chapter are based on two conference papers [80, 82].

[1] As an example, suppose there are two arms. Arm A always returns payoff $\frac{1}{2}$, while arm B returns payoff 1 with probability 0.01 and payoff zero otherwise. For large n, the optimal strategy is to pull arm A once and then pull arm B the remaining $n - 1$ times.

5.1.1 Related work

The classical k-armed bandit problem was first studied by Robbins [70] and has since been the subject of numerous papers; see Berry and Fristedt [11] and Kaelbling [40] for overviews.

The max variant of the k-armed bandit problem was introduced by Cicirello and Smith [19, 21], whose experiments with randomized priority dispatching rules for the RCPSP/max form the basis of our experimental evaluation in §5.4. Cicirello and Smith show that their max k-armed bandit strategy yields performance on the RCPSP/max that is competitive with the state of the art. The design of Cicirello and Smith's strategy is motivated by an analysis of the special case in which each arm's payoff distribution is a GEV distribution with shape parameter $\xi = 0$.

5.1.2 A negative result for arbitrary payoff distributions

Ideally, we would like to come up with a no-regret strategy for the max k-armed bandit problem that requires as few distributional assumptions as possible. As a first step, it seems reasonable to require that all payoffs come from a bounded interval (this will be true in all the applications we intend to consider). In fact, a no-regret strategy does not exist even under the stronger assumption that all payoffs are either 0 or 1, as the following theorem shows.

Theorem 29. *For any max k-armed bandit strategy $\mathcal{S}$ and any positive integer n, there exists a max k-armed bandit instance on which $\mathcal{S}$ has regret at least $1 - \frac{1}{k} - \frac{1}{e}$ after n pulls.*

Proof. Let $I_j = \langle G_1, G_2, \ldots, G_k \rangle$ denote a max k-armed bandit instance in which distribution G_i always returns payoff 0 for $i \neq j$, while distribution G_j returns payoff 1 with probability $\frac{1}{n}$ and returns payoff 0 otherwise. Thus, pulling arm G_j for all n pulls yields expected maximum payoff

$$1 - \left(1 - \frac{1}{n}\right)^n \geq 1 - \frac{1}{e}\,.$$

It suffices to show that for some choice of j, $\mathcal{S}$ receives expected maximum payoff at most $\frac{1}{k}$. To see this, assume without loss of generality that the behavior of $\mathcal{S}$ is unaffected by the payoffs it receives. This is without loss of generality because all payoffs are in $\{0, 1\}$, and once $\mathcal{S}$ receives a payoff of 1 its subsequent choices have no effect on the maximum payoff that it receives. Under this assumption, there must be some arm j whose

expected number of pulls is $\leq \frac{n}{k}$. Thus on instance I_j, the expected *total* payoff that $\mathcal{S}$ receives is $\leq \frac{1}{k}$, which implies that the expected maximum payoff is $\leq \frac{1}{k}$. □

5.2 A Simple, Distribution-Free Approach

In this section, we do not assume that the payoff distributions belong to any specific parametric family. In fact, we will not make any formal assumptions at all about the payoff distributions, although (as Theorem 29 shows) our approach cannot be expected to work well if the distributions are chosen adversarially. Roughly speaking, our approach will work best when the following two criteria are satisfied.

1. There is a (relatively low) threshold $t_{critical}$ such that, for all $t > t_{critical}$, the arm that is most likely to yield a payoff $> t$ is the same as the arm most likely to yield a payoff $> t_{critical}$. Call this arm i^*.

2. As t increases beyond $t_{critical}$, there is a growing gap between the probability that arm i^* yields a payoff $> t$ and the corresponding probability for other arms. Specifically, if we let $p_i(t)$ denote the probability that the i^{th} arm returns a payoff $> t$, the ratio $\frac{p_{i^*}(t)}{p_i(t)}$ should increase as a function of t for $t > t_{critical}$, for any $i \neq i^*$.

Figure 5.1 illustrates a set of two payoff distributions that satisfy these assumptions.

In this section we present a new algorithm, Chernoff Interval Estimation, for the classical k-armed bandit problem and prove a bound on its regret. Our algorithm is simple and has performance guarantees competitive with the state of the art. Building on Chernoff Interval Estimation, we develop a new algorithm, Threshold Ascent, for solving the max k-armed bandit problem. Our algorithm is designed to work well as long as the two mild distributional assumptions just described are satisfied. In §5.4 we evaluate Threshold Ascent experimentally by using it to select among randomized priority dispatching rules for the RCPSP/max. We find that Threshold Ascent (*i*) performs better than any of the priority rules perform in isolation, and (*ii*) outperforms the recent QD-BEACON max k-armed bandit algorithm of Cicirello and Smith [19, 21].

5.2.1 Chernoff Interval Estimation

In this section we present and analyze a simple algorithm, Chernoff Interval Estimation, for the classical k-armed bandit problem. In §5.2.2 we use this algorithm as subroutine in Threshold Ascent, an algorithm for the max k-armed bandit problem.

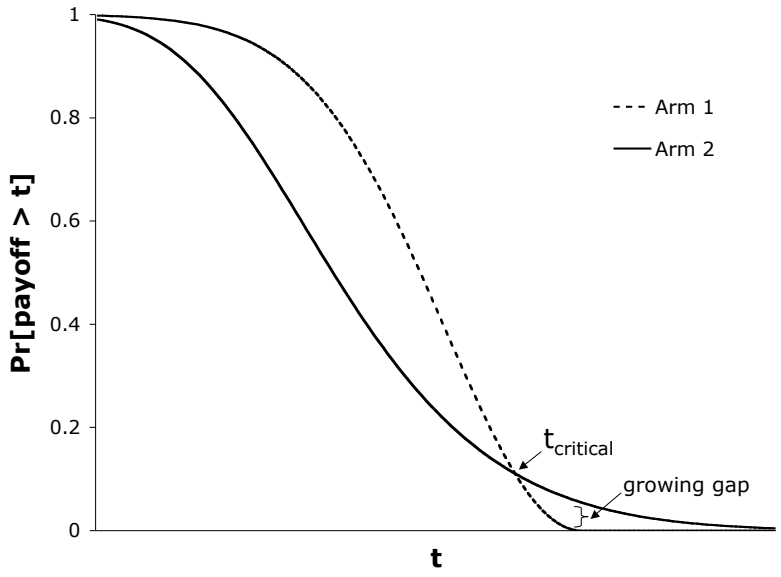

Figure 5.1: A max k-armed bandit instance on which Threshold Ascent should perform well.

In the classical k-armed bandit problem one is faced with a set of k arms. The i^{th} arm, when pulled, returns a payoff drawn independently at random from a fixed (but unknown) distribution. All payoffs are real numbers between 0 and 1. We denote by μ_i the expected payoff obtained from a single pull of arm i, and define $\mu^* = \max_{1 \leq i \leq k} \mu_i$. We consider the finite-time version of the problem, in which our goal is to maximize the cumulative payoff received using a fixed budget of n pulls. The *regret* of an algorithm (on a particular instance of the classical k-armed bandit problem) is the difference between $\mu^* n$ (the expected cumulative payoff the algorithm would have received by pulling the single best arm n times) and the expected cumulative payoff the algorithm receives on the instance.

Chernoff Interval Estimation is simply the well-known interval estimation algorithm [40, 57] with confidence intervals derived using Chernoff's inequality. Although various interval estimation algorithms have been analyzed in the literature and a variety of guarantees have been proved, both (*i*) our use of Chernoff's inequality in an interval estimation algorithm and (*ii*) our analysis appear to be novel. In particular, when the mean payoff returned by each arm is small (relative to the maximum possible payoff) our algorithm has much better performance than the recent algorithm of Auer *et al.* [4], which is identical to our algorithm except that confidence intervals are derived using Hoeffding's inequality. We give further discussion of related work later in this section.

Classical k-armed bandit strategy Chernoff Interval Estimation
Input: positive integer n, real number $\delta \in (0, 1)$

1. Initialize $x_i \leftarrow 0$, $n_i \leftarrow 0 \; \forall i \in \{1, 2, \ldots, k\}$.
2. Repeat n times:
   (a) $\hat{i} \leftarrow \arg\max_i U(\bar{\mu}_i, n_i)$, where $\bar{\mu}_i = \frac{x_i}{n_i}$ and

$$U(\mu, n_0) = \begin{cases} \mu + \frac{\alpha + \sqrt{2n_0\mu\alpha + \alpha^2}}{n_0} & \text{if } n_0 > 0 \\ \infty & \text{otherwise} \end{cases}$$

   where $\alpha = \ln\left(\frac{2nk}{\delta}\right)$.
   (b) Pull arm $\hat{i}$, receive payoff R, set $x_{\hat{i}} \leftarrow x_{\hat{i}} + R$, and set $n_{\hat{i}} \leftarrow n_{\hat{i}} + 1$.

We now bound the expected regret of Chernoff Interval Estimation. Our analysis proceeds as follows. Lemma 13 shows that (with a certain minimum probability) the value $U(\bar{\mu}_i, n_i)$ is always an upper bound on μ_i. Lemma 14 then places a bound on the number of times the algorithm will sample an arm whose mean payoff is suboptimal. Theorem 30 puts these results together to obtain a bound on Chernoff Interval Estimation's worst-case regret.

We will make use of the following well-known Chernoff bound, which we simply refer to as "Chernoff's inequality".

Chernoff's inequality. *Let $X = \sum_{i=1}^{n} X_i$ be the sum of n independent identically distributed random variables with $X_i \in [0, 1]$ and $\mu = \mathbb{E}[X_i]$. Then for $\beta > 0$,*

$$\mathbb{P}\left[\frac{X}{n} < (1 - \beta)\mu\right] < \exp\left(-\frac{n\mu\beta^2}{2}\right)$$

and

$$\mathbb{P}\left[\frac{X}{n} > (1 + \beta)\mu\right] < \exp\left(-\frac{n\mu\beta^2}{3}\right) .$$

We will also use the following algebraic fact, which holds by construction.

Fact 3. *If $z = U(\mu, n_0)$ then*

$$zn_0\left(1 - \frac{\mu}{z}\right)^2 = 2\alpha .$$

Lemma 13. *During a run of Chernoff Interval Estimation*(n, δ) *it holds with probability at least* $1 - \frac{\delta}{2}$ *that for all arms* $i \in \{1, 2, \ldots, k\}$ *and for all* n *repetitions of the loop,* $U(\bar{\mu}_i, n_i) \geq \mu_i$.

Proof. It suffices to show that for any arm i and any particular repetition of the loop, $\mathbb{P}\left[U(\bar{\mu}_i, n_i) < \mu_i\right] < \frac{\delta}{2nk}$. Consider some particular fixed values of μ_i, α, and n_i, and let μ_c be the largest solution to the equation

$$U(\mu_c, n_i) = \mu_i \tag{5.2}$$

By inspection, $U(\mu_c, n_i)$ is strictly increasing as a function of μ_c. Thus $U(\bar{\mu}_i, n_i) < \mu_i$ if and only if $\bar{\mu}_i < \mu_c$, so $\mathbb{P}\left[U(\bar{\mu}_i, n_i) < \mu_i\right] = \mathbb{P}\left[\bar{\mu}_i < \mu_c\right]$. Thus

$$\begin{aligned}
\mathbb{P}\left[U(\bar{\mu}_i, n_i) < \mu_i\right] &= \mathbb{P}\left[\bar{\mu}_i < \mu_c\right] \\
&= \mathbb{P}\left[\bar{\mu}_i < \mu_i \left(1 - \left(1 - \frac{\mu_c}{\mu_i}\right)\right)\right] \\
&< \exp\left(-\frac{\mu_i n_i}{2}\left(1 - \frac{\mu_c}{\mu_i}\right)^2\right) && \text{(Chernoff's inequality)} \\
&= \exp\left(-\alpha\right) && \text{(Fact 3 and equation 5.2)} \\
&= \frac{\delta}{2nk}\,.
\end{aligned}$$

□

Lemma 14. *During a run of Chernoff Interval Estimation*(n, δ) *it holds with probability at least* $1 - \delta$ *that each suboptimal arm* i *(i.e., each arm* i *with* $\mu_i < \mu^*$*) is pulled at most* $\frac{3\alpha}{\mu^*} \frac{1}{(1 - \sqrt{y_i})^2}$ *times, where* $y_i = \frac{\mu_i}{\mu^*}$.

Proof. Let i^* be some optimal arm (i.e., $\mu_{i^*} = \mu^*$) and assume that $U(\bar{\mu}_{i^*}, n_{i^*}) \geq \mu^*$ for all n repetitions of the loop. By Lemma 13, this assumption is valid with probability at least $1 - \frac{\delta}{2}$. Consider some particular suboptimal arm i. By inspection, we will stop sampling arm i once $U(\bar{\mu}_i, n_i) < \mu^*$. So it suffices to show that if

$$n_i \geq \frac{3\alpha}{\mu^*} \frac{1}{(1 - \sqrt{y_i})^2} \tag{5.3}$$

then $U(\bar{\mu}_i, n_i) < \mu^*$ with probability at least $1 - \frac{\delta}{2k}$ (then the probability that any of our assumptions fail is at most $\frac{\delta}{2} + k\frac{\delta}{2k} = \delta$).

To show this, it suffices to show that (5.3) implies two things. First, with probability at least $1 - \frac{\delta}{2k}$, we have $\bar{\mu}_i \leq \sqrt{y_i^{-1}}\mu_i$. Second, if $\bar{\mu}_i \leq \sqrt{y_i^{-1}}\mu_i$, then $U(\bar{\mu}_i, n_i) < \mu^*$.

We first show that (5.3) implies that with probability at least $1 - \frac{\delta}{2k}$, $\bar{\mu}_i \leq \sqrt{y_i^{-1}}\mu_i$. This is true because

$$\begin{aligned}
\mathbb{P}\left[\bar{\mu}_i > \sqrt{y_i^{-1}}\mu_i\right] &= \mathbb{P}\left[\bar{\mu}_i > \left(1 + \frac{1-\sqrt{y_i}}{\sqrt{y_i}}\right)\mu_i\right] \\
&< \exp\left(-\frac{n_i\mu_i}{3}\frac{(1-\sqrt{y_i})^2}{y_i}\right) && \text{(Chernoff's inequality)} \\
&= \exp\left(-\frac{n_i\mu^*}{3}(1-\sqrt{y_i})^2\right) \\
&< \exp\left(-\alpha\right) && \text{(equation 5.3)} \\
&= \frac{\delta}{2nk} < \frac{\delta}{2k}\,.
\end{aligned}$$

To complete the proof, we show that (5.3) implies that if $\bar{\mu}_i \leq \sqrt{y_i^{-1}}\mu_i$, then $U(\bar{\mu}_i, n_i) < \mu^*$. To see this, let $U_i = U(\bar{\mu}_i, n_i)$, and suppose for contradiction that $U_i \geq \mu^*$. By Fact 3,

$$n_i = \frac{2\alpha}{U_i}\left(1 - \frac{\bar{\mu}_i}{U_i}\right)^{-2}\,.$$

The right hand side increases as a function of $\bar{\mu}_i$ (assuming $\bar{\mu}_i < U_i$, which is true by definition). So if $\bar{\mu}_i \leq \sqrt{y_i^{-1}}\mu_i$ then replacing $\bar{\mu}_i$ with $\sqrt{y_i^{-1}}\mu_i$ only increases the value of the right hand side. Similarly, the right hand side decreases as a function of U_i, so if $U_i \geq \mu^*$ then replacing replacing U_i with μ^* only increases the value of the right hand side. Thus

$$n_i \leq \frac{2\alpha}{\mu^*}\left(1 - \frac{\sqrt{y_i^{-1}}\mu_i}{\mu^*}\right)^{-2} = \frac{2\alpha}{\mu^*}\left(1 - \sqrt{y_i}\right)^{-2}$$

which contradicts (5.3). □

The following theorem shows that when n is large (and the parameter δ is small), the total payoff obtained by Chernoff Interval Estimation over n trials is almost as high as what would be obtained by pulling the single best arm for all n trials.

Theorem 30. *The regret incurred by Chernoff Interval Estimation*(n, δ) *is at most*

$$2\sqrt{3\mu^* n(k-1)\alpha} + \delta\mu^* n$$

where $\alpha = \ln\left(\frac{2nk}{\delta}\right)$.

Proof. First, assume that $k \geq 2$ (for $k = 1$ the theorem holds trivially).

The conclusion of Lemma 14 fails to hold with probability at most δ. Because regret cannot exceed $\mu^* n$, this scenario contributes at most $\delta\mu^* n$ to overall regret. Thus it remains to show that, conditioned on the event that the conclusion of Lemma 2 holds, regret is at most $2\sqrt{3\mu^* n(k-1)\alpha}$.

Consider some arm i with $\mu_i < \mu^*$. Let $y = \frac{\mu_i}{\mu^*}$, and let n_i be the number of times arm i is pulled. By Lemma 14, $n_i \leq \frac{3\alpha}{\mu^*}\frac{1}{(1-\sqrt{y})^2}$. Each pull of arm i adds $\mu^* - \mu_i = \mu^*(1-y)$ to the regret. Thus, letting R_i be the total regret incurred due to pulling arm i, we have

$$R_i = n_i\mu^*(1-y) \leq \frac{3\alpha(1-y)}{(1-\sqrt{y})^2}\,.$$

Using the fact that $y < 1$, we have

$$\begin{aligned}\frac{1-y}{(1-\sqrt{y})^2} &= \frac{1-y}{(1-\sqrt{y})^2}\cdot\frac{(1+\sqrt{y})^2}{(1+\sqrt{y})^2} \\ &= \frac{(1+\sqrt{y})^2}{1-y} \\ &< \frac{4}{1-y}\,.\end{aligned}$$

Thus

$$R_i \leq \min\left\{n_i\mu^*(1-y), \frac{12\alpha}{1-y}\right\}\,.$$

For any fixed n_i, this expression is maximized when $1 - y = 2\sqrt{\frac{3\alpha}{n_i\mu^*}}$. Thus $R_i \leq 2\sqrt{3n_i\mu^*\alpha}$.

Assume without loss of generality that arm 1 is optimal (i.e., $\mu_1 = \mu^*$). Then the total regret is at most $\sum_{i=2}^{k} R_i \leq \sum_{i=2}^{k} 2\sqrt{3n_i\mu^*\alpha}$. Subject to the constraint $\sum_{i=2}^{k} n_i \leq n$, this expression is maximized when $n_i = \frac{n}{k-1}$ for all i ($2 \leq i \leq k$). Thus the total regret is at most $2\sqrt{3\mu^* n(k-1)\alpha}$, as claimed. □

We now compare Theorem 30 to previous regret bounds for the classical k-armed bandit problem.

Types of Regret Bounds

In comparing the regret bound of Theorem 30 to previous work, we must distinguish between two different types of regret bounds. The first type of bound describes the asymp-

totic behavior of regret (as $n \to \infty$) on a *fixed* problem instance (i.e., with all k payoff distributions held constant). In this framework, a lower bound of $\Omega(\ln(n))$ has been proved, and algorithms exist that achieve regret $O(\ln(n))$ [4]. Though we do not prove it here, Chernoff Interval Estimation also achieves $O(\ln(n))$ regret in this framework when δ is set appropriately.

The second type of bound concerns the maximum, over all possible instances, of the regret incurred by the algorithm when run on that instance for n pulls. In this setting, a lower bound of $\Omega(\sqrt{kn})$ has been proved [5]. It is this second form of bound that Theorem 30 provides. In what follows, we will only consider bounds of this second form.

The Classical k-Armed Bandit Problem

We are not aware of any work on the classical k-armed bandit problem that offers a better regret bound (of the second form) than the one proved in Theorem 30. Auer *et al.* [4] analyze an algorithm that is identical to ours except that the confidence intervals are derived from Hoeffding's inequality rather than Chernoff's inequality. An analysis analogous to the one given in this chapter shows that their algorithm has worst-case regret $O(\sqrt{nk\ln(n)})$ when the instance is chosen adversarially as a function of n. Plugging $\delta = \frac{1}{n^2}$ into Theorem 30 gives a bound of $O(\sqrt{n\mu^* k\ln(n)})$, which is never any worse than the latter bound (because $\mu^* \leq 1$) and is much better when μ^* is small.

The Nonstochastic Multiarmed Bandit Problem

In a different paper, Auer *et al.* [5] consider a variant of the classical k-armed bandit problem in which the sequence of payoffs returned by each arm is determined adversarially in advance. For this more difficult problem, they present an algorithm called **Exp3.1** with expected regret

$$8\sqrt{(e-1)G_{\max}k\ln(k)} + 8(e-1)k + 2k\ln(k)$$

where $G_{\max}$ is the maximum, over all k arms, of the total payoff that would be obtained by pulling that arm for all n trials. If we plug in $G_{\max} = \mu^* n$, this bound is sometimes better than the one given by Theorem 30 and sometimes not, depending on the values of n, k, and μ^*, as well as the choice of the parameter δ.

5.2.2 Threshold Ascent

To solve the max k-armed bandit problem, we use Chernoff Interval Estimation to maximize the number of payoffs that exceed a threshold T that varies over time. Initially, we set T to zero. Whenever s or more payoffs $> T$ have been received so far, we increment T. We refer to the resulting algorithm as Threshold Ascent. To ease explanation, we assume that all payoffs are integer multiples of some known constant Δ when presenting the code for Threshold Ascent.

Max k-armed bandit strategy Threshold Ascent
Input: positive integers n and s, real number $\delta \in (0, 1)$.

1. Initialize $T \leftarrow 0$ and $n_i^R = 0$, $\forall i \in \{1, 2, \ldots, k\}, R \in \{0, \Delta, 2\Delta, \ldots, 1 - \Delta, 1\}$.
2. Repeat n times:
   (a) While $\sum_{i=1}^{k} S_i(T) \geq s$ do:

   $$T \leftarrow T + \Delta$$

   where $S_i(t) = \sum_{R>t} n_i^R$ is the number of payoffs $> t$ received so far from arm i.
   (b) $\hat{i} \leftarrow \arg\max_i U\left(\frac{S_i(T)}{n_i}, n_i\right)$, where $n_i = \sum_R n_i^R$ is the number of times arm i has been pulled and

   $$U(\mu, n_0) = \begin{cases} \mu + \frac{\alpha + \sqrt{2n_0\mu\alpha + \alpha^2}}{n_0} & \text{if } n_0 > 0 \\ \infty & \text{otherwise} \end{cases}$$

   where $\alpha = \ln\left(\frac{2nk}{\delta}\right)$.
   (c) Pull arm $\hat{i}$, receive payoff R, and set $n_{\hat{i}}^R \leftarrow n_{\hat{i}}^R + 1$.

The parameter s controls the tradeoff between exploration and exploitation. To understand this tradeoff, it is helpful to consider two extreme cases.

Case $s = 1$. Threshold Ascent$(1, n, \delta)$ is equivalent to round-robin sampling. When $s = 1$, the threshold T is incremented whenever a payoff $> T$ is obtained. Thus the value

$\frac{S_i(T)}{n_i}$ calculated in 2 (b) is always 0, so the value of $U\left(\frac{S_i(T)}{n_i}, n_i\right)$ is determined strictly by n_i. Because U is a decreasing function of n_i, the algorithm simply samples whatever arm has been sampled the smallest number of times so far.

Case $s = \infty$. Threshold Ascent(∞, n, δ) is equivalent to Chernoff Interval Estimation (n, δ) running on a k-armed bandit instance where payoffs $> T$ are mapped to 1 and payoffs $\leq T$ are mapped to 0.

5.3 A No-Regret Algorithm for GEV Payoff Distributions

In this section we consider a restricted version of the max k-armed bandit problem in which each arm yields payoff drawn from a generalized extreme value (GEV) distribution (defined in §5.3.1). This section presents the first provably asymptotically optimal algorithm for this problem.

Roughly speaking, the reason for assuming a GEV distribution is the Extremal Types Theorem (stated in §5.3.1), which states that the distribution of the sample maximum of n independent identically distributed random variables approaches a GEV distribution as $n \to \infty$. In fact, there are two arguments for assuming that each arm is a GEV distribution. First, in practice the distribution of payoffs returned by a strong heuristic may be approximately GEV, even if the conditions of the Extremal Types Theorem are not formally satisfied [19].

A second argument runs as follows. Suppose $I = \langle G_1, G_2, \ldots, G_k \rangle$ is an instance of the max k-armed bandit problem in which each distribution G_i satisfies the conditions required by the Extremal Types Theorem. Consider the instance $\bar{I} = \langle \bar{G}_1, \bar{G}_2, \ldots, \bar{G}_k \rangle$, where $\bar{G}_i$ returns the maximum of m samples from G_i. Effectively, $\bar{I}$ is a restricted version of I in which the arms must be pulled in batches of size m, rather than in any arbitrary order. For m sufficiently large, the Extremal Types Theorem guarantees that for each i, $\bar{G}_i$ is approximately equal to a GEV, call it G'_i. Thus, the instance $I' = \langle G'_1, G'_2, \ldots, G'_k \rangle$ is approximately equivalent to the original instance I, and satisfies our distributional assumptions.

The form our algorithm is very simple. Initially, the algorithm pulls each arm a fixed number of times. Based on the observed payoffs, the algorithm then estimates, for each arm, the (expected) maximum payoff that would be obtained by pulling that arm for all remaining trials. The arm with the highest estimate is then used for all remaining trials.

An algorithm of this form has previously been analyzed for the classical k-armed bandit problem [28]. As it turns out, the analysis in the case of the max k-armed bandit problem is considerably more technical.

For reasons that will become clear, the nature of our results depends on the shape parameter (ξ) of the GEV distributions. Assuming all arms have $\xi \leq 0$, we obtain a strategy whose regret is $o(1)$. In the exotic case where one or more arms have $\xi > 0$, the expected maximum payoff obtained by pulling the best arm n times grows without bound, and grows too fast for us to be able to obtain additive regret that is $o(1)$. In this case, we obtain expected maximum payoff within a factor $1 - o(1)$ of that of the best arm.

We should note up front that the results presented in this section are primarily of theoretical interest, and are quite a bit more technical than the results presented in the previous section. In our experimental evaluation, we found that Threshold Ascent performed much better than the no-regret strategy for GEV payoff distributions described in this section.

5.3.1 Background: extreme value theory

This section provides a self-contained overview of results in extreme value theory that are relevant to this work. Our presentation is based on the text by Coles [23].

The central result of extreme value theory is an analogue of the Central Limit Theorem that applies to extremely rare events. Recall that the Central Limit Theorem states that (under certain regularity conditions) the distribution of the sum of n independent, identically distributed (i.i.d) random variables converges to a normal distribution as $n \to \infty$. The Extremal Types Theorem states that (under certain regularity conditions) the distribution of the maximum of n i.i.d random variables converges to a generalized extreme value (GEV) distribution.

Definition (GEV distribution). *A random variable Z has a* generalized extreme value distribution *if, for constants μ, $\sigma > 0$, and ξ, $\mathbb{P}[Z \leq z] = GEV_{(\mu,\sigma,\xi)}(z)$, where*

$$GEV_{(\mu,\sigma,\xi)}(z) = \exp\left(-\left(1 + \frac{\xi(z-\mu)}{\sigma}\right)^{-\frac{1}{\xi}}\right)$$

for z such that $1 + \xi(z-\mu)\sigma^{-1} > 0$, and $GEV_{(\mu,\sigma,\xi)}(z) = 1$ otherwise. The case $\xi = 0$ is interpreted as the limit

$$\lim_{\xi' \to 0} GEV_{(\mu,\sigma,\xi')}(z) = \exp\left(-\exp\left(\frac{\mu - z}{\sigma}\right)\right) .$$

The following three propositions establish properties of the GEV distribution.

Proposition 1. *Let Z be a random variable with $\mathbb{P}[Z \leq z] = GEV_{(\mu,\sigma,\xi)}(z)$. Then*

$$\mathbb{E}[Z] = \begin{cases} \mu + \frac{\sigma}{\xi}(\Gamma(1-\xi) - 1) & \text{if } \xi < 1, \xi \neq 0 \\ \mu + \sigma\gamma & \text{if } \xi = 0 \\ \infty & \text{if } \xi \geq 1 \end{cases}$$

where

$$\Gamma(z) = \int_0^\infty t^{z-1} \exp(-t)\, dt$$

is the complete gamma function and

$$\gamma = \lim_{n\to\infty} \left(\sum_{k=1}^{n} \frac{1}{k} - \ln(n) \right)$$

is Euler's constant.

We now introduce some additional notation. Let $G = GEV_{(\mu,\sigma,\xi)}$ be a GEV distribution, and let the random variable M_n equal the maximum of n independent samples from G.

Proposition 2. *M_n has distribution $GEV_{(\mu',\sigma',\xi')}$, where*

$$\mu' = \begin{cases} \mu + \frac{\sigma}{\xi}(n^\xi - 1) & \text{if } \xi \neq 0 \\ \mu + \sigma \ln(n) & \text{otherwise,} \end{cases}$$
$$\sigma' = \sigma n^\xi, \text{ and}$$
$$\xi' = \xi\,.$$

Substituting the parameters of M_n given by Proposition 2 into Proposition 1 gives an expression for $\mathbb{E}[M_n]$.

Proposition 3. *Let $G = GEV_{(\mu,\sigma,\xi)}$ where $\xi < 1$. Then*

$$\mathbb{E}[M_n] = \begin{cases} \mu + \frac{\sigma}{\xi}\left(n^\xi \Gamma(1-\xi) - 1\right) & \text{if } \xi \neq 0 \\ \mu + \sigma\gamma + \sigma \ln(n) & \text{otherwise.} \end{cases}$$

It follows that

- *for $\xi > 0$, $\mathbb{E}[M_n]$ is $\Theta(n^\xi)$;*

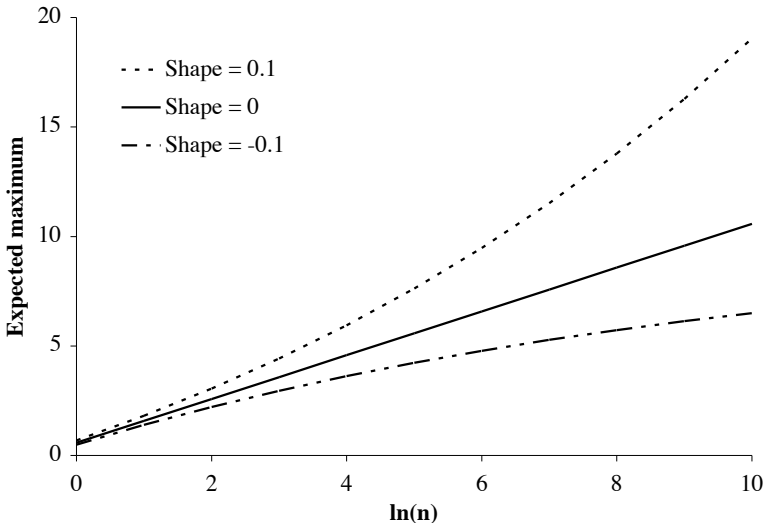

Figure 5.2: The effect of the shape parameter (ξ) on the expected maximum of n independent draws from a GEV distribution.

- *for* $\xi = 0$, $\mathbb{E}[M_n]$ *is* $\Theta(\ln(n))$*; and*
- *for* $\xi < 0$, $\mathbb{E}[M_n] = \mu - \frac{\sigma}{\xi} - \Theta(n^{\xi})$.

In the analysis that follows in later sections, it will be useful to have a visual picture of what Proposition 3 means. Figure 1 plots $\mathbb{E}[M_n]$ as a function of $\ln n$ for three GEV distributions with $\mu = 0$, $\sigma = 1$, and $\xi \in \{0.1, 0, -0.1\}$. When the shape parameter ξ is negative, the expected maximum payoff approaches an asymptote as $n \to \infty$; when $\xi = 0$, the expected maximum payoff grows linearly as a function of $\ln n$; and when $\xi > 0$, the expected maximum payoff grows super-linearly as a function of $\ln n$.

The central result of extreme value theory is the following theorem.

The Extremal Types Theorem. *Let G be an arbitrary cumulative distribution function, and suppose there exist sequences of constants $\{a_n > 0\}$ and $\{b_n\}$ such that*

$$\lim_{n\to\infty} \mathbb{P}\left[\frac{M_n^G - b_n}{a_n} \leq z\right] = G^*(z) \tag{5.4}$$

for any continuity point z of G^*, where G^* is a not a point mass. Then there exist constants μ, $\sigma > 0$, and ξ such that $G^*(z) = GEV_{(\mu,\sigma,\xi)}(z)\ \forall z$. Furthermore,

$$\lim_{n\to\infty} \mathbb{P}[M_n \leq z] = GEV_{(\mu a_n + b_n, \sigma a_n, \xi)}(z) .$$

Condition (5.4) holds for a variety of distributions including the normal, lognormal, uniform, and Cauchy distributions.

5.3.2 A no-regret algorithm

In this section we will analyze the max k-armed bandit strategy $\mathcal{S}^{GEV}$ shown below. Here, and throughout this section, we use M_n^i to denote the maximum payoff obtained by pulling the i^{th} arm n times, and we define

$$m_n^i = \mathbb{E}\left[M_n^i\right] .$$

Max k-armed bandit strategy $\mathcal{S}^{GEV}$
Input: real numbers $\epsilon > 0$, $\delta \in (0, 1)$

1. *(Exploration)* For each arm $G_i \in \mathcal{G}$:

   Using $t = O\left(\ln(\frac{1}{\delta})\frac{\ln(n)^2}{\epsilon^2}\right)$ samples of G_i, obtain an estimate $\bar{m}_n^i$ of m_n^i. Assuming that arm G_i has shape parameter $\xi_i \leq 0$, our estimate will have the property that

   $$\mathbb{P}\left[\left|\bar{m}_n^i - m_n^i\right| < \epsilon\right] \geq 1 - \delta .$$

2. *(Exploitation)* Set $\hat{i} = \arg\max_{1 \leq i \leq k} \bar{m}_n^i$, and pull arm $G_{\hat{i}}$ for the remaining $n - tk$ trials.

If an arm G_i has shape parameter $\xi_i > 0$, the estimate obtained in step 1 (a) will instead have the property that $\mathbb{P}\left[\frac{1}{1+\epsilon} < \frac{\bar{m}_n - \alpha_1}{m_n - \alpha_1} < 1 + \epsilon\right] \geq 1 - \delta$ for constant α_1 independent of n.

Assumptions

We require that each arm G_i have finite, bounded mean and variance. To ensure this, it suffices to assume that each arm $G_i = GEV_{(\mu_i, \sigma_i, \xi_i)}$ is a GEV distribution whose parameters satisfy

1. $|\mu_i| \leq \mu_u$

2. $0 < \sigma_\ell \leq \sigma_i \leq \sigma_u$

3. $\xi_\ell \leq \xi_i \leq \xi_u < \frac{1}{2}$

for known constants μ_u, σ_ℓ, σ_u, ξ_ℓ, and ξ_u.

Analysis

The following theorem shows that with appropriate settings of ϵ and δ, strategy $\mathcal{S}^{GEV}$ is asymptotically optimal when each arm has shape parameter $\xi_i \leq 0$. In Appendix A, we establish a similar guarantee (using the same parameter settings) when one or more arms have $\xi_i > 0$.

Theorem 31. *Let $I = \langle G_1, G_2, \ldots, G_k \rangle$ be an instance of the max k-armed bandit problem, where $G_i = GEV_{(\mu_i, \sigma_i, \xi_i)}$, and $\xi_i \leq 0$ for all i. Then strategy $\mathcal{S}^{GEV}$, run on instance I with parameters $\epsilon = \sqrt[3]{\frac{k}{n}}$ and $\delta = \frac{1}{kn^2}$, has regret $O\left(\ln(nk)\ln(n)^2\sqrt[3]{\frac{k}{n}}\right)$.*

Proof (sketch). There are three potential sources of regret. We will show that the contribution from each source is $O(\Delta)$, where $\Delta = \ln(nk)\ln(n)^2\sqrt[3]{\frac{k}{n}}$.

First, with probability at most $k\delta = \frac{1}{n^2}$, one of the estimates obtained during the exploration phase will be more than ϵ away from its true value. However, if all arms have $\xi_i \leq 0$ then by Proposition 3, the maximum regret is $O(\ln n)$. Thus, this possibility contributes $O\left(\frac{\ln n}{n^2}\right) = o(\Delta)$ to regret.

The second source of regret is that, even if all estimates are within ϵ of their true values, the expected maximum payoff from n pulls of the arm $\hat{i}$ selected at the end of the exploration phase could be up to 2ϵ smaller than that of some other arm. This possibility contributes at most 2ϵ to regret, and $\epsilon = O(\Delta)$.

The final source of regret is that, due to the time spent on the exploration phase, the presumed best arm $\hat{i}$ will only be pulled $n - t(k-1)$ times, rather than n times. To complete the proof, we show in Appendix A that for any arm i,

$$m_n^i - m_{n-t(k-1)}^i = O\left(\frac{tk}{n}\right) = O(\Delta) \ .$$

□

Theorem 31 completes our analysis of the performance of $\mathcal{S}^{GEV}$. It remains only to describe how the estimates in step 1 (a) are obtained.

Obtaining the required estimates

We now describe how to obtain accurate estimates of the expected maximum of n independent draws from a GEV distribution. Although the estimation procedure itself is not complicated, the proofs that the estimates have the required properties are quite technical. In this section, we provide only sketches of these proofs, deferring the full proofs to Appendix A.

We adopt the following notation:

- Let $G = GEV_{(\mu,\sigma,\xi)}$ denote a GEV distribution with (unknown) parameters μ, σ, and ξ satisfying the conditions stated in §5.3.1, and
- let m_j be the expected maximum of j samples from G.

Our procedure for estimating m_n is as follows. First, we obtain an accurate estimate of ξ. Then

1. if $\xi \approx 0$ (so that the growth of m_n as a function of $\ln n$ is linear), we estimate m_n by first estimating m_1 and m_2, then performing a linear interpolation;
2. otherwise we estimate m_n by first estimating m_1, m_2, and m_4, then performing a nonlinear interpolation.

Our estimation procedure will take a different form depending on the GEV distribution is estimated to have shape parameter $\xi < 0$, $\xi = 0$, or $\xi > 0$. Although we will analyze all three cases, it is worth noting that the case $\xi < 0$ is really the only one that can arise in practice. This is true because in any real combinatorial optimization problem the maximum payoff is bounded from above, which (by Proposition 3) can only happen when $\xi < 0$.

For the purpose of the proofs presented in this section, we will make an additional minor assumption concerning an arm's shape parameter ξ_i: we assume that for some known constant $\xi^* > 0$,

$$|\xi_i| < \xi^* \Rightarrow \xi_i = 0 \,.$$

Removing this assumption does not fundamentally change the results, but it makes the proofs more complicated.

We will repeatedly make use of the following lemma, which shows how to accurately estimate m_j when j is small (the required number of samples grows linearly with j). The proof is given in Appendix A, and uses a standard probabilistic trick called the "median of means" method.

Lemma 15. *Let j be a positive integer and let $\epsilon > 0$ and $\delta \in (0,1)$ be real numbers. Then $O\left(\ln\left(\frac{1}{\delta}\right)\frac{j}{\epsilon^2}\right)$ draws from G suffice to obtain an estimate $\bar{m}_j$ of m_j such that*

$$\mathbb{P}\left[|\bar{m}_j - m_j| < \epsilon\right] \geq 1 - \delta\,.$$

Our first lemma proves a bound on the number of samples needed to accurately estimate ξ.

Lemma 16. *For real numbers $\epsilon > 0$ and $\delta \in (0,1)$, $O\left(\ln(\frac{1}{\delta})\frac{1}{\epsilon^2}\right)$ draws from G suffice to obtain an estimate $\bar{\xi}$ of ξ such that*

$$\mathbb{P}\left[\left|\bar{\xi} - \xi\right| < \epsilon\right] \geq 1 - \delta\,.$$

Proof (sketch). Using Proposition 3, it is straightforward to check that for any $\xi < 1$,

$$\xi = \log_2\left(\frac{m_4 - m_2}{m_2 - m_1}\right)\,. \tag{5.5}$$

Let $\bar{m}_1$, $\bar{m}_2$, and $\bar{m}_4$ be estimates of m_1, m_2, and m_4, respectively, and let $\bar{\xi}$ be the estimate of ξ obtained by plugging $\bar{m}_1$, $\bar{m}_2$, and $\bar{m}_4$ into (5.5).

It can be shown (see the full proof in Appendix A) that

$$|\bar{\xi} - \xi| = O\left(\sum_{j\in\{1,2,4\}} |\bar{m}_j - m_j|\right)\,.$$

Thus to guarantee $\mathbb{P}\left[\left|\bar{\xi} - \xi\right| < \epsilon\right] \geq 1 - \delta$, it suffices that

$$\mathbb{P}\left[|\bar{m}_j - m_j| \leq \Omega(\epsilon)\right] \geq 1 - \frac{\delta}{3}$$

for all $j \in \{1,2,4\}$. By Lemma 15, this requires $O\left(\ln(\frac{1}{\delta})\frac{1}{\epsilon^2}\right)$ draws from G. □

Having just shown how to accurately estimate ξ, it remains to describe how to estimate m_n given knowledge of ξ. Lemmas 17, 18 and 19 show how to efficiently estimate m_n in the cases $\xi = 0$, $\xi < 0$ and $\xi > 0$, respectively. The case $\xi = 0$ is by far the most straightforward.

Lemma 17. *Assume G has shape parameter $\xi = 0$. Let n be a positive integer and let $\epsilon > 0$ and $\delta \in (0,1)$ be real numbers. Then $O\left(\ln(\frac{1}{\delta})\frac{\ln(n)^2}{\epsilon^2}\right)$ draws from G suffice to obtain an estimate $\bar{m}_n$ of m_n such that*

$$\mathbb{P}\left[|\bar{m}_n - m_n| < \epsilon\right] \geq 1 - \delta\,.$$

Proof. By Proposition 3, $m_j = \mu + \sigma\gamma + \sigma \ln(j)$. Thus

$$m_n = m_1 + (m_2 - m_1)\log_2(n)\,. \tag{5.6}$$

Let $\bar{m}_1$ and $\bar{m}_2$ be estimates of m_1 and m_2, respectively, and let $\bar{m}_n$ be the estimate of m_n obtained by plugging $\bar{m}_1$ and $\bar{m}_2$ into (5.6). Define $\Delta_j = |\bar{m}_j - m_j|$ for $j \in \{1, 2, n\}$. Then

$$\Delta_n \leq (1 + \log_2(n))(\Delta_1 + \Delta_2)\,.$$

Thus to guarantee $\mathbb{P}\left[\Delta_n < \epsilon\right] \geq 1 - \delta$, it suffices that $\mathbb{P}\left[\Delta_j \leq \frac{\epsilon}{2(1+\log_2(n))}\right] \geq 1 - \frac{\delta}{2}$ for all $j \in \{1, 2\}$. By Lemma 15, this requires $O\left(\ln(\frac{1}{\delta})\frac{(\ln n)^2}{\epsilon^2}\right)$ draws from G. □

Lemma 18. *Assume G has shape parameter $\xi \leq -\xi^*$. Let n be a positive integer and let $\epsilon > 0$ and $\delta \in (0, 1)$ be real numbers. Then $O\left(\ln(\frac{1}{\delta})\frac{1}{\epsilon^2}\right)$ draws from G suffice to obtain an estimate $\bar{m}_n$ of m_n such that*

$$\mathbb{P}\left[|\bar{m}_n - m_n| < \epsilon\right] \geq 1 - \delta\,.$$

Proof (sketch). By Proposition 3,

$$m_j = \mu + \frac{\sigma}{\xi}\left(j^{\xi}\Gamma(1 - \xi) - 1\right)\,.$$

Define

$$\begin{aligned}\alpha_1 &= \mu - \sigma\xi^{-1}\\ \alpha_2 &= \sigma\xi^{-1}\Gamma(1 - \xi)\\ \alpha_3 &= 2^{\xi}\end{aligned}$$

so that

$$m_j = \alpha_1 + \alpha_2\alpha_3^{\log_2(j)}\,. \tag{5.7}$$

Plugging in the values $j = 1$, $j = 2$, and $j = 4$ into (5.7) yields a system of three quadratic equations. Solving this system for α_1, α_2, and α_3 yields

$$\begin{aligned}\alpha_1 &= (m_1m_4 - m_2^2)(m_1 - 2m_2 + m_4)^{-1}\\ \alpha_2 &= (-2m_1m_2 + m_1^2 + m_2^2)(m_1 - 2m_2 + m_4)^{-1}\\ \alpha_3 &= (m_4 - m_2)(m_2 - m_1)^{-1}\,.\end{aligned}$$

Let $\bar{m}_1$, $\bar{m}_2$, and $\bar{m}_4$ be estimates of m_1, m_2, and m_4, respectively. Plugging $\bar{m}_1$, $\bar{m}_2$, and $\bar{m}_4$ into the above equations yields estimates, say $\bar{\alpha}_1$, $\bar{\alpha}_2$, and $\bar{\alpha}_3$, of α_1, α_2, and α_3, respectively.

With no small amount of algebraic effort, it can be shown (see the full proof in Appendix A) that

$$|\bar{m}_n - m_n| = O\left(\sum_{i\in\{1,2,4\}} |\bar{m}_j - m_j|\right) .$$

Thus to guarantee $\mathbb{P}\left[|\bar{m}_n - m_n| < \epsilon\right] \geq 1 - \delta$, it suffices that

$$\mathbb{P}\left[|\bar{m}_j - m_j| < \Omega(\epsilon)\right] \geq 1 - \frac{\delta}{3}$$

for all $j \in \{1, 2, 4\}$. By Lemma 15, this requires $O(\ln(\frac{1}{\delta})\frac{1}{\epsilon^2})$ draws from G. □

Lemma 19. *Assume G has shape parameter $\xi \geq \xi^*$. Let n be a positive integer and let $\epsilon > 0$ and $\delta \in (0, 1)$ be real numbers. Then $O\left(\ln(\frac{1}{\delta})\frac{\ln(n)^2}{\epsilon^2}\right)$ draws from G suffice to obtain an estimate $\bar{m}_n$ of m_n such that*

$$\mathbb{P}\left[\frac{1}{1+\epsilon} < \frac{\bar{m}_n - \alpha_1}{m_n - \alpha_1} < (1+\epsilon)\right] \geq 1 - \delta$$

where $\alpha_1 = \mu - \frac{\sigma}{\xi}$.

Proof. See Appendix A. □

Putting the results of lemmas 16, 17, 18, and 19 together, we obtain the following theorem.

Theorem 32. *Let n be a positive integer and let $\epsilon > 0$ and $\delta \in (0, 1)$ be real numbers. Then $O\left(\ln(\frac{1}{\delta})\frac{\ln(n)^2}{\epsilon^2}\right)$ draws from G suffice to obtain an estimate $\bar{m}_n$ of m_n such that with probability at least $1 - \delta$, one of the following holds:*

- *$\xi \leq 0$ and $|\bar{m}_n - m_n| < \epsilon$, or*
- *$\xi > 0$ and $\frac{1}{1+\epsilon} < \frac{\bar{m}_n - \alpha_1}{m_n - \alpha_1} < 1 + \epsilon$, where $\alpha_1 = \mu - \frac{\sigma}{\xi}$.*

Proof. First, invoke Lemma 16 with parameters $\frac{\xi^*}{3}$ and $\frac{\delta}{2}$. Then invoke one of Lemmas 17, 18, or 19 (depending on the estimate $\bar{\xi}$ obtained from Lemma 16) with parameters ϵ and $\frac{\delta}{2}$. □

Theorem 32 shows that step 1 (a) of strategy $\mathcal{S}^{GEV}$ can be performed as described, completing our analysis of $\mathcal{S}^{GEV}$.

5.4 Experimental Evaluation

Following Cicirello and Smith [19, 21], we evaluate our algorithm for the max k-armed bandit problem by using it to select among randomized priority dispatching rules for the resource-constrained project scheduling problem with maximal time lags (RCPSP/max). Cicirello and Smith's work showed that a max k-armed bandit approach yields good performance on benchmark instances of this problem.

Briefly, in the RCPSP/max one must assign start times to each of a number of activities in such a way that certain temporal and resource constraints are satisfied. Such an assignment of start times is called a *feasible schedule*. The goal is to find a feasible schedule whose makespan is as small as possible, where the makespan of a schedule is the maximum completion time of any activity.

Even without maximal time lags (which make the problem more difficult), the resource-constrained project scheduling problem is NP-hard and is "one of the most intractable problems in operations research" [63]. When maximal time lags are included, even the feasibility problem (i.e., deciding whether a feasible schedule exists) is NP-hard.

Our experimental evaluation focuses on Threshold Ascent. In these experiments, we found that the number of trials is small enough that $\mathcal{S}^{GEV}$ never makes it past the initial exploration phase, and thus performs similarly to round-robin sampling.

5.4.1 Experimental setup

In this section we define the RCPSP/max and discuss the heuristics and benchmark instances used in our experiments.

The RCPSP/max

Formally, an instance of the RCPSP/max is a tuple $\mathcal{I} = (\mathcal{A}, R, \mathcal{T})$, where $\mathcal{A}$ is a set of activities, R is a vector of resource capacities, and $\mathcal{T}$ is a list of temporal constraints. Each activity $a_i \in \mathcal{A}$ has a *processing time* p_i, and a resource demand $r_{i,k}$ for each $k \in \{1, 2, \ldots, |R|\}$. Each temporal constraint $T \in \mathcal{T}$ is a triple $T = (i, j, \delta)$, where i and j are activity indices and δ is an integer. The constraint $T = (i, j, \delta)$ indicates that activity a_j cannot start until δ time units after activity a_i has started.

A schedule S assigns a *start time* $S(a)$ to each activity $a \in \mathcal{A}$. S is feasible if

$$S(a_j) - S(a_i) \geq \delta \quad \forall (i, j, \delta) \in \mathcal{T}$$

(i.e., all temporal constraints are satisfied), and

$$\sum_{a_i \in A(S,t)} r_{i,k} \leq R_k \quad \forall\, t \geq 0, k \in \{1, 2, \ldots, |R|\}$$

where $A(S,t) = \{a_i \in \mathcal{A} \mid S(a_i) \leq t < S(a_i) + p_i\}$ the set of activities that are in progress at time t. The latter equation ensures that no resource capacity is ever exceeded.

Randomized priority dispatching rules

A priority dispatching rule for the RCPSP/max is a procedure that assigns start times to activities one at a time, in a greedy fashion. The order in which start times are assigned is determined by a rule that assigns priorities to each activity. As noted above, it is NP-hard to generate a feasible schedule for the RCPSP/max, and a simple priority rule will often fail to find a feasible schedule in practice. Priority dispatching rules are therefore augmented to perform a limited amount of backtracking in order to increase the odds of producing a feasible schedule. For more details, see [66].

Cicirello and Smith describe experiments with randomized priority dispatching rules, in which the next activity to schedule is chosen from a probability distribution, with the probability assigned to an activity being proportional to its priority. Cicirello and Smith consider the five randomized priority dispatching rules in the set

$$\mathcal{H} = \{\texttt{LPF}, \texttt{LST}, \texttt{MST}, \texttt{MTS}, \texttt{RSM}\}\ .$$

See Cicirello and Smith [19, 21] for a description of these heuristics. We use the same five heuristics as Cicirello and Smith, with two modifications. First, we added a form of conflict-driven backtracking to the procedure of [66] in order to increase the odds of generating a feasible schedule. We found that this modification improved performance in practice. Second, we modified the RSM heuristic to improve its performance.

Instances

We evaluate our approach on a set of 169 RCPSP/max instances from the ProGen/max library [74]. These instances were selected as follows. We first ran the heuristic `LPF` (the heuristic identified by Cicirello and Smith as having the best performance) 10,000 times on all 540 instances from the `TESTSETC` data set of the ProGen/max library. For many of these instances, `LPF` found a (provably) optimal schedule on a large proportion of the runs. We considered any instance in which the best makespan found by `LPF` was found

with frequency > 0.01 to be "easy" and discarded it from the data set. What remained was a set of 169 "hard" RCPSP/max instances.

For each "hard" RCPSP/max instance, we ran each heuristic $h \in \mathcal{H}$ 10,000 times, storing the results in a file. Using this data, we created a set $\mathcal{K}$ of 169 five-armed bandit problems (each of the five heuristics $h \in \mathcal{H}$ represents an arm). After the data were collected, makespans were converted to payoffs by multiplying each makespan by -1 and scaling them to lie in the interval $[0, 1]$.

5.4.2 Payoff distributions in the RCPSP/max

To better understand the potential advantages and disadvantages of approximating payoff distributions by GEV distributions, we examined the payoff distributions generated by randomized priority dispatching rules for the RCPSP/max. For a number of instances, we plotted the payoff distribution functions for each heuristic $h \in \mathcal{H}$. For each distribution, we fitted a GEV to the empirical data using maximum likelihood estimation of the parameters μ, σ, and ξ, as recommended by Coles [23].

Our experience was that the GEV sometimes provides a good fit to the empirical cumulative distribution function but sometimes provides a very poor fit. Figure 2 shows the empirical distribution and the GEV fit to the payoff distribution of `LPF` on instances `PSP129` and `PSP121`. For the instance `PSP129`, the GEV accurately models the entire distribution, including the right tail. For the instance `PSP121`, however, the GEV fit severely overestimates the probability mass in the right tail. Indeed, the distribution in Figure 2 (B) is so erratic that no parametric family of distributions can be expected to be a good model of its behavior. In such cases a distribution-free approach is preferable.

5.4.3 An illustrative run

Before presenting our results, we illustrate the typical behavior of Threshold Ascent by showing how it performs on the instance `PSP124`. For this and all subsequent experiments, we run Threshold Ascent with parameters $n = 10,000$, $s = 100$, and $\delta = 0.01$.

Figure 3 (A) depicts the payoff distributions for each of the five arms. As can be seen, `LPF` has the best performance on `PSP124`. MST has zero probability of generating a payoff > 0.8, while `LST` and `RMS` have zero probability of generating a payoff > 0.9. `MTS` gives competitive performance up to a payoff of $t \approx 0.9$, at which point the probability of obtaining a payoff $> t$ suddenly drops to zero.

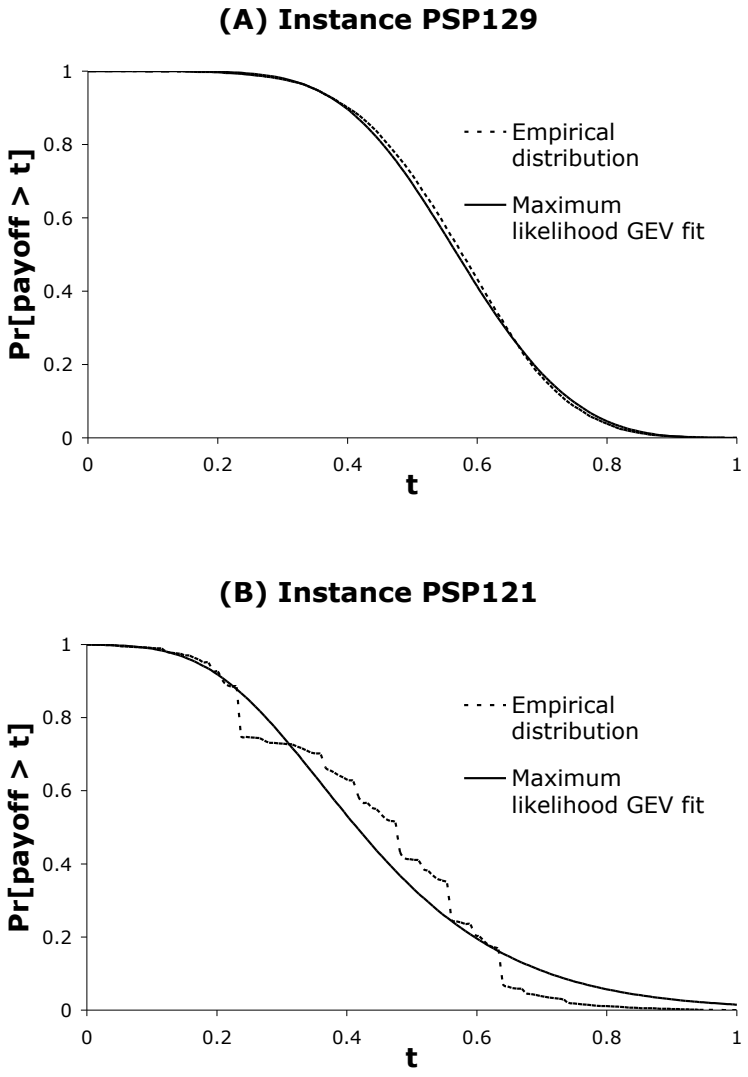

Figure 5.3: Empirical cumulative distribution function of the `LPF` heuristic for two RCPSP/max instances. (A) depicts an instance for which the GEV provides a good fit; (B) depicts an instance for which the GEV provides a poor fit.

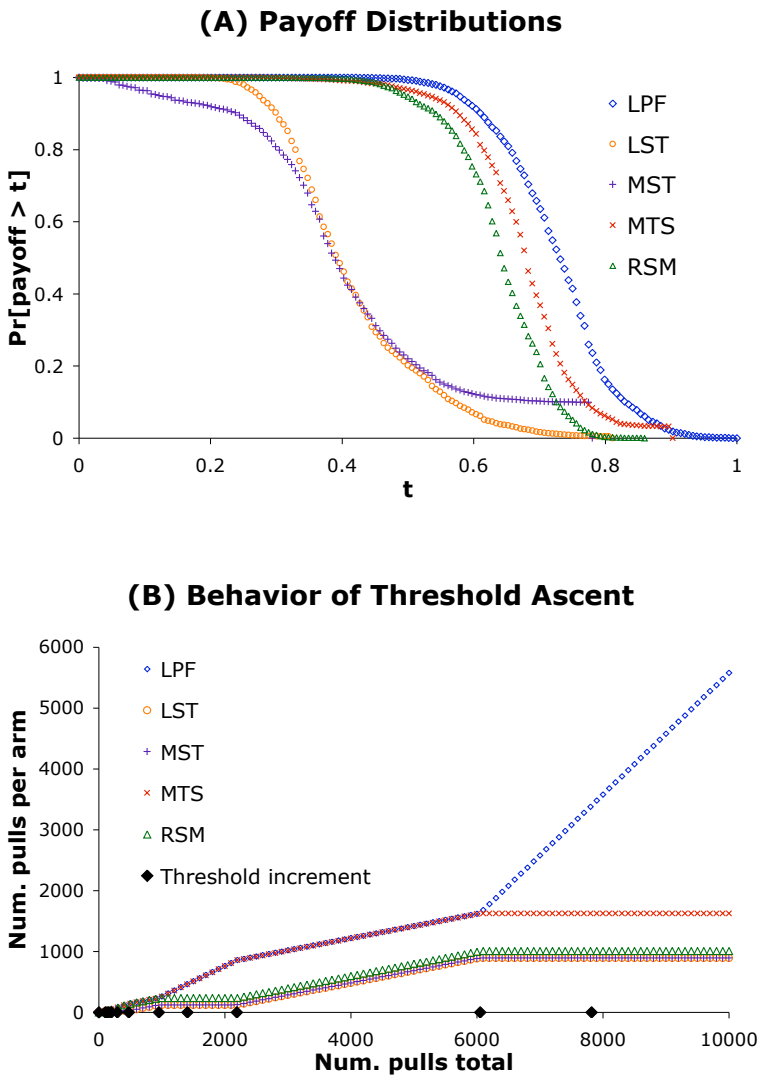

Figure 5.4: Behavior of Threshold Ascent on instance `PSP124`. (A) shows the payoff distributions; (B) shows the number of pulls allocated to each arm.

Table 5.1: Performance of eight max k-armed bandit strategies on 169 RCPSP/max instances.

Strategy	Σ **Regret**	$\mathbb{P}\left[\text{Regret} = 0\right]$	**Num. Feasible**
Threshold Ascent	188	0.722	166
Round-robin sampling	345	0.556	166
`LPF`	355	0.675	164
`MTS`	402	0.657	166
QD-BEACON	609	0.538	165
`RSM`	2130	0.166	155
`LST`	3199	0.095	164
`MST`	4509	0.107	164

Figure 3 (B) shows the number of pulls allocated by Threshold Ascent to each of the five arms as a function of the number of pulls performed so far. As can be seen, Threshold Ascent is a somewhat conservative strategy, allocating a fair number of pulls to heuristics that might seem "obviously" suboptimal to a human observer. Nevertheless, Threshold Ascent spends the majority of its time sampling the single best heuristic (`LPF`).

5.4.4 Results

For each instance $K \in \mathcal{K}$, we ran three max k-armed bandit algorithms, each with a budget of $n = 10,000$ pulls: Threshold Ascent with parameters $n = 10,000$, $s = 100$, and $\delta = 0.01$, the QD-BEACON algorithm of Cicirello and Smith [21], and an algorithm that simply sampled the arms in a round-robin fashion. Cicirello and Smith describe three versions of QD-BEACON; we use the one based on the GEV distribution. For each instance $K \in \mathcal{K}$, we define the *regret* of an algorithm as the difference between the minimum makespan (which corresponds to the maximum payoff) sampled by the algorithm and the minimum makespan sampled by any of the five heuristics (on any of the $10,000$ stored runs of each of the five heuristics). For each of the three algorithms, we also recorded the number of instances for which the algorithm generated a feasible schedule. Table 1 summarizes the performance of these three algorithms, as well as the performance of each of the five heuristics in isolation.

Of the eight max k-armed bandit strategies we evaluated (Threshold Ascent, QD-BEACON, round-robin sampling, and the five pure strategies), Threshold Ascent has the

least regret and achieves zero regret on the largest number of instances. Additionally, Threshold Ascent generated a feasible schedule for the 166 (out of 169) instances for which any of the five heuristics was able to generate a feasible schedule (for three instances, none of the five randomized priority rules generated a feasible schedule after 10,000 runs).

5.4.5 Discussion

Two of the findings summarized in Table 1 may be surprising: the fact that round-robin sampling performs better than any single heuristic, and the fact that QD-BEACON performs worse than round-robin. We now examine each of these findings in more detail.

Why Round-Robin Sampling Performs Well

In the classical k-armed bandit problem, round-robin sampling can never outperform the best pure strategy (where a pure strategy is one that samples the same arm the entire time), either on a single instance or across multiple instances. In the max k-armed bandit problem, however, the situation is different, as the following example illustrates.

Example 5. Suppose we have 2 heuristics, and we run them each for n trials on a set of m instances. On half the instances, heuristic A returns payoff 0 with probability 0.9 and returns payoff 1 with probability 0.1, while heuristic B returns payoff 0 with probability 1. On the other half of the instances, the roles of heuristics A and B are reversed.

If n is large, round-robin sampling will yield total regret ≈ 0, while either of the two heuristics will have regret $\approx \frac{1}{2}m$. By allocating pulls equally to each arm, round-robin sampling is guaranteed to sample the best heuristic at least $\frac{n}{k}$ times, and if n is large this number of samples may be enough to exploit the tail behavior of the best heuristic.

Understanding QD-BEACON

QD-BEACON is designed to converge to a single arm at a doubly-exponential rate. That is, the number of pulls allocated to the (presumed) optimal arm increases doubly-exponentially relative to the number of pulls allocated to presumed suboptimal arms. In our experience, QD-BEACON usually converges to a single arm after at most 10-20 pulls from each arm. This rapid convergence can lead to large regret if the presumed best arm is actually suboptimal.

5.5 Conclusions

The max k-armed bandit problem is a variant of the classical k-armed bandit problem with practical applications to combinatorial optimization.

We presented an algorithm, Chernoff Interval Estimation, for solving the classical k-armed bandit problem, and proved that it has good performance guarantees when the mean payoff returned by each arm is small relative to the maximum possible payoff. Building on Chernoff Interval Estimation we presented an algorithm, Threshold Ascent, that solves the max k-armed bandit problem without making strong assumptions about the payoff distributions. We demonstrated the effectiveness of Threshold Ascent experimentally on the problem of selecting among randomized priority dispatching rules for the RCPSP/max.

Motivated by extreme value theory, we then studied a restricted version of this problem in which each arm yields payoff drawn from a GEV distribution. We derived bounds on the number of samples required to accurately estimate the expected maximum of n draws from a GEV distribution. Using these bounds, we showed that a simple algorithm for the max k-armed bandit problem is asymptotically optimal. Ours is the first algorithm for this problem with rigorous asymptotic performance guarantees.

Chapter 6

Conclusions

In this thesis, we developed techniques for solving hard computational problems more efficiently. Toward this end, we introduced several new online optimization problems, and developed online algorithms for solving these problems. Our online algorithms come with rigorous performance guarantees, stated either as regret bounds or as a competitive ratio. Experimentally, we showed that these techniques can be used to improve the performance of state-of-the-art algorithms in a wide variety of problem domains.

Interpreted narrowly, the contributions of this thesis consist of new theoretical and experimental results for three previously-studied problems: *algorithm portfolio design*, *using decision procedures efficiently for optimization*, and *the max k-armed bandit problem*. In each case, our results provide new ways to improve the performance of certain classes of algorithms.

Interpreted more broadly, this thesis presents three successful examples of a high-level strategy for solving NP-hard problems: namely, to leverage the power of existing heuristics in a principled way. This strategy seems under-exploited at the moment, and we hope our work will encourage more people to pursue it.

Appendix A

Additional Proofs

A.1 Online Algorithms for Maximizing Submodular Functions

Theorem 5. $c\left(f, G'\right) \leq 4 \int_{t=0}^{\infty} 1 - \max_{S \in \mathcal{S}} \left\{ f\left(S_{\langle t \rangle}\right) \right\} dt \leq 4 \min_{S \in \mathcal{S}} \left\{ c\left(f, S\right) \right\}$.

Proof. Recall that $G' = \langle g'_1, g'_2, \ldots \rangle$, where $G'_j = \langle g'_1, g'_2, \ldots, g'_{j-1} \rangle$ and

$$g'_j = \operatorname*{arg\,max}_{(v,\tau) \in \mathcal{V} \times \mathbb{R}_{>0}} \frac{f\left(G'_j \oplus (v,\tau)\right) - f\left(G'_j\right)}{\int_{t'=0}^{\tau} 1 - f\left(G'_j + (v,t')\right) dt'} \,. \tag{A.1}$$

Let s'_j equal the j^{th} value of the $\arg\max$ in (A.1), multiplied by the quantity $1 - f(G'_j)$. We will make use of the following claim.

Claim 1. For any schedule S, any positive integer j, and any $t \geq 0$, $f\left(S_{\langle t \rangle}\right) \leq f\left(G'_j\right) + t s'_j$.

Proof. Fix an action $a = (v, \tau)$. By monotonicity of f, we have $\int_{t'=0}^{\tau} 1 - f\left(G'_j \oplus \langle (v,\tau) \rangle\right) dt' \leq \tau(1 - f\left(G'_j\right))$, or equivalently,

$$\frac{1}{\tau} \leq \frac{1 - f\left(G'_j\right)}{\int_{t'=0}^{\tau} 1 - f\left(G'_j + (v,\tau)\right) dt'} \,.$$

This and the definition of s'_j imply

$$\frac{f\left(G'_j \oplus \langle a \rangle\right) - f\left(G'_j\right)}{\tau} \leq \left(1 - f\left(G'_j\right)\right) \cdot \frac{f\left(G'_j \oplus \langle a \rangle\right) - f\left(G'_j\right)}{\int_{t'=0}^{\tau} 1 - f\left(G'_j \oplus \langle (v,t') \rangle\right) dt'} \leq s'_j \,.$$

The claim then follows by exactly the same argument that was used to prove Fact 1. □

The remainder of the proof parallels the proof of Theorem 4. Using Claim 1 and the argument in the proof of Theorem 4, we get that

$$\int_{t=0}^{\infty} 1 - \max_{S \in \mathcal{S}} \left\{ f\left(S_{\langle t \rangle}\right) \right\} \, dt \geq \sum_{j \geq 1} x_j (y_j - y_{j+1})$$

where $x_j = \frac{R_j}{2s'_j}$, $y_j = \frac{R_j}{2}$, and $R_j = 1 - f\left(G'_j\right)$. Letting $g'_j = (v_j, \tau_j)$, we have

$$\sum_{j \geq 1} x_j (y_j - y_{j+1}) = \frac{1}{4} \sum_{j \geq 1} \int_{t'=0}^{\tau_j} 1 - f\left(G'_j \oplus \langle (v_j, t') \rangle\right) \, dt' = \frac{1}{4} c\left(f, G'\right) \, .$$

which proves the theorem. □

We now prove the theorems concerning the performance of the greedy schedule $\bar{G}$, in which the j^{th} evaluation of the $\arg\max$ in (2.4) is performed with additive error ϵ_j. To ease notation, let $\bar{G} = \langle g_1, g_2, \ldots \rangle$, where $g_j = (v_j, \tau_j)$. Let $s_j = \frac{f(\bar{G}_{j+1}) - f(\bar{G}_j)}{\tau_j}$. To prove Theorems 6 and 7, we will make use of the following fact, which can be proved in exactly the same way as Fact 1.

Fact 4. *For any schedule S, any positive integer j, and any $t > 0$, we have $f(S_{\langle t \rangle}) \leq f(\bar{G}_j) + t \cdot (s_j + \epsilon_j)$.*

Theorem 6. *Let L be a positive integer, and let $T = \sum_{j=1}^{L} \tau_j$, where $g_j = (v_j, \tau_j)$. Then*

$$f\left(\bar{G}_{\langle T \rangle}\right) > \left(1 - \frac{1}{e}\right) \max_{S \in \mathcal{S}} \left\{ f\left(S_{\langle T \rangle}\right) \right\} - \sum_{j=1}^{L} \epsilon_j \tau_j \, .$$

Proof. Let $C^* = \max_{S \in \mathcal{S}} \left\{ f\left(S_{\langle T \rangle}\right) \right\}$, and for any positive integer j, let $\Delta_j = C^* - f\left(G_j\right)$. By Fact 4, $C^* \leq f\left(\bar{G}_j\right) + T(s_j + \epsilon_j)$. Thus

$$\Delta_j \leq T(s_j + \epsilon_j) = T\left(\frac{\Delta_j - \Delta_{j+1}}{\tau_j} + \epsilon_j\right) \, .$$

Rearranging this inequality gives $\Delta_{j+1} \leq \Delta_j \left(1 - \frac{\tau_j}{T}\right) + \tau_j \epsilon_j$. Unrolling this inequality (and using the fact that $1 - \frac{\tau_j}{T} < 1$ for all j), we get

$$\Delta_{L+1} \leq \Delta_1 \left(\prod_{j=1}^{L} 1 - \frac{\tau_j}{T} \right) + \sum_{j=1}^{L} \tau_j \epsilon_j \, .$$

Let $E = \sum_{j=1}^{L} \tau_j \epsilon_j$. Subject to the constraint $\sum_{j=1}^{L} \tau_j = T$, the product series is maximized when $\tau_j = \frac{T}{L}$ for all j. Thus we have

$$C^* - f\left(\bar{G}_{L+1}\right) = \Delta_{L+1} \leq \Delta_1 \left(1 - \frac{1}{L}\right)^L + E < \Delta_1 \frac{1}{e} + E \leq C^* \frac{1}{e} + E \, .$$

Thus $f\left(\bar{G}_{L+1}\right) > (1 - \frac{1}{e})C^* - E$, as claimed. □

Theorem 7. *Let L be a positive integer, and let $T = \sum_{j=1}^{L} \tau_j$, where $g_j = (v_j, \tau_j)$. For any schedule S, define $c^T(f, S) \equiv \int_{t=0}^{T} 1 - f\left(S_{\langle t \rangle}\right) dt$. Then*

$$c^T\left(f, \bar{G}\right) \leq 4 \int_{t=0}^{\infty} 1 - \max_{S \in \mathcal{S}} \left\{ f\left(S_{\langle t \rangle}\right) \right\} dt + \sum_{j=1}^{L} E_j \tau_j \, .$$

where $E_j = \sum_{l<j} \epsilon_l \tau_l$.

Proof. Let $R_j = 1 - f(G_j)$, let $R'_j = R_j - E_j$. Assume for the moment that $R_L \geq E_L$, so that R'_j is non-negative for $j \leq L$. Let $s'_j = s_j + \epsilon_j$. By construction,

$$R'_j - R'_{j+1} = f\left(\bar{G}_{j+1}\right) - f\left(\bar{G}_j\right) + \epsilon_j \tau_j = \tau_j s'_j \, . \tag{A.2}$$

Let $x_j = \frac{R'_j}{2s'_j}$; let $y_j = \frac{R'_j}{2}$; and let $h(x) = 1 - \max_S \left\{ f\left(S_{\langle x \rangle}\right) \right\}$. By Fact 4,

$$\max_S \left\{ f(S_{\langle x_j \rangle}) \right\} \leq f(G_j) + x_j s'_j = f(G_j) + \frac{R'_j}{2} \, .$$

Thus $h(x_j) \geq R_j - \frac{R'_j}{2} = \frac{R_j + E_j}{2} \geq y_j$. The monotonicity of f implies that $h(x)$ is non-increasing and (together with the fact that E_j is non-decreasing as a function of j) implies

that the sequence $\langle y_1, y_2, \ldots \rangle$ is non-increasing. As illustrated in Figure 2.1, these facts imply that $\int_{x=0}^{\infty} h(x)\, dx \geq \sum_{j=1}^{L} x_j \left(y_j - y_{j+1}\right)$. Thus we have

$$\begin{aligned}
\int_{t=0}^{\infty} 1 - \max_{S \in \mathcal{S}} \left\{ f\left(S_{\langle t \rangle}\right) \right\} dt &= \int_{x=0}^{\infty} h(x)\, dx \\
&\geq \sum_{j=1}^{L} x_j \left(y_j - y_{j+1}\right) && \text{(Figure 2.1)} \\
&= \frac{1}{4} \sum_{j=1}^{L} R'_j \frac{\left(R'_j - R'_{j+1}\right)}{s'_j} \\
&= \frac{1}{4} \sum_{j=1}^{L} R'_j \tau_j && \text{(equation (A.2))} \\
&= \frac{1}{4} \left(\sum_{j=1}^{L} R_j \tau_j - \sum_{j \geq 1} E_j \tau_j \right) \\
&\geq \frac{1}{4} c^T\left(f, G\right) - \frac{1}{4} \sum_{j=1}^{L} E_j \tau_j && \text{(monotonicity of } f\text{)}
\end{aligned}$$

which proves the theorem, subject to the assumption that $R_L \geq E_L$.

Now suppose $R_L < E_L$. Let K be the largest integer such that $R_K \geq E_K$, and let $T_K = \sum_{j=1}^{K} \tau_j$. By the argument just given,

$$c^{T_K}\left(f, G\right) \leq 4 \int_{t=0}^{\infty} 1 - \max_{S \in \mathcal{S}} \left\{ f\left(S_{\langle t \rangle}\right) \right\} dt + \sum_{j=1}^{K} E_j \tau_j \,.$$

Thus to prove the theorem, it suffices to show that $c^T\left(f, G\right) \leq c^{T_K}\left(f, G\right) + \sum_{j=K+1}^{L} E_j \tau_j$. This holds because

$$\begin{aligned}
c^T\left(f, G\right) - c^{T_K}\left(f, G\right) &= \int_{t=T_K}^{T} 1 - f\left(\bar{G}_{\langle t \rangle}\right) dt \\
&\leq (T - T_K)(1 - f(\bar{G}_{\langle T_K \rangle})) \\
&= (T - T_K) R_{K+1} \\
&< (T - T_K) E_{K+1} \\
&\leq \sum_{j=K+1}^{L} E_j \tau_j \,.
\end{aligned}$$

□

Lemma 2. *Any sequence $\langle f_1, f_2, \ldots, f_n \rangle$ of jobs satisfies Condition 2. That is, for any sequence $S_1, S_2, \ldots, S_n$ of schedules and any schedule S,*

$$\frac{\sum_{i=1}^{n} f_i(S_i \oplus S) - f_i(S_i)}{\ell(S)} \leq \max_{(v,\tau)\in\mathcal{V}\times\mathbb{R}_{>0}} \left\{ \frac{\sum_{i=1}^{n} f_i(S_i \oplus \langle (v,\tau) \rangle) - f_i(S_i)}{\tau} \right\} .$$

Proof. Let r denote the right hand side of the inequality. Let $S = \langle a_1, a_2, \ldots, a_L \rangle$, where $a_l = (v_l, \tau_l)$. Let

$$\Delta_{i,l} = f_i(S_i \oplus \langle a_1, a_2, \ldots, a_l \rangle) - f(S_i \oplus \langle a_1, a_2, \ldots, a_{l-1} \rangle) .$$

We have

$$\begin{aligned}
\sum_{i=1}^{n} f_i(S_i \oplus S) &= \sum_{i=1}^{n} \left(f_i(S_i) + \sum_{l=1}^{L} \Delta_{i,l} \right) && \text{(telescoping series)} \\
&\leq \sum_{i=1}^{n} \left(f_i(S_i) + \sum_{l=1}^{L} \left(f_i(S_i \oplus \langle a_l \rangle) - f(S_i) \right) \right) && \text{(submodularity)} \\
&= \sum_{i=1}^{n} f_i(S_i) + \sum_{l=1}^{L} \sum_{i=1}^{n} \left(f_i(S_i \oplus \langle a_l \rangle) - f(S_i) \right) \\
&\leq \sum_{i=1}^{n} f_i(S_i) + \sum_{l=1}^{L} r \cdot \tau_l && \text{(definition of } r\text{)} \\
&= \sum_{i=1}^{n} f_i(S_i) + r \cdot \ell(S) \ .
\end{aligned}$$

Rearranging this inequality gives $\frac{\sum_{i=1}^{n} f_i(S_i \oplus S) - f_i(S_i)}{\ell(S)} \leq r$, as claimed. □

Lemma 4. *Algorithm $\mathbf{OG_{unit}}$ with randomized weighted majority as the subroutine experts algorithm has $\mathbb{E}[R] = O\left(\sqrt{Tn \ln |\mathcal{A}|}\right)$ in the worst case.*

Proof. Let $k = |\mathcal{A}|$. Let x_t be the total payoff received by $\mathcal{E}_t$, and let $g_t = x_t + r_t$ be the total payoff that could have been received by $\mathcal{E}_t$ in hindsight (had it been forced to choose a fixed expert each day). Because $\sum_{t=1}^{T} x_t \leq n$, we have $\sum_{t=1}^{T} g_t \leq n + R$. Using WMR, $\mathbb{E}[r_t] = O\left(\sqrt{g_t \ln k}\right)$. Using WMR, the actual value of r_t will be tightly concentrated about its expectation, as can be shown using Azuma's inequality. In particular, because $g_t \leq n$, the probability that $R > n$ is exponentially small. Assuming $R \leq n$, we have $\sum_{t=1}^{T} g_t \leq 2n$. Subject to this constraint, $\sum_{t=1}^{T} \sqrt{g_t}$ is maximized when $g_t = \frac{2n}{T}$ for all t. Thus in the worst case, $\mathbb{E}[R] = O\left(\sqrt{Tn \ln k}\right)$. □

In order to prove Theorem 10, we first prove the following lemma. The lemma relates the expected cost of the schedule S_i (selected by **OG** on round i) to the expected cost S_i would incur if, hypothetically, each of the "meta-actions" selected by each experts algorithm $\mathcal{E}_t$ consumed unit time on every job (require that this assumption was made in the analysis in the main text).

Lemma 20. *Fix a sequence of jobs $\langle f_1, f_2, \ldots, f_n \rangle$ and an integer i ($1 \leq i \leq n$). Let S_i be the schedule produced by* **OG** *to use on job f_i, and let $S_{i,t-1}$ denote the partial schedule that exists after the first $t-1$ experts algorithms has selected actions. Then*

$$\mathbb{E}\left[c^{\ell(S_i)}(f_i, S_i)\right] \leq \mathbb{E}\left[\sum_{t=1}^{L} (1 - f_i(S_{i,t-1}))\right].$$

Proof. Fix some t. Let $a_t^i = (v, \tau)$ be the action selected by $\mathcal{E}_t$ on round i, and define

$$c_t^i = \begin{cases} \int_{t'=0}^{\tau} 1 - f_i(S_{i,t-1} \oplus \langle (v, t') \rangle)\, dt' & \text{if } a_t^i \text{ is appended to } S_i \\ 0 & \text{otherwise.} \end{cases}$$

By construction, $c^{\ell(S_i)}(f_i, S_i) = \sum_{t=1}^{L} c_t^i$. Because a_t^i is appended to S_i with probability $\frac{1}{\tau}$, and because f_i is monotone, we have

$$\mathbb{E}\left[c_t^i | S_{i,t-1}\right] = \frac{1}{\tau} \int_{t'=0}^{\tau} 1 - f_i(S_{i,t-1} \oplus \langle (v, t') \rangle)\, dt' \leq 1 - f_i(S_{i,t-1}).$$

Taking the expectation of both sides yields $\mathbb{E}[c_t^i] \leq \mathbb{E}[1 - f_i(S_{i,t-1})]$. Then by linearity of expectation,

$$\mathbb{E}\left[c^{\ell(S_i)}(f_i, S_i)\right] = \mathbb{E}\left[\sum_{t=1}^{L} c_t^i\right] \leq \mathbb{E}\left[\sum_{t=1}^{L} (1 - f_i(S_{i,t-1}))\right].$$

□

Theorem 10. *Algorithm* **OG***, run with input* $L = T \ln n$*, has* $\mathbb{E}[R_{cost}] = O(T \ln n \cdot \mathbb{E}[R] + T\sqrt{n})$*. In particular,* $\mathbb{E}[R_{cost}] = O\left((\ln n)^{\frac{3}{2}} T\sqrt{Tn \ln |\mathcal{A}|}\right)$ *if WMR is used as the subroutine experts algorithm.*

Proof. The arguments in the main text showed that **OG** can be viewed as a version of the greedy schedule for the function $f = \frac{1}{n}\sum_{i=1}^{n} f_i$, in which the t^{th} decision is made with additive error ϵ_t, under the assumption that all "meta-actions" $\tilde{a}_t^i$ require unit time on every job. Thus by Theorem 7, we have

$$\sum_{i=1}^{n}\sum_{t=1}^{L}(1 - f_i(S_{i,t-1})) \leq 4 \cdot \min_{S\in\mathcal{S}}\left\{\sum_{i=1}^{n} c(f_i, S)\right\} + nL\sum_{t=1}^{L}\epsilon_t \,. \tag{A.3}$$

Also recall from the main text that $\mathbb{E}[\epsilon_t] = \mathbb{E}\left[\frac{r_t}{n}\right]$, where r_t is the regret experienced by $\mathcal{E}_t$, and that we define $R = \sum_{t=1}^{L} r_t$. Thus, we have

$$\begin{aligned}
\mathbb{E}\left[\sum_{i=1}^{n} c^{\ell(S_i)}(f_i, S_i)\right] &\leq \mathbb{E}\left[\sum_{i=1}^{n}\sum_{t=1}^{L}(1 - f_i(S_{i,t-1}))\right] && \text{(Lemma 20)} \\
&\leq 4 \cdot \min_{S\in\mathcal{S}}\left\{\sum_{i=1}^{n} c(f_i, S)\right\} + L \cdot \mathbb{E}[R] \,. && \text{(equation A.3)}
\end{aligned}$$

If it was always the case that $\ell(S_i) \geq T$, then we would have $c^T(f_i, S_i) \leq c^{\ell(S_i)}(f_i, S_i)$, and this inequality would imply $\mathbb{E}[R_{cost}] \leq L \cdot \mathbb{E}[R]$. In order to bound $\mathbb{E}[R_{cost}]$, we now address the possibility that $\ell(S_i) < T$. Letting $p_i = \mathbb{P}[\ell(S_i) < T]$, we have

$$\begin{aligned}
\mathbb{E}\left[c^T(S_i, f_i)\right] &= (1 - p_i) \cdot \mathbb{E}\left[c^T(S_i, f_i) \,|\ell(S_i) \geq T\right] + p_i \cdot \mathbb{E}\left[c^T(S_i, f_i) \,|\ell(S_i) < T\right] \\
&\leq \mathbb{E}\left[c^{\ell(S_i)}(f_i, S_i)\right] + p_i \cdot T \,.
\end{aligned}$$

Putting these inequalities together yields

$$\mathbb{E}[R_{cost}] \leq L \cdot \mathbb{E}[R] + T\sum_{i=1}^{n} p_i \,. \tag{A.4}$$

We now bound p_i. As already mentioned, $\mathbb{E}[\ell(S_i)] = L$ regardless of which actions are selected by the various experts algorithms. If $L \gg T$, then $\ell(S_i)$ will be sharply

concentrated about its mean, as we can prove using standard concentration inequalities (e.g., Theorem 5 of [18]). In particular, for any $\lambda > 0$, we have

$$\mathbb{P}\left[\ell\left(S_i\right) \leq L - \lambda\right] <= \exp\left(-\frac{\lambda^2}{2LT}\right) .$$

Setting $\lambda = L - T$ and simplifying yields $p_i \leq \exp\left(-\frac{L}{2T} + 1\right)$. Setting $L = T \ln n$ then yields $p_i \leq \frac{e}{\sqrt{n}}$, so the right hand side of (A.4) is $O\left(T\sqrt{n}\right)$. Thus $\mathbb{E}\left[R_{cost}\right] = O\left(T \ln n \cdot \mathbb{E}\left[R\right] + T\sqrt{n}\right)$, as claimed. Substituting the bound on $\mathbb{E}\left[R\right]$ stated in Lemma 4 then proves the claim about WMR. □

Theorem 11. *Algorithm* $\mathbf{OG^P}$*, run with WMR as the subroutine experts algorithm, has* $\mathbb{E}\left[R_{coverage}\right] = O\left((C \ln |\mathcal{A}|)^{\frac{1}{3}}(Tn)^{\frac{2}{3}}\right)$ *(when run with input* $L = T$*) and has* $\mathbb{E}\left[R_{cost}\right] = O\left((T \ln n)^{\frac{5}{3}}(C \ln |\mathcal{A}|)^{\frac{1}{3}}(n)^{\frac{2}{3}}\right)$ *(when run with input* $L = T \ln n$*) in the priced feedback model.*

Proof. Let M be the number of exploration rounds (so $\mathbb{E}\left[M\right] = \gamma n$). The maximum payoff to any single expert cannot exceed M. Thus, by Lemma 5 and the regret bound of WMR, we have $\mathbb{E}\left[r_t | M\right] = O\left(\frac{1}{\gamma}\sqrt{M \ln |\mathcal{A}|}\right)$. Using the fact that $\mathbb{E}\left[\sqrt{X}\right] \leq \sqrt{\mathbb{E}\left[X\right]}$ for any random variable X, this implies

$$\mathbb{E}\left[r_t\right] = \mathbb{E}\left[\mathbb{E}\left[r_t | M\right]\right] = O\left(\frac{1}{\gamma}\sqrt{\mathbb{E}\left[M\right] \ln |\mathcal{A}|}\right) = O\left(\sqrt{\frac{n}{\gamma} \ln |\mathcal{A}|}\right) .$$

By Theorem 9, we have $\mathbb{E}\left[R_{coverage}\right] \leq \mathbb{E}\left[R\right] + C\gamma n = O\left(L\sqrt{\frac{n}{\gamma} \ln |\mathcal{A}|}\right) + C\gamma n$. Setting $\gamma = \left(\frac{L}{C}\sqrt{\frac{\ln |\mathcal{A}|}{n}}\right)^{\frac{2}{3}}$ then yields $\mathbb{E}\left[R_{coverage}\right] = O\left((C \ln |\mathcal{A}|)^{\frac{1}{3}}(Ln)^{\frac{2}{3}}\right)$, as claimed.

Similarly, by Theorem 10, we have $\mathbb{E}\left[R_{cost}\right] \leq L \cdot \mathbb{E}\left[R\right] + T\sqrt{n} + TC\gamma n = L \cdot O\left(L\sqrt{\frac{n}{\gamma} \ln |\mathcal{A}|} + C\gamma n\right)$, so the same setting of γ yields $\mathbb{E}\left[R_{cost}\right] = O\left(L^{\frac{5}{3}}(C \ln |\mathcal{A}|)^{\frac{1}{3}} n^{\frac{2}{3}}\right)$. □

Theorem 13. *Algorithm* $\mathbf{OG^O}$*, run with WMR as the subroutine experts algorithm, has* $\mathbb{E}\left[R_{coverage}\right] = O\left(T(|\mathcal{A}| \ln |\mathcal{A}|)^{\frac{1}{3}} n^{\frac{2}{3}}\right)$ *(when run with input* $L = T$*) and has* $\mathbb{E}\left[R_{cost}\right] = O\left((T \ln n)^2(|\mathcal{A}| \ln |\mathcal{A}|)^{\frac{1}{3}} n^{\frac{2}{3}}\right)$ *(when run with input* $L = T \ln n$*) in the opaque feedback model.*

Proof. We showed in the main text that $\mathbb{E}\left[\hat{x}^i_{t,a}\right] = \frac{\gamma}{L|\mathcal{A}|}x^i_{t,a} + \delta_i$, where $\hat{x}^i_{t,a}$ is the estimated payoff fed back by $\mathbf{OG^o}$ and $x^i_{t,a}$ is the true payoff. Thus by Lemma 5, $\mathbb{E}[r_t]$ is bounded by $\frac{|\mathcal{A}|L}{\gamma}$ times the worst-case regret of $\mathcal{E}_t$. Using the same argument we used in the proof of Theorem 11, we get $\mathbb{E}[R] = O\left(L\sqrt{\frac{n}{\gamma'}\ln|\mathcal{A}|}\right)$, where $\gamma' = \frac{\gamma}{|\mathcal{A}|L}$. By Theorem 9, we have $\mathbb{E}[R_{coverage}] \leq \mathbb{E}[R] + \gamma n = O\left(L\sqrt{\frac{n}{\gamma'}\ln|\mathcal{A}|}\right) + C\gamma' n$, where $C = L|\mathcal{A}|$. As in the proof of Theorem 9, setting $\gamma' = \left(\frac{L}{C}\sqrt{\frac{\ln|\mathcal{A}|}{n}}\right)^{\frac{2}{3}}$ then yields $\mathbb{E}[R_{coverage}] = O\left((C\ln|\mathcal{A}|)^{\frac{1}{3}}(Ln)^{\frac{2}{3}}\right) = O\left(T(|\mathcal{A}|\ln|\mathcal{A}|)^{\frac{1}{3}}n^{\frac{2}{3}}\right)$, and the same setting of γ' yields $\mathbb{E}[R_{cost}] = O\left(L^{\frac{5}{3}}(C\ln|\mathcal{A}|)^{\frac{1}{3}}n^{\frac{2}{3}}\right) = O\left((T\ln n)^2(|\mathcal{A}|\ln|\mathcal{A}|)^{\frac{1}{3}}n^{\frac{2}{3}}\right)$. □

We now prove lower bounds on regret. As mentioned in the main text, our lower bounds will hold for the online versions of MAX k-COVERAGE and MIN-SUM SET COVER.

We consider the following online version of MAX k-COVERAGE. One is given a collection $\mathcal{C}$ of sets, where each set in $\mathcal{C}$ is a subset of a universe $E = \{e_1, e_2, \ldots, e_n\}$. One cannot examine the sets (or even determine their cardinalities) directly. On round i of the game, one must specify a subcollection $C \subset \mathcal{C}$, with $|C| = k$. One then receives a reward of 1 if element e_i belongs to some set in the collection, and receives a reward of zero otherwise. One then learns as feedback which sets e_i belonged to.

This problem is a special case of the online version of BUDGETED MAXIMUM SUBMODULAR COVERAGE. To see this, let $\mathcal{V} = \mathcal{C}$ be the set of activities, and think of the action (v, τ) as including the set v in the collection assuming $\tau \geq 1$, and having no effect otherwise. For any schedule S, let $f_i(S) = 1$ if one of the sets added to the collection by S contains e_i, and let $f_i(S) = 0$ otherwise. Then BUDGETED MAXIMUM SUBMODULAR COVERAGE on the sequence of jobs $\langle f_1, f_2, \ldots, f_n\rangle$, with time limit $T = k$, is exactly the problem just described.

The online version of MIN-SUM SET COVER is similar, except that instead of specifying a subcollection of cardinality k, one specifies a sequence of k sets from $\mathcal{C}$. One then incurs a loss equal to the number of sets one must look through in the sequence in order to find e_i, or a loss of k if e_i does not appear in the sequence at all. By the arguments just given, this is equivalent to online MIN-SUM SUBMODULAR COVER on the sequence of jobs $\langle f_1, f_2, \ldots, f_n\rangle$, where $T = k$ is the time at which schedule costs are truncated.

To prove lower bounds on regret, we will require the following technical lemma. The proof is a straightforward generalization of the proof of Lemma 3.2.1 of [16], which con-

sidered the special case $p = \frac{1}{2}$.

Lemma 21 ([16]). *Let $X_1, X_2, \ldots, X_s$ be s independent random variables, where X_i equals the number of heads in n flips of a coin with bias p. Let $\mu = np$ and let $\sigma = \sqrt{np(1-p)}$. Then*

$$\mathbb{E}\left[\max\left\{X_1, X_2, \ldots, X_s\right\}\right] = \mu + \Omega\left(\sigma\sqrt{\ln s}\right).$$

Theorem 14. *Any algorithm for online* MAX k-COVERAGE *has worst-case expected 1-regret $\Omega\left(\sqrt{Tn\ln\frac{|\mathcal{V}|}{T}}\right)$, where $\mathcal{V}$ is the collection of sets and $T = k$ is the number of sets selected by the online algorithm on each round.*

Proof. Let $\mathcal{V}$ be a collection of sets. On each round of the online game, whether or not a given set covers the element will be determined by flipping a coin of bias $p = \frac{1}{2T}$. Thus, regardless of which T sets are selected by the online algorithm, the probability that it covers the element is $q = 1 - \left(1 - \frac{1}{2T}\right)^T \in [\frac{1}{2}, \frac{1}{\sqrt{e}}]$, and the expected number of elements the online algorithm covers is nq.

We now consider the number of elements that could have been covered in hindsight. Let $R = \sqrt{\frac{n}{T}\ln\frac{|\mathcal{V}|}{T}}$. Partition $\mathcal{V}$ into T bins, each of size $s = \frac{|\mathcal{V}|}{T}$. Let S_i^* denote the set in the i^{th} bin which covers the largest number of elements, and let $C^* = \{S_1^*, S_2^*, \ldots, S_T^*\}$. To prove the theorem, it suffices to show that C^* covers $nq + \Omega(TR)$ elements in expectation.

Let a collection $C = \{S_1, S_2, \ldots, S_T\}$ consist of a *random* set drawn from each bin. In expectation C covers nq elements. Let $x_i := |S_i^*| - |S_i|$ and note that $x_i \geq 0$ and $\mathbb{E}[x_i] = \Omega(R)$ by Lemma 21. Randomly mark x_i elements of S_i^* and let M_i and U_i denote the marked and unmarked elements of S_i^*, respectively. Note that the collection $\{U_i : 1 \leq i \leq T\}$ covers nq elements in expectation. Let X denote the (random) number of additional elements covered by the collection $\{M_i : 1 \leq i \leq T\}$ (i.e., $X = |\cup_i M_i - \cup_i U_i|$). We claim that $\mathbb{E}[X] = \Omega(TR)$. To prove this, define ξ to be the event "for all $S \in \mathcal{C}$, $|S| \leq n/T$" and let Y be the number of marked elements covered exactly once in C^*. We will show that $\mathbb{E}[Y \mid \xi]\,\mathbb{P}[\xi] = \Omega(TR)$. Since $\mathbb{E}[Y \mid \xi] \cdot \mathbb{P}[\xi] \leq \mathbb{E}[Y] \leq \mathbb{E}[X]$, this is sufficient to complete the proof.

Fix i and any element $e \in M_i$. Then $\mathbb{P}[e \text{ uniquely covered} \mid \xi] = \prod_{j\neq i}\left(1 - |S_j^*|/n\right) \geq (1 - 1/T)^{T-1} \geq 1/e$. This implies $\mathbb{E}[Y \mid \xi] \geq \frac{1}{e}\mathbb{E}\left[\sum_i |M_i|\right] = \frac{1}{e}\Omega(TR)$, since, as mentioned, $\mathbb{E}[|M_i|] = \Omega(R)$ for all i. Finally, the Chernoff bound easily yields $\mathbb{P}[\xi] \geq (1 - |\mathcal{V}| \cdot \exp\{-n/8T\}) = 1 - o(1)$, and so $\mathbb{E}[Y \mid \xi] \cdot \mathbb{P}[\xi] = \Omega(TR)$ as claimed. □

The lower bound in Theorem 14 is optimal up to constant factors. To see this, observe that running randomized weighted majority with one expert for each of the $\binom{|\mathcal{V}|}{T}$ possible collections of T sets yields worst-case regret $O\left(\sqrt{n \ln \binom{|\mathcal{V}|}{T}}\right) = O\left(\sqrt{nT \ln \frac{|\mathcal{V}|}{T}}\right)$ for online MAX k-COVERAGE, using the fact that $\binom{|\mathcal{V}|}{T} \leq \left(\frac{|\mathcal{V}|e}{T}\right)^T$. Similarly, using a separate expert for each of the $O\left(|\mathcal{V}|^T\right)$ possible permutations of T sets yields regret $O\left(T\sqrt{Tn \ln |\mathcal{V}|}\right)$ for online MIN-SUM SET COVER, which shows that the lower bound in Theorem 15 is optimal up to logarithmic factors.

Theorem 15. *Any algorithm for online* MIN-SUM SET COVER *has worst-case expected 1-regret* $\Omega\left(T\sqrt{Tn \ln \frac{|\mathcal{V}|}{T}}\right)$*, where* $\mathcal{V}$ *is a collection of sets and* T *is the number of sets selected by the online algorithm on each round.*

Proof. We use the same construction as in the proof of Theorem 14. Define the *coverage time* of a schedule $S_i = \langle S_1^i, S_2^i, \ldots, S_T^i \rangle$ to be the smallest t such that S_t^i covers the i^{th} element, or T if no such t exists. As in the proof of Theorem 14, the probability that the online algorithm covers any particular element is q. Given that the online algorithm covers an element, the expected coverage time is zT for some $z < \frac{1}{2}$. Thus, any online algorithm has expected coverage time $\bar{t} = qzT + (1-q)T$ for each element.

Now consider the schedule $S^* = \langle S_1^*, S_2^*, \ldots, S_T^* \rangle$, where $S_i^* = U_i \cup M_i$ was defined in the proof of Theorem 14, and let the sets be indexed in random order. The schedule $U = \langle U_1, U_2, \ldots, U_T \rangle$ is statistically equivalent to a random schedule, and thus has expected coverage time $\bar{t}$ per element. Using S^* in place of U causes X additional elements to be covered, where $\mathbb{E}[X] = \Omega\left(\sqrt{Tn \ln \frac{|\mathcal{V}|}{T}}\right)$. Because the sets in S^* are ordered randomly, the expected coverage time for each of the X additional elements is at most $\frac{T}{2}$. Thus, the total expected coverage time of S^* is smaller than that of U by at least $\frac{T}{2}\mathbb{E}[X] = \Omega\left(T\sqrt{Tn \ln \frac{|\mathcal{V}|}{T}}\right)$. □

A.2 Combining Multiple Heuristics Online

Lemma 9. *For any schedule* S *and any* $\alpha > 1$*, there exists an* α*-regular schedule* S_α *such that, for any instance* x*,* $\mathbb{E}[T(S_\alpha, x)] \leq \alpha^2 \cdot \mathbb{E}[T(S, x)]$*. In the special case where all heuristics are executed in the suspend-and-resume model,* $\mathbb{E}[T(S_\alpha, x)] \leq \alpha \cdot \mathbb{E}[T(S, x)]$*.*

Proof. For any profile $P = \langle \tau_1, \tau_2, \ldots, \tau_L \rangle$, let $\lceil P \rceil$ be the α-regular profile obtained as follows: we first round each τ_i up to the nearest power of α. Then, we round the number of runs of each length up to the nearest floor of a power of α (i.e., the nearest member of $\{\lfloor \alpha^i \rfloor : i \in \mathbb{Z}\}$). For example if $\alpha = 2$ and $P = \langle 4, 3, 3, 2 \rangle$, then $\lceil P \rceil = \langle 4, 4, 4, 4, 2 \rangle$. Note that the size of $\lceil P \rceil$ is at most α^2 times the size of P (recall that the size of a profile $P = \langle \tau_1, \tau_2, \ldots, \tau_L \rangle$ equals $\sum_{i=1}^{L} \tau_i$).

For any state $Y = \langle P_1, P_2, \ldots, P_k \rangle$, define

$$\lceil Y \rceil = \langle \lceil P_1 \rceil, \lceil P_2 \rceil, \ldots, \lceil P_k \rceil \rangle$$

Again, note that the size of $\lceil Y \rceil$ is at most α^2 times the size of Y.

Fix some schedule S, and consider the set of states $\left\{ \mathcal{Y}(S_{\langle t \rangle}) : t \geq 0 \right\}$. Let $\langle Y_1, Y_2, \ldots \rangle$ be a list of the elements of this set, arranged in increasing order of size. As a simple example, if $|\mathcal{H}| = 2$, $S = \langle (h_1, 3), (h_2, 1) \rangle$, and $\alpha = 2$, then the set contains five elements: $Y_1 = \langle \langle \rangle, \langle \rangle \rangle$, $Y_2 = \langle \langle 1 \rangle, \langle \rangle \rangle$, $Y_3 = \langle \langle 2 \rangle, \langle \rangle \rangle$, $Y_4 = \langle \langle 4 \rangle, \langle \rangle \rangle$, and $Y_5 = \langle \langle 4 \rangle, \langle 1 \rangle \rangle$.

There is a unique α-regular schedule that passes through the sequence of profiles $\langle Y_1, Y_2, \ldots \rangle$. Call this schedule S_α. We claim that for any time t,

$$\mathbb{P}\left[T\left(S, x\right) \leq t\right] \leq \mathbb{P}\left[T\left(S_\alpha, x\right) \leq \alpha^2 t\right] \ . \tag{A.5}$$

This follows from the fact that S_α passes through the profile $\left\lceil \mathcal{Y}(S_{\langle t \rangle}) \right\rceil$, which has size at most $\alpha^2 t$. Thus by time $\alpha^2 t$, S_α has done all the work that S has done at time t and more. The fact that (A.5) holds for all t implies $\mathbb{E}\left[T\left(S_\alpha, x\right)\right] \leq \alpha^2 \cdot \mathbb{E}\left[T\left(S, x\right)\right]$.

In the special case when all heuristics are executed in the suspend-and-resume model, the argument is exactly the same, except that now each profile P of interest contains only a single number. This means that the size of $\lceil P \rceil$ is at most α times the size of P, and thus for any state Y of interest, the size of $\lceil Y \rceil$ is at most α times the size of Y. □

Theorem 21. *Fix an approximation ratio $\alpha > 1$, a budget B, and an error tolerance $\delta > 0$. Then an α^2 approximation to schedule*

$$S^* = \arg\min_{S \in \mathcal{S}_\delta} \sum_{x \in \mathcal{X}} \mathbb{E}\left[\min\left\{Bk, T\left(S, x\right)\right\}\right]$$

may be found by computing a shortest path in a state-space graph $G^{rs}_{\alpha,B} = \langle V, E, S_e \rangle$, where $|V| = O\left(B^{O(k \log_\alpha \log_\alpha B)}\right)$ and $|E| = O\left(\log_\alpha B \, |V|\right)$. The overall running time is $O\left(n(\log_\alpha B) B^{O(k \log_\alpha \log_\alpha B)}\right)$, where $n = |\mathcal{X}|$.

Proof. The graph $G^{rs}_{\alpha,B} = \langle V, E, S_e \rangle$ may be defined inductively as follows. The vertex set contains the empty profile $Y^{\emptyset}$. Let $Y = \langle P_1, P_2, \ldots, P_k \rangle$ be a state in the vertex set. Let P_j be a profile with size $< B$ (assuming one exists). Let r be a power of α between 1 and B, and let n_r be the number of runs of length r in P_j. Let P' be the profile obtained by increasing the number of runs of length r to n'_r, where n'_r is the smallest integer $> n_r$ that is the floor of a power of α (i.e., $n'_r = \lfloor \alpha^i \rfloor$ for some integer i). As an example, if $\alpha = 2$, $P_j = \langle 2, 2, 1, 1 \rangle$, and $r = 2$, then $P' = \langle 2, 2, 2, 2, 1, 1 \rangle$. The vertex set contains the state $Y' = \langle P_1, P_2, \ldots, P_{j-1}, P', P_{j+1}, \ldots P_k \rangle$. The edge set contains the edge $e = \langle Y, Y' \rangle$, where $S_e(e)$ contains $n'_r - n_r$ copies of the action (h_j, r). Finally, if no profile of size $< B$, exists, then there is an edge from Y to v^*, labeled with the empty schedule.

By construction, any α-regular schedule that runs each heuristic for time at most B corresponds to a path from $Y^{\emptyset}$ to v^* in the graph. To make the weight of this path equal the value of the objective function, we must modify the weights in two ways. First, any edge from Y to v^*, where performing the runs in Y does not yield a solution to each instance with probability at least $1 - \delta$, is assigned infinite weight. Thus, only paths corresponding to schedules in $\mathcal{S}_\delta$ have finite weight. Second, the schedules $S_e(e)$ must be truncated appropriately so that every schedule in $\mathcal{S}_{G^{rs}_{\alpha,B}}$ has length Bk. With these modifications, computing a shortest path from $Y^{\emptyset}$ to v^* yields the α-regular schedule in $\mathcal{S}_\delta$ that minimizes the value of the objective function. By Lemma 9, this yields an α^2 approximation to S^* (the fact that running time is truncated at Bk only helps, as far as the proof of that lemma is concerned).

We now bound $|V|$ and $|E|$. Assume for simplicity that B is a power of α. First, note that the number of distinct run lengths that can appear in an α-regular profile equals the number of powers of α between 1 and B, which is $1 + \log_\alpha B$. In an α-regular profile, the number of runs of each particular length is either 0 or a power of α between 1 and B. Thus the number of α-regular profiles of size at most B is at most $(1 + \log_\alpha B)^{2+\log_\alpha B} = B^{O(\log_\alpha \log_\alpha B)}$. Because each vertex is a k-tuple of α-regular profiles of size at most B, it follows that, $|V| \leq B^{O(k \log_\alpha \log_\alpha B)}$. Lastly, because each edge represents increasing the number of runs of length r, where r is a power of α between 1 and B, each vertex has at most $1 + \log_\alpha B$ outgoing edges, so $|E| \leq (1 + \log_\alpha B)\,|V|$.

To complete the proof, it suffices to show that edge weights can be computed in time $O(n)$ per edge. Consider the special case $n = 1$, and let x be the single problem instance in $\mathcal{X}$. Consider an edge $e = \langle Y, Y' \rangle$, and let $S = S_e(e)$. The weight assigned to edge e can be written as

$$w(e, x) = Q(Y) \cdot \int_{t=0}^{\ell(S)} 1 - p_x\left(S_{\langle t \rangle}\right) dt$$

where $Q(Y)$ is the probability that performing the runs in state Y does not yield a solution

to x (this definition of $w(e,x)$ is consistent with equation (3.4)). Suppose the value of $Q(Y)$ is stored at the vertex Y. Also, suppose that the value of $\int_{t=0}^{\ell(S)} 1 - p_x\left(S_{\langle t \rangle}\right)\, dt$ has been precomputed in advance, for all choices of $S = S_e(e)$ (the number of choices is at most $k(1+\log_\alpha B)^2$, so the time required for the precomputation step does not contribute to the overall time complexity). Then the time required to compute $w(e,x)$ is $O(1)$. Given the value of $Q(Y)$, the value of $Q(Y') = Q(Y)(1-p_x(S))$ can also be computed in $O(1)$ time and stored at vertex Y' for future use, assuming that we precompute $p_x(S)$ for all possible choices of S (again, this does not affect overall time complexity). Finally, in the general case $n > 1$, the weights obtained in this way can simply be summed across all n instances. □

Theorem 18. *If $\mathcal{H}$ contains a single (randomized) heuristic and $\mathcal{X} = \{x\}$ contains a single instance, then*

$$\mathbb{E}\left[T\left(G,x\right)\right] = \min_{S\in\mathcal{S}} \mathbb{E}\left[T\left(S,x\right)\right] .$$

Proof. Luby *et al.* [61] proved that, when running a single randomized heuristic on a single problem instance, the optimal schedule is a uniform restart schedule of the form

$$S_\tau = \langle (h,\tau),(h,\tau),(h,\tau),\ldots\rangle .$$

Let $p(\tau)$ denote the probability that performing the action (h,τ) yields a solution to x. $\mathbb{E}\left[T\left(S_\tau,x\right)\right]$ satisfies the recurrence $\mathbb{E}\left[T\left(S_\tau,x\right)\right] = \int_{t'=0}^{\tau} 1-p(t')\,dt' + (1-p(\tau))T_\tau$, or rearranging,

$$\mathbb{E}\left[T\left(S_\tau,x\right)\right] = \frac{1}{p(\tau)}\left(\int_{t'=0}^{\tau} 1-p(t')\,dt'\right) .$$

To complete the proof, we show that $G' = S_{\tau^*}$, where $\tau^* = \arg\min_{\tau>0}\mathbb{E}\left[T\left(S_\tau,x\right)\right]$ (we assume for the moment that τ^* is unique). Inductively, suppose G'_j is of this form, and consider the quantity that is maximized when choosing g'_j. Let $f(S) = p_x(S)$. For any action (h,τ), $f\left(G'_j \oplus \langle (h,\tau)\rangle\right) - f\left(G'_j\right)$ is the probability that action (h,τ) solves the problem after every action in G'_j has failed, which can also be written as $(1-f(G'_j))\cdot p(\tau)$. Thus $g'_j = (h,\tau)$ is selected so as to maximize the quantity

$$\frac{f\left(G'_j \oplus \langle (h,\tau)\rangle\right) - f\left(G'_j\right)}{\int_{t'=0}^{\tau} 1 - f\left(G'_j \oplus \langle (h,t')\rangle\right)\,dt'} = \frac{(1-f(G'_j))\cdot p(\tau)}{(1-f(G'_j))\cdot\int_{t'=0}^{\tau} 1-p(t')\,dt'} = \frac{1}{\mathbb{E}\left[T\left(S_\tau,x\right)\right]} .$$

By definition, this quantity is maximized by setting $\tau = \tau^*$. Thus, we have $G' = S_{\tau^*}$, so G' is optimal as claimed.

Finally, note that even if τ^* is not unique, G' will be a uniform schedule as long as ties are broken in a consistent manner (e.g., in favor of the smallest value of τ), and thus G' will be optimal by the arguments just given. □

Lemma 12. *For any profile $P = \langle \tau_1, \tau_2, \ldots, \tau_K \rangle$ of size $< B$, define $L_i(P) = \{i' : 1 \leq i' \leq B, \frac{B}{i'} > \tau_i\}$. Then the quantity*

$$\bar{q}_h(P) = \prod_{i=1}^{K} \frac{|\{i' \in L_i(P) : T_{i'} > \tau_i\}| - i + 1}{|L_i(P)| - i + 1}$$

is an unbiased estimate of $q_h(P)$ (i.e., $\mathbb{E}[\bar{q}_h(P)] = q_h(P)$).

Proof. Given the trace $\mathcal{T} = \langle T_1, T_2, \ldots, T_B \rangle$, suppose we construct a new trace $\mathcal{T}'$ by randomly permuting the elements of $\mathcal{T}$ using the following procedure (the procedure is well-defined assuming $|L_i(P)| \geq i$ for each i, which follows from the fact that $\frac{B}{i} > \tau_i$ if P has size $< B$):

1. For i from 1 to K:
   - Choose l_i uniformly at random from $L_i(P) \setminus \{l_1, l_2, \ldots, l_{i-1}\}$.
2. Set $\mathcal{T}' \leftarrow \{T_{l_1}, T_{l_2}, \ldots T_{l_K}\}$.

Because the indices are arbitrary, $\mathbb{P}[\mathcal{T}' \text{ encloses } P] = \mathbb{P}[\mathcal{T} \text{ encloses } P] = q_h(P)$.

On the other hand, it is not difficult to show that the product series $\bar{q}_h(P)$ equals the conditional probability $\mathbb{P}[\mathcal{T}' \text{ encloses } P \mid \mathcal{T}]$ (by construction, the i^{th} factor in the product series is the probability that $T_{l_i} > \tau_i$, conditioned on the fact that $T_{l_g} > \tau_g$ for all $g < i$).

Thus we have

$$\begin{aligned} \mathbb{E}[\bar{q}_h(P)] &= \mathbb{E}[\mathbb{P}[\mathcal{T}' \text{ encloses } P \mid \mathcal{T}]] \\ &= \mathbb{P}[\mathcal{T}' \text{ encloses } P] \\ &= q_h(P) \,. \end{aligned}$$

□

A.3 The Max k-Armed Bandit Problem

In the proofs that follow, we make use of the notation introduced in §5.3.2. In particular, we use G to denote a GEV distribution satisfying the conditions described in §5.3.2, and we use m_j to denote the expected maximum of j independent samples from G.

Theorem 31. *Let $I = \langle G_1, G_2, \ldots, G_k \rangle$ be an instance of the max k-armed bandit problem, where $G_i = GEV_{(\mu_i,\sigma_i,\xi_i)}$, and $\xi_i \leq 0$ for all i. Then strategy $\mathcal{S}^{GEV}$, run on instance I with parameters $\epsilon = \sqrt[3]{\frac{k}{n}}$ and $\delta = \frac{1}{kn^2}$, has regret $O\left(\ln(nk)\ln(n)^2\sqrt[3]{\frac{k}{n}}\right)$.*

Proof. Building on the proof sketch given in the main text, it remains only to show that for any arm i,

$$m_n^i - m_{n-t(k-1)}^i = O\left(\frac{tk}{n}\right)$$

To ease notation, let $k' = k - 1$; and let $\mu = \mu_i$, $\sigma = \sigma_i$, and $\xi = \xi_i$ be the parameters of arm i. Suppose $\xi = 0$. Then by Proposition 3, $m_n^i - m_{n-tk'}^i = \sigma\left(\ln(n) - \ln(n - tk')\right)$. Thus for n sufficiently large,

$$\begin{aligned} m_n^i - m_{n-tk'}^i &= \sigma\left(\ln(n) - \ln(n - tk')\right) \\ &= -\sigma \ln\left(\frac{n - tk'}{n}\right) \\ &= -\sigma \ln\left(1 - \frac{tk'}{n}\right) \\ &< 2\sigma\frac{tk'}{n} \\ &= O\left(\frac{tk'}{n}\right) \end{aligned}$$

where on the fourth line we have used the fact that for n sufficiently large, $\frac{tk'}{n} < \frac{1}{2}$, and for $0 < x < \frac{1}{2}$, $\ln(1 - x) > -2x$.

Now suppose $\xi < 0$. By Proposition 3, $m_n^i - m_{n-tk'}^i = \frac{\sigma}{\xi}\Gamma(1-\xi)(n^\xi - (n - tk')^\xi) = O((n - tk')^\xi - n^\xi)$ where we have used the fact that $\frac{\sigma}{\xi}\Gamma(1-\xi)$ is negative and has bounded absolute value. Expanding $(n - tk')^\xi$ in powers of t about $t = 0$ gives

$$(n - tk')^\xi = n^\xi - \xi n^{\xi-1} tk' + O\left((tk')^2 n^{\xi-2}\right) .$$

Because $\xi \leq 0$ and $|\xi|$ is bounded, it follows that $(n - tk')^\xi - n^\xi$ is $O(\frac{tk'}{n})$. □

In order to prove Lemma 15, we first prove the following lemma.

Lemma 22. *For any fixed positive integer j, $O\left(\frac{j}{\epsilon^2}\right)$ draws from G suffice to obtain an estimate $\bar{m}_j$ of m_j such that*

$$\mathbb{P}\left[|\bar{m}_j - m_j| < \epsilon\right] \geq \frac{3}{4}\,.$$

Proof. First consider the special case $j = 1$. Let X denote the sum of t draws from G, for some to-be-specified positive integer t. Then $\mathbb{E}[X] = m_1 t$ and $Var[X] = \tilde{\sigma}^2 t$, where $\tilde{\sigma}$ is the (unknown) standard deviation of G ($\tilde{\sigma}$ is proportional to, but not the same as, the scale parameter σ of the GEV distribution G). We take $\bar{m}_1 = \frac{X}{t}$ as our estimate of m_1. Then

$$\begin{aligned}
\mathbb{P}\left[|\bar{m}_1 - m_1| \geq \epsilon\right] &= \mathbb{P}\left[|t\bar{m}_1 - tm_1| \geq t\epsilon\right] \\
&= \mathbb{P}\left[|X - \mathbb{E}[X]| \geq \frac{\sqrt{t}\epsilon}{\tilde{\sigma}}\sqrt{Var[X]}\right] \\
&\leq \frac{\tilde{\sigma}^2}{t\epsilon^2}
\end{aligned}$$

where the last inequality is Chebyshev's. Thus to guarantee $\mathbb{P}\left[|\bar{m}_1 - m_1| \geq \epsilon\right] \leq \frac{1}{4}$ we must set $t = \frac{4\tilde{\sigma}^2}{\epsilon^2} = O\left(\frac{1}{\epsilon^2}\right)$ (note that due to the assumptions in §5.3.2, $\tilde{\sigma}$ is $O(1)$).

In the general case $j > 1$, we let X be the sum of t block maxima (each the maximum of j independent draws from G). Because the standard deviation of M_j and j itself are both $O(1)$, the lemma follows by exactly the same argument. □

To boost the probability that $|\bar{m}_j - m_j| < \epsilon$ from $\frac{3}{4}$ to $1 - \delta$, we use the "median of means" method.

Lemma 15. *Let j be a positive integer and let $\epsilon > 0$ and $\delta \in (0, 1)$ be real numbers. Then $O\left(\ln\left(\frac{1}{\delta}\right)\frac{j}{\epsilon^2}\right)$ draws from G suffice to obtain an estimate $\bar{m}_j$ of m_j such that*

$$\mathbb{P}\left[|\bar{m}_j - m_j| < \epsilon\right] \geq 1 - \delta\,.$$

Proof. We invoke Lemma 22 r times (for r to be determined), yielding a set

$$E = \left\{\bar{m}_j^{(1)}, \bar{m}_j^{(2)}, \ldots, \bar{m}_j^{(r)}\right\}$$

of estimates of m_j. Let $\bar{m}_j$ be the median element of E. Let $\mathcal{A} = \{\bar{m} \in E : |\bar{m} - m_j| < \epsilon\}$ be the set of "accurate" estimates of m_j; and let $A = |\mathcal{A}|$. Then $|\bar{m}_j - m_j| \geq \epsilon$ implies $A \leq \frac{r}{2}$, while $\mathbb{E}[A] \geq \frac{3}{4}r$. Using Chernoff's inequality, we have

$$\mathbb{P}\left[|\bar{m}_j - m_j| \geq \epsilon\right] \leq \mathbb{P}\left[A \leq \frac{r}{2}\right] \leq \exp\left(-\frac{r}{C}\right)$$

for constant $C > 0$. Thus $r = O\left(\ln(\frac{1}{\delta})\right)$ repetitions suffice to ensure $\mathbb{P}\left[|\bar{m}_j - m_j| > \epsilon\right] \leq \delta$. $\square$

In order to prove Lemma 16, we must first prove the following lemma. Again, we use m_j to denote the expected maximum of j samples from a GEV distribution G satisfying the assumptions described in §5.3.2.

Lemma 23.

$$m_4 - m_2 \geq \tfrac{1}{4}\sigma \textit{ and}$$
$$m_2 - m_1 \geq \tfrac{1}{8}\sigma\,.$$

Proof. If $\xi = 0$, then by Proposition 3, $m_4 - m_2 = m_2 - m_1 = \ln(2)\sigma$ and we are done. Otherwise,

$$m_4 - m_2 = \sigma(2^\xi - 1)\xi^{-1}\Gamma(1-\xi) \text{ and}$$
$$m_2 - m_1 = \sigma(4^\xi - 2^\xi)\xi^{-1}\Gamma(1-\xi)\,.$$

It thus suffices to prove that

$$\min_{\xi < \frac{1}{2}} \left\{ \frac{2^\xi - 1}{\xi}\Gamma(1-\xi) \right\} \geq \frac{1}{4}$$

and

$$\min_{\xi < \frac{1}{2}} \left\{ \frac{4^\xi - 2^\xi}{\xi}\Gamma(1-\xi) \right\} \geq \frac{1}{8}\,.$$

To do so, we first state without proof the following properties of the Γ function:

$$\Gamma(z) \geq \lfloor z \rfloor! \quad \forall z \geq 2$$
$$\Gamma(z) \geq \tfrac{1}{2} \quad \forall z > 0$$

Making the change of variable $y = -\xi$, it suffices to show

$$\min_{y > -\frac{1}{2}} \left\{ \frac{1 - 2^{-y}}{y}\Gamma(1+y) \right\} \geq \frac{1}{4}\text{, and} \tag{A.6}$$

$$\min_{y > -\frac{1}{2}} \left\{ \frac{2^{-y}(1 - 2^{-y})}{y}\Gamma(1+y) \right\} \geq \frac{1}{8}\,. \tag{A.7}$$

(A.6) holds because for $-\frac{1}{2} < y \leq 1$,

$$\frac{1 - 2^{-y}}{y}\Gamma(1+y) \geq \frac{1}{2}\Gamma(1+y) \geq \frac{1}{4}\,,$$

while for $y > 1$,

$$\frac{1-2^{-y}}{y}\Gamma(1+y) \geq \frac{\lfloor y+1 \rfloor !}{2y} \geq \frac{1}{2}\,.$$

Similarly, (A.7) holds because for $-\frac{1}{2} < y \leq 1$,

$$\frac{2^{-y}(1-2^{-y})}{y}\Gamma(1+y) \geq \frac{1}{8}\,,$$

while for $y > 1$,

$$\frac{2^{-y}(1-2^{-y})}{y}\Gamma(1+y) \geq \frac{\lfloor y+1 \rfloor !}{2y(2^y)} \geq \frac{1}{8}\,.$$

□

We are now ready to prove Lemma 16.

Lemma 16. *For real numbers $\epsilon > 0$ and $\delta \in (0,1)$, $O\left(\ln(\frac{1}{\delta})\frac{1}{\epsilon^2}\right)$ draws from G suffice to obtain an estimate $\bar{\xi}$ of ξ such that*

$$\mathbb{P}\left[|\bar{\xi} - \xi| < \epsilon\right] \geq 1 - \delta\,.$$

Proof. In the proof sketch in the main text, we showed that.

$$\xi = \log_2\left(\frac{m_4 - m_2}{m_2 - m_1}\right)\,. \tag{A.8}$$

Let $\bar{m}_1$, $\bar{m}_2$, and $\bar{m}_4$ be estimates of m_1, m_2, and m_4, respectively, and let $\bar{\xi}$ be the estimate of ξ obtained by plugging $\bar{m}_1$, $\bar{m}_2$, and $\bar{m}_4$ into this equation. Define $\Delta_m = \max_{j\in\{1,2,4\}} |\bar{m}_j - m_j|$ and define $\Delta_\xi = |\bar{\xi} - \xi|$. Building on the proof sketch in the main text, it remains only to show that $\Delta_\xi = O\left(\Delta_m\right)$.

In the proof of Theorem 31 we showed that $|\ln(x+\beta) - \ln(x)| \leq 2\frac{\beta}{x}$ for $\beta \leq \frac{x}{2}$. Letting $N = m_4 - m_2$ and $D = m_2 - m_1$, and noting that $\xi = \log_2(N) - \log_2(D) = \frac{1}{\ln 2}(\ln(N) - \ln(D))$, it follows that

$$\Delta_\xi \leq \frac{1}{\ln(2)}\left(\frac{2(2\Delta_m)}{N} + \frac{2(2\Delta_m)}{D}\right)$$

for $\Delta_m < \frac{1}{2}\min(N, D)$. Thus by Lemma 23 and the assumption that $\sigma \geq \sigma_\ell$, Δ_ξ is $O(\Delta_m)$, as claimed. □

Lemma 18. *Assume G has shape parameter $\xi \leq -\xi^*$. Let n be a positive integer and let $\epsilon > 0$ and $\delta \in (0,1)$ be real numbers. Then $O\left(\ln(\frac{1}{\delta})\frac{1}{\epsilon^2}\right)$ draws from G suffice to obtain an estimate $\bar{m}_n$ of m_n such that*

$$\mathbb{P}\left[|\bar{m}_n - m_n| < \epsilon\right] \geq 1 - \delta\,.$$

Proof. In the proof sketch in the main text, we showed that

$$m_j = \alpha_1 + \alpha_2 \alpha_3^{\log_2(j)}$$

where

$$\begin{aligned}
\alpha_1 &= (m_1 m_4 - m_2^2)(m_1 - 2m_2 + m_4)^{-1} \\
\alpha_2 &= (-2m_1 m_2 + m_1^2 + m_2^2)(m_1 - 2m_2 + m_4)^{-1} \\
\alpha_3 &= (m_4 - m_2)(m_2 - m_1)^{-1}\,.
\end{aligned}$$

Let $\bar{m}_1$, $\bar{m}_2$, and $\bar{m}_4$ be estimates of m_1, m_2, and m_4, respectively. Plugging $\bar{m}_1$, $\bar{m}_2$, and $\bar{m}_4$ into the above equations yields estimates, say $\bar{\alpha}_1$, $\bar{\alpha}_2$, and $\bar{\alpha}_3$, of α_1, α_2, and α_3, respectively. Define $\Delta_m = \max_{j\in\{1,2,4\}} |\bar{m}_j - m_j|$ and $\Delta_\alpha = \max_{j\in\{1,2,3\}} |\bar{\alpha}_i - \alpha_i|$. To complete the proof, it remains to show that

$$|\bar{m}_n - m_n| = O(\Delta_m)\,.$$

The argument consists of two parts: in claims 1 through 3 we show that Δ_α is $O(\Delta_m)$, then in Claim 4 we show that $|\bar{m}_n - m_n|$ is $O(\Delta_\alpha)$.

Claim 1. Each of the numerators in the expressions for α_1, α_2, and α_3 has absolute value bounded from above, while each of the denominators has absolute value bounded from below. (The bounds are independent of the unknown parameters of G.)

Proof of claim 1. The numerators will have bounded absolute value as long as m_1, m_2, and m_3 are bounded. Upper bounds on m_1, m_2, and m_3 follow from the restrictions on the parameters μ, σ, and ξ. As for the denominators, by Lemma 23 we have

$$\begin{aligned}
|m_1 - 2m_2 + m_4| &= |(m_2 - m_1)(\alpha_3 - 1)| \\
&\geq \tfrac{1}{8}\sigma_\ell |2^{-\xi^*} - 1|\,.
\end{aligned}$$

□

Claim 2. Let N and D be fixed real numbers, and let β_N and β_D be real numbers with $|\beta_D| < \frac{|D|}{2}$. Then $|\frac{N+\beta_N}{D+\beta_D} - \frac{N}{D}|$ is $O(|\beta_N| + |\beta_D|)$.

Proof of claim 2. First, using the Taylor series expansion of $\frac{N}{D+\beta_D}$,

$$\begin{aligned}\left|\frac{N}{D+\beta_D}-\frac{N}{D}\right| &= \left|\frac{N\beta_D}{D^2}\sum_{i=0}^{\infty}(-1)^{i+1}\left(\frac{\beta_D}{D}\right)^i\right| \\ &\leq \left|\frac{N\beta_D}{D^2(1-\beta_D D^{-1})}\right| \\ &= O\left(|\beta_D|\right) .\end{aligned}$$

Then

$$\begin{aligned}\left|\frac{N+\beta_N}{D+\beta_D}-\frac{N}{D}\right| &\leq \left|\frac{N}{D+\beta_D}-\frac{N}{D}\right| + \left|\frac{\beta_N}{D+\beta_D}\right| \\ &= O\left(|\beta_N|+|\beta_D|\right) .\end{aligned}$$

□

Claim 3. Δ_α is $O(\Delta_m)$.

Proof of claim 3. We show that $|\bar{\alpha}_1-\alpha_1|$ is $O(\Delta_m)$. Similar arguments show that $|\bar{\alpha}_2-\alpha_2|$ and $|\bar{\alpha}_4-\alpha_4|$ are $O(\Delta_m)$, which proves the claim. To see that $|\bar{\alpha}_1-\alpha_1|$ is $O(\Delta_m)$, let $N=m_1m_4-m_2^2$, and let $D=m_1-2m_2+m_4$, so that $\alpha_1=\frac{N}{D}$. Define $\bar{N}$ and $\bar{D}$ in the natural way so that $\bar{\alpha}_1=\frac{\bar{N}}{\bar{D}}$. Because m_1,m_2, and m_3 are all $O(1)$ (by Claim 1), it follows that both $|\bar{N}-N|$ and $|\bar{D}-D|$ are $O(\Delta_m)$. That $|\bar{\alpha}_1-\alpha_1|$ is $O(\Delta_m)$ follows by Claim 2. □

Claim 4. $|\bar{m}_n-m_n|$ is $O\left(\Delta_\alpha\right)$.

Proof of claim 4. Because $\xi_\ell\leq\xi\leq-\xi^*$ it must be that $0<2^{\xi_\ell}<\alpha_3<2^{-\xi^*}<1$. So for Δ_α sufficiently small, $0<\bar{\alpha}_3<1$.

$$\begin{aligned}||\bar{m}_n-m_n| &= \left|\left(\bar{\alpha}_1+\bar{\alpha}_2\bar{\alpha}_3^{\log_2(n)}\right)-\left(\alpha_1+\alpha_2\alpha_3^{\log_2(n)}\right)\right| \\ &\leq |\bar{\alpha}_1-\alpha_1|+\left|\bar{\alpha}_2\bar{\alpha}_3^{\log_2(n)}-\bar{\alpha}_2\alpha_3^{\log_2(n)}\right| \\ &\qquad +\left|\bar{\alpha}_2\alpha_3^{\log_2(n)}-\alpha_2\alpha_3^{\log_2(n)}\right| \\ &\leq |\bar{\alpha}_1-\alpha_1|+|\bar{\alpha}_2|\,|\bar{\alpha}_3-\alpha_3|+|\bar{\alpha}_2-\alpha_2| \\ &= O\left(\Delta_\alpha\right)\end{aligned}$$

where on the third line we have used the fact that both α_3 and $\bar{\alpha}_3$ are between 0 and 1, and in the last line we have used the fact that $|\bar{\alpha}_2|$ is $O(1)$. □

□

Lemma 19. *Assume G has shape parameter $\xi \geq \xi^*$. Let n be a positive integer and let $\epsilon > 0$ and $\delta \in (0,1)$ be real numbers. Then $O\left(\ln(\frac{1}{\delta})\frac{\ln(n)^2}{\epsilon^2}\right)$ draws from G suffice to obtain an estimate $\bar{m}_n$ of m_n such that*

$$\mathbb{P}\left[\frac{1}{1+\epsilon} < \frac{\bar{m}_n - \alpha_1}{m_n - \alpha_1} < (1+\epsilon)\right] \geq 1 - \delta$$

where $\alpha_1 = \mu - \frac{\sigma}{\xi}$.

Proof. We use the same estimation procedure as in the proof of Lemma 18. Let α_1, α_2, α_3, Δ_α, and Δ_m be defined as they were in that proof.

The inequality $\frac{1}{1+\epsilon} < \frac{\bar{m}_n - \alpha_1}{m_n - \alpha_1} < 1 + \epsilon$ is the same as $|\ln(\frac{\bar{m}_n - \alpha_1}{m_n - \alpha_1})| < \ln(1+\epsilon)$. For $\epsilon < \frac{1}{2}$, $\ln(1+\epsilon) \geq \frac{7}{8}\epsilon$, so it suffices to guarantee that

$$|\ln(\bar{m}_n - \alpha_1) - \ln(m_n - \alpha_1)| < \frac{7}{8}\epsilon\,.$$

Claim 1. $|\ln(\bar{m}_n - \alpha_1) - \ln(m_n - \alpha_1)|$ is $O(\ln(n)\Delta_\alpha)$.

Proof of claim 1. Because $|\ln(\bar{m}_n - \alpha_1) - \ln(\bar{m}_n - \bar{\alpha}_1)|$ is $O(\Delta_\alpha)$, it suffices to show that $|\ln(\bar{m}_n - \bar{\alpha}_1) - \ln(m_n - \alpha_1)|$ is $O(\ln(n)\Delta_\alpha)$. This is true because

$$\begin{aligned}
\ln(\bar{m}_n - \bar{\alpha}_1) &= \ln\left(\bar{\alpha}_2 \bar{\alpha}_3^{\log_2(n)}\right) \\
&= \log_2(n)\ln(\bar{\alpha}_3) + \ln(\bar{\alpha}_2) \\
&= \log_2(n)\ln(\alpha_3) + \ln(\alpha_2) \pm O\left(\ln(n)\Delta_\alpha\right) \\
&= \ln\left(\alpha_2 \alpha_3^{\log_2(n)}\right) \pm O\left(\ln(n)\Delta_\alpha\right) \\
&= \ln(m_n - \alpha_1) \pm O\left(\ln(n)\Delta_\alpha\right)\ .
\end{aligned}$$

□

Setting $\Delta_\alpha < \Omega\left(\ln(n)^{-1}\epsilon\right)$ then guarantees $|\ln(\bar{m}_n) - \ln(m_n)| < \frac{7}{8}\epsilon$. By Claim 3 of the proof of Lemma 18 (which did not depend on the assumption $\xi < 0$), Δ_α is $O\left(\Delta_m\right)$, so we require $\mathbb{P}\left[\Delta_m < \Omega\left(\ln(n)^{-1}\epsilon\right)\right] \geq 1 - \delta$. Define $\Delta_j = |\bar{m}_j - m_j|$, so that $\Delta_m = \max_{j\in\{1,2,4\}} \Delta_j$. It suffices that $\mathbb{P}\left[\Delta_j < \Omega\left(\ln(n)^{-1}\epsilon\right)\right] \geq 1 - \frac{\delta}{3}$ for $j \in \{1,2,4\}$. By Lemma 15, ensuring this requires $O\left(\ln(\frac{1}{\delta})\frac{\ln(n)^2}{\epsilon^2}\right)$ draws from G.

□

Lastly, the following theorem complements Theorem 31 by describing the behavior of $\mathcal{S}^{GEV}$ when some arms have shape parameter $\xi > 0$.

Theorem 33. *Let $I = \langle G_1, G_2, \ldots, G_k \rangle$ be an instance of the max k-armed bandit problem, where $G_i = GEV_{(\mu_i,\sigma_i,\xi_i)}$, where $\xi_i > 0$ for some i. Let S denote the strategy $\mathcal{S}^{GEV}$, run with parameters $\epsilon = \sqrt[3]{\frac{k}{n}}$ and $\delta = \frac{1}{kn^2}$. Then*

$$\frac{\mathbb{E}\left[M(I,S,n)\right] - \alpha_1}{m_n^* - \alpha_1} = 1 - O\left(\ln(nk)\ln(n)^2\sqrt[3]{\frac{k}{n}}\right)$$

where $m_n^* = \max_{1\leq i\leq k} m_n^i$, and $\alpha_1 = \max_{1\leq i\leq k} \alpha_i$, where $\alpha_i = \mu_i - \frac{\sigma_i}{\xi_i}$.

Proof. For the moment, let us assume that *all* arms have shape parameter $\xi_i > 0$. Let $\mathcal{A}$ be the event (which occurs with probability at least $1 - k\delta$) that all estimates obtained in step 1 (a) satisfy the inequality in Theorem 32.

To ease notation, let $\Delta = \ln(nk)\ln(n)^2\sqrt[3]{\frac{k}{n}}$, and let $\hat{m}_j = m_j^{\hat{i}}$ denote the expected maximum of j draws from the arm $\hat{i}$ selected for exploitation.

Claim 1. To prove the theorem, it suffices to show that $\mathcal{A}$ implies $\frac{m_n^* - \alpha_1}{\hat{m}_{n-tk} - \alpha_1} = 1 + O(\Delta)$.

Proof of claim 1. Because $M(I,S,n) \geq \hat{m}_{n-tk}$ and the event $\mathcal{A}$ occurs with probability at least $1 - k\delta$, it suffices to show that $\mathcal{A}$ implies

$$\frac{(1-\delta k)\hat{m}_{n-tk} - \alpha_1}{m_n^* - \alpha_1} = 1 - O(\Delta)\,.$$

Because $\frac{\delta k(\hat{m}_{n-tk})}{m_n^* - \alpha_1}$ is $O\left(\frac{1}{n^2}\right) = o(\Delta)$, it suffices to show that $\mathcal{A}$ implies

$$\frac{\hat{m}_{n-tk} - \alpha_1}{m_n^* - \alpha_1} = 1 - O(\Delta)\,.$$

This can be rewritten as $m_n^* - \alpha_1 = (\hat{m}_{n-tk} - \alpha_1)\frac{1}{1-O(\Delta)} = (\hat{m}_{n-tk} - \alpha_1)(1 + O(\Delta))$ (we can replace $\frac{1}{1-O(\Delta)}$ with $1 + O(\Delta)$ because for $r < \frac{1}{2}$, $\frac{1}{1-r} = 1 + \frac{r}{1-r} < 1 + 2r$). □

Claim 2. $\frac{\hat{m}_n - \hat{\alpha}_1}{\hat{m}_{n-tk} - \hat{\alpha}_1} = 1 + O(\Delta)$.

Proof of claim 2. Using Proposition 3,

$$\begin{aligned}\ln\left(\frac{\hat{m}_n - \hat{\alpha}_1}{\hat{m}_{n-tk} - \hat{\alpha}_1}\right) &= \ln\left(\frac{n^\xi}{(n-t)^\xi}\right) \\ &= \xi\left(\ln(n) - \ln(n-tk)\right) \\ &= O(\frac{tk}{n}) \\ &= O(\Delta)\,.\end{aligned}$$

The claim follows from the fact that $\exp(\beta) < 1 + \frac{3}{2}\beta$ for $\beta < \frac{1}{2}$, so that $\exp(O(\Delta)) = 1 + O(\Delta)$. □

Claim 3. $\mathcal{A}$ implies that for all i,

$$\frac{\bar{m}_n^i - \alpha_1}{m_n^i - \alpha_1} < 1 + \epsilon\,.$$

Proof of claim 3. By definition, $\alpha_1 = \alpha_1^i - \beta$ for some $\beta \geq 0$. The claim follows from the fact that for positive N and D and $\beta \geq 0$, $\frac{N}{D} < 1 + \epsilon$ implies $\frac{N+\beta}{D+\beta} < 1 + \epsilon$. □

Claim 4. $\mathcal{A}$ implies $\frac{m_n^* - \alpha_1}{\hat{m}_{n-tk} - \alpha_1} = 1 + O(\Delta)$.

Proof of claim 4.

$$\begin{aligned}\frac{m_n^* - \alpha_1}{m_{n-tk}^{\hat{i}} - \alpha_1} &= \frac{m_n^* - \alpha_1}{\bar{m}_n^* - \alpha_1} \cdot \frac{\bar{m}_n^* - \alpha_1}{\bar{m}_n^{\hat{i}} - \alpha_1} \cdot \frac{\bar{m}_n^{\hat{i}} - \alpha_1}{m_n^{\hat{i}} - \alpha_1} \cdot \frac{m_n^{\hat{i}} - \alpha_1}{m_{n-tk}^{\hat{i}} - \alpha_1} \\ &\leq (1+\epsilon) \cdot 1 \cdot (1+\epsilon) \cdot (1 + O(\Delta)) \\ &= 1 + O(\Delta)\end{aligned}$$

where in the second step we have used claims 2 and 3. □

Putting claims 1 and 4 together completes the proof. To remove the assumption that all arms have $\xi_i > 0$, we need to show that $\mathcal{A}$ implies that for n sufficiently large, the arms $\hat{i}$ and i^* (the only arms that play a role in the proof) will have shape parameters > 0. This follows from the fact that if $\xi_i \leq 0$, m_n^i is $O(\ln(n))$, while if $\xi_i > \xi^* > 0$, m_n^i is $\Omega(n^{\xi_i})$. □

Bibliography

[1] Noga Alon, Baruch Awerbuch, and Yossi Azar. The online set cover problem. In *Proceedings of the* 35^{th} *annual ACM Symposium on Theory of Computing*, pages 100–105, 2003. 2.2

[2] Fadi A. Aloul, Arathi Ramani, Igor L. Markov, and Karem A. Sakallah. Generic ILP versus specialized 0-1 ILP: an update. In *Proceedings of the 2002 IEEE/ACM International Conference on Computer-Aided Design*, pages 450–457, 2002. 5

[3] H. Alt, L. Guibas, K. Mehlhorn, R. Karp, and A. Wigderson. A method for obtaining randomized algorithms with small tail probabilities. *Algorithmica*, 16(4/5):543–547, 1996. 3.2.2

[4] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47:235–256, 2002. 5.2.1, 5.2.1, 5.2.1

[5] Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing*, 32(1):48–77, 2002. 2.4.4, 2.4.4, 3.6.1, 2, 4.4.2, 5.2.1, 5.2.1

[6] Peter Auer and Claudio Gentile. Adaptive and self-confident on-line learning algorithms. In *Proceedings of the Thirteenth Annual Conference on Computational Learning Theory*, pages 107–117, 2000. 1, 3

[7] Giorgio Ausiello, Aristotelis Giannakos, and Vangelis Th. Paschos. Greedy algorithms for on-line set-covering and related problems. In *Twelfth Computing: The Australasian Theory Symposium (CATS2006)*, pages 145–151, 2006. 2.2

[8] Baruch Awerbuch and Robert Kleinberg. Adaptive routing with end-to-end feedback: Distributed learning and geometric approaches. In *Proceedings of the* 36^{th} *annual ACM Symposium on Theory of Computing*, pages 45–53, 2004. 2.4.4

[9] Shivnath Babu, Rajeev Motwani, Kamesh Munagala, Itaru Nishizawa, and Jennifer Widom. Adaptive ordering of pipelined stream filters. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, pages 407–418, 2004. 2.1, 2.2

[10] J. E. Beasley. OR-library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990. 4.5.2

[11] Donald. A. Berry and Bert Fristedt. *Bandit Problems: Sequential Allocation of Experiments*. Chapman and Hall, London, 1986. 5.1.1

[12] Avrim Blum and Yishay Mansour. From external to internal regret. *Journal of Machine Learning Research*, 8:1307–1324, 2007. 3.8, 3.8, 2, 3.9.7, 3.9.7

[13] Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281–300, 1997. 4.1.3

[14] J. L. Bresina. Heuristic-biased stochastic sampling. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 271–278, 1996. 1.1.4

[15] Peter Brucker, Bernd Jurisch, and Bernd Sievers. A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics*, 49(1-3):107–127, 1994. 4.5.2

[16] Nicolò Cesa-Bianchi, Yoav Freund, David Haussler, David Helmbold, Robert Schapire, and Manfred Warmuth. How to use expert advice. *Journal of the ACM*, 44(3):427–485, 1997. A.1, 21

[17] Nicolò Cesa-Bianchi, Gábor Lugosi, and Gilles Stoltz. Minimizing regret with label efficient prediction. *IEEE Transactions on Information Theory*, 51:2152–2162, 2005. 2.4.4, 2.4.4, 3.6.1, 3.6.2, 4.4.2

[18] Fan Chung and Linyuan Lu. Concentration inequalities and martingale inequalities – a survey. *Internet Mathematics*, 3(1):79–127, 2006. A.1

[19] Vincent A. Cicirello and Stephen F. Smith. Heuristic selection for stochastic search optimization: Modeling solution quality by extreme value theory. In *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming*, pages 197–211, 2004. 1.1.4, 1.1.4, 5.1.1, 5.2, 5.3, 5.4, 5.4.1

[20] Vincent A. Cicirello and Stephen F. Smith. Enhancing stochastic search performance by value-biased randomization of heuristics. *Journal of Heuristics*, 11(1):5–34, 2005. 1.1.4

[21] Vincent A. Cicirello and Stephen 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*, pages 1355–1361, 2005. 1.1.4, 1.1.4, 5.1, 5.1.1, 5.2, 5.4, 5.4.1, 5.4.4

[22] Edith Cohen, Amos Fiat, and Haim Kaplan. Efficient sequences of trials. In *Proceedings of the* 14^{th} *annual ACM-SIAM Symposium on Discrete Algorithms*, pages 737–746, 2003. 1.1.1, 2.1.4, 2.2

[23] Stuart Coles. *An Introduction to Statistical Modeling of Extreme Values*. Springer-Verlag, London, 2001. 1.1.4, 5.1, 5.3.1, 5.4.2

[24] Uriel Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998. 1.1.1, 2.1.3, 2.3.1

[25] Uriel Feige. Rigorous analysis of heuristics for NP-hard problems. In *Proceedings of the* 16^{th} *annual ACM-SIAM Symposium on Discrete Algorithms*, pages 927–927, 2005. 1

[26] Uriel Feige, László Lovász, and Prasad Tetali. Approximating min sum set cover. *Algorithmica*, 40(4):219–234, 2004. 1.1.1, 2.1.3, 2.1.4, 2.2, 2.3.1, 2.3.2, 3.1.3, 3.4.1, 4.4.1

[27] Thomas A. Feo and Mauricio G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133, 1995. 1.1.4

[28] Philip W. L. Fong. A quantitative study of hypothesis selection. In *Proceedings of the International Conference on Machine Learning*, pages 226–234, 1995. 5.3

[29] Yoav Freund, Robert E. Schapire, Yoram Singer, and Manfred K. Warmuth. Using and combining predictors that specialize. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 334–343, 1997. 3.8

[30] Matteo Gagliolo and Jürgen Schmidhuber. Learning dynamic algorithm portfolios. *Annals of Mathematics and Artificial Intelligence*, 47(3-4):295–328, 2006. 3.2.1

[31] Matteo Gagliolo and Jürgen Schmidhuber. Learning restart strategies. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, pages 792–797, 2007. 1.1.2, 3.2.2, 3.9.9

[32] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989. 3

[33] Carla P. Gomes and Bart Selman. Algorithm portfolios. *Artificial Intelligence*, 126:43–62, 2001. 1.1.2, 2.1, 2.1.5, 3.1.1, 3.2.1

[34] Carla P. Gomes, Bart Selman, Nuno Crato, and Henry Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfications problems. *Journal of Automated Reasoning*, 24(1/2):67–100, 2000. 2.1.5, 3.1.1

[35] Carla P. Gomes, Bart Selman, and Henry Kautz. Boosting combinatorial search through randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 431–437, 1998. 1.1.1, 1.1.2, 2.1, 2.1.4, 2.2, 3.1.1, 3.2.2, 3.9.9

[36] András György and György Ottucsák. Adaptive routing using expert advice. *The Computer Journal*, 49(2):180–189, 2006. 3.6.2

[37] Chih-Wei Hsu, Benjamin W. Wah, Ruoyun Huang, and Yixin Chen. New features in SGPlan for handling preferences and constraints in PDDL3.0. In *Proceedings of the Fifth International Planning Competition*, pages 39–42, 2006. 4.5.1

[38] Bernardo A. Huberman, Rajan M. Lukose, and Tad Hogg. An economics approach to hard computational problems. *Science*, 275:51–54, 1997. 1.1.2, 2.1, 2.1.5, 3.1.1, 3.2.1, 4

[39] A.S. Jain and S. Meeran. A state-of-the-art review of job-shop scheduling techniques. Technical report, Department of Applied Physics, Electronic and Mechanical Engineering, University of Dundee, Dundee, Scotland, 1998. 2

[40] Leslie P. Kaelbling. *Learning in Embedded Systems*. The MIT Press, Cambridge, MA, 1993. 5.1.1, 5.2.1

[41] Sham Kakade, Adam Kalai, and Katrina Ligett. Playing games with approximation algorithms. In *Proceedings of the* 39^{th} *annual ACM symposium on Theory of Computing*, pages 546–555, 2007. 2.2

[42] Adam Kalai and Santosh Vempala. Efficient algorithms for online decision problems. *Journal of Computer and Systems Sciences*, 71(3):291–307, 2005. 3.6.2

[43] Ming-Yang Kao, Yuan Ma, Michael Sipser, and Yiqun Yin. Optimal constructions of hybrid algorithms. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 372–381, 1994. 3.2.1

[44] Haim Kaplan, Eyal Kushilevitz, and Yishay Mansour. Learning with attribute costs. In *Proceedings of the* 37^{th} *annual ACM symposium on Theory of Computing*, pages 356–365, 2005. 1.1.1, 2.1.4, 2.2, 2.3.2

[45] Henry Kautz, Yongshao Ruan, and Eric Horvitz. Dynamic restarts. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pages 674–681, 2002. 3.2.2

[46] Henry Kautz and Bart Selman. Unifying SAT-based and graph-based planning. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 318–325, 1999. 3.9.9

[47] Henry Kautz, Bart Selman, and Joerg Hoffmann. SATPLAN: Planning as satisfiability. In *Proceedings of the Fifteenth International Planning Competition*, 2006. 4.1.1

[48] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the* 9^{th} *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 137–146, 2003. 2.1.5

[49] Samir Khuller, Anna Moss, and Joseph (Seffi) Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70(1):39–45, 1999. 1.1.1, 2.1.4, 2.2, 2.3.2

[50] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983. 3

[51] Donald E. Knuth. Optimum binary search trees. *Acta Informatica*, 1:14–25, 1971. 4.4.1

[52] R. E. Korf. Depth-first iterative deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985. 4.1.3

[53] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. The MIT Press, 1992. 3

[54] John R. Koza, Martin A. Keane, Matthew J. Streeter, William Mydlowec, Jessen Yu, and Guido Lanza. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers, 2003. 3

[55] Andreas Krause and Carlos Guestrin. Near-optimal nonmyopic value of information in graphical models. In *Proceedings of the* 21^{st} *Annual Conference on Uncertainty in Artificial Intelligence (UAI-05)*, pages 324–331, 2005. 2.1.5

[56] Andreas Krause and Carlos Guestrin. A note on the budgeted maximization of submodular functions. Technical Report CMU-CALD-05-103, Carnegie Mellon University, 2005. 1.1.1, 2.1.4, 2.2, 2.3.2

[57] Tze Leung Lai. Adaptive treatment allocation and the multi-armed bandit problem. *The Annals of Statistics*, 15(3):1091–1114, 1987. 5.2.1

[58] Kate Larson and Tuomas Sandholm. Using performance profile trees to improve deliberation control. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, pages 73–79, 2004. 2

[59] Kevin Leyton-Brown, Eugene Nudelman, Galen Andrew, James McFadden, and Yoav Shoham. Boosting as a metaphor for algorithm design. In *Proceedings of the Ninth International Conference on Principles and Practice of Constraint Programming*, pages 899–903, 2003. 3.2.1, 4

[60] Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994. 2.4.1, 3.8

[61] Michael Luby, Alistair Sinclair, and David Zuckerman. Optimal speedup of Las Vegas algorithms. *Information Processing Letters*, 47:173–180, 1993. 1.1.1, 1.1.2, 2.1, 2.1.4, 2.2, 3.1.2, 3.2.2, 3.2.2, A.2

[62] Adam Meyerson. Online facility location. In *FOCS '01: Proceedings of the* 42^{nd} *IEEE Symposium on Foundations of Computer Science*, pages 426–431, 2001. 2.2

[63] Rolf H. Möhring, Andreas S. Schulz, Frederik Stork, and Marc Uetz. Solving project scheduling problems by minimum cut computations. *Management Science*, 49(3):330–350, 2003. 5.4

[64] Kamesh Munagala, Shivnath Babu, Rajeev Motwani, Jennifer Widom, and Eiter Thomas. The pipelined set cover problem. In *Proceedings of the International Conference on Database Theory*, pages 83–98, 2005. 1.1.1, 2.1, 2.1.4, 2.1.5, 2.2, 2.3.2

[65] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions. *Mathematical Programming*, 14(1):265–294, 1978. 1.1.1, 2.1.4, 2.2

[66] Klaus Neumann, Christoph Schwindt, and Jürgen Zimmerman. *Project Scheduling with Time Windows and Scarce Resources*. Springer-Verlag, 2002. 5.4.1

[67] Marek Petrik. Learning parallel portfolios of algorithms. Master's thesis, Comenius University, 2005. 3.2.1, 3.4.3

[68] Marek Petrik and Shlomo Zilberstein. Learning parallel portfolios of algorithms. *Annals of Mathematics and Artificial Intelligence*, 48(1-2):85–106, 2006. 1.1.2, 3.2.1

[69] Jussi Rintanen. Evaluation strategies for planning as satisfiability. In *Proceedings of the Sixteenth European Conference on Artificial Intelligence*, pages 682–687, 2004. 4.1.3, 4.3.1, 4.5.1

[70] Herbert Robbins. Some aspects of sequential design of experiments. *Bulletin of the American Mathematical Society*, 58:527–535, 1952. 5.1.1

[71] Yongshao Ruan, Eric Horvitz, and Henry Kautz. Restart policies with dependence among runs: A dynamic programming approach. In *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming*, pages 573–586, 2002. 3.2.2

[72] Tuomas Sandholm. Terminating decision algorithms optimally. In *Proceedings of the Ninth International Conference on Principles and Practice of Constraint Programming*, pages 950–955, 2003. 2

[73] Tzur Sayag, Shai Fine, and Yishay Mansour. Combining multiple heuristics. In *Proceedings of the* 23^{rd} *International Symposium on Theoretical Aspects of Computer Science*, pages 242–253, 2006. 1.1.1, 1.1.2, 3.1.2, 3.2.1, 3.4.3, 3.4.3

[74] C. Schwindt. Generation of resource–constrained project scheduling problems with minimal and maximal time lags. Technical Report WIOR-489, Universität Karlsruhe, 1996. 5.4.1

[75] John Slaney and Sylvie Thiébaux. On the hardness of decision and optimisation problems. In *Proceedings of the* 13^{th} *European Conference on Artificial Intelligence (ECAI-98)*, pages 244–248, 1998. 4.2.2

[76] Matthew Streeter and Daniel Golovin. Online algorithms for maximizing submodular functions. Working paper, 2007. 1.1, 2.1.3

[77] Matthew Streeter, Daniel Golovin, and Stephen F. Smith. Combining multiple constraint solvers: Results on the CPAI'06 competition. In *Proceedings of the Second CSP/Max-CSP Solver Competition*, 2007.

[78] Matthew Streeter, Daniel Golovin, and Stephen F. Smith. Combining multiple heuristics online. In *Proceedings of the Twenty-Second Conference on Artificial Intelligence*, pages 1197–1203, 2007. 1.1, 1.1.1, 2.1.5, 2.3.2, 3.1.3

[79] Matthew Streeter, Daniel Golovin, and Stephen F. Smith. Restart schedules for ensembles of problem instances. In *Proceedings of the Twenty-Second Conference on Artificial Intelligence*, pages 1204–1210, 2007. 1.1, 1.1.1, 2.1.4, 2.2, 2.3.2, 3.1.3

[80] Matthew Streeter and Stephen F. Smith. An asymptotically optimal algorithm for the max k-armed bandit problem. In *Proceedings of the Twenty-First National Conference on Artificial lntelligence*, pages 135–142, 2006. 1.1, 5.1

[81] Matthew Streeter and Stephen F. Smith. Exploiting the power of local search in a branch and bound algorithm for job shop scheduling. In *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling*, pages 324–332, 2006. 4.5.2

[82] Matthew Streeter and Stephen F. Smith. A simple distribution-free approach to the max k-armed bandit problem. In *Proceedings of the Twelfth International Conference on Principles and Practice of Constraint Programming*, pages 560–574, 2006. 1.1, 1.1.4, 5.1

[83] Matthew Streeter and Stephen F. Smith. Using decision procedures efficiently for optimization. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling*, pages 312–319, 2007. 1.1, 4.1

[84] Maxim Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Operations Research Letters*, 32:41–43, 2004. 1.1.1, 2.1.3, 2.1.4, 2.2, 2.3.2, 2.3.2

[85] Sylvie Thiébaux, John Slaney, and Phil Kilby. Estimating the hardness of optimization. In *Proceedings of the* 14^{th} *European Conference on Artificial Intelligence (ECAI-2000)*, pages 123–127, 2000. 4.2.2

[86] Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984. 3.1.3

[87] Vijay V. Vazirani. *Approximation Algorithms*. Springer, 2001. 1

[88] Gerhard J. Woeginger. Exact algorithms for NP-hard problems: a survey. In *Combinatorial Optimization – Eureka, You Shrink!*, pages 185–207. Springer, 2003. 1

[89] Zhao Xing, Yixin Chen, and Weixiong Zhang. MaxPlan: Optimal planning by decomposed satisfiability and backward reduction. In *Proceedings of the Fifteenth International Planning Competition, International Conference on Automated Planning and Scheduling*, pages 53–56, 2006. 4.1.1

[90] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. SATzilla07: The design and analysis of an algorithm portfolio for SAT. In *Proceedings of the Thirteenth International Conference on Principles and Practice of Constraint Programming*, pages 712–727, 2007. 1.1.2, 3.2.1

[91] Lintao Zhang and Sharad Malik. The quest for efficient Boolean satisfiability solvers. In *Proceedings of the 14th International Conference on Computer Aided Verification*, pages 17–36, 2002. 2