Approximation Algorithms for Stochastic Scheduling Problems

by

Brian Christopher Dean

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2005

Author .....
Department of Electrical Engineering and Computer Science
January 31, 2005

Certified by .....
Michel X. Goemans
Professor of Applied Mathematics
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Approximation Algorithms for Stochastic Scheduling Problems

by

Brian Christopher Dean

Submitted to the Department of Electrical Engineering and Computer Science
on January 31, 2005, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science

Abstract

In this dissertation we study a broad class of stochastic scheduling problems characterized by the presence of hard deadline constraints. The input to such a problem is a set of jobs, each with an associated value, processing time, and deadline. We would like to schedule these jobs on a set of machines over time. In our stochastic setting, the processing time of each job is random, known in advance only as a probability distribution (and we make no assumptions about the structure of this distribution). Only after a job completes do we know its actual "instantiated" processing time with certainty. Each machine can process only a singe job at a time, and each job must be assigned to only one machine for processing. After a job starts processing we require that it must be allowed to complete — it cannot be canceled or "preempted" (put on hold and resumed later). Our goal is to devise a scheduling policy that maximizes the expected value of jobs that are scheduled by their deadlines.

A scheduling policy observes the state of our machines over time, and any time a machine becomes available for use, it selects a new job to execute on that machine. Scheduling policies can be classified as adaptive or non-adaptive based on whether or not they utilize information learned from the instantiation of processing times of previously-completed jobs in their future scheduling decisions. A novel aspect of our work lies in studying the benefit one can obtain through adaptivity, as we show that for all of our stochastic scheduling problems, adaptivity can only allow us to improve the expected value obtained by an optimal policy by at most a small constant factor.

All of the problems we consider are at least NP-hard since they contain the deterministic 0/1 knapsack problem as a special case. We therefore seek to develop approximation algorithms: algorithms that run in polynomial time and compute a policy whose expected value is provably close to that of an optimal adaptive policy. For all the problems we consider, we can approximate the expected value obtained by an optimal adaptive policy to within a small constant factor (which depends on the problem under consideration, but is always less than 10). A small handful of our results are pseudo-approximation algorithms, delivering an approximately optimal policy that is feasible with respect to a slightly expanded set of deadlines. Our algorithms utilize a wide variety of techniques, ranging from fairly well-established methods like randomized rounding to more novel techniques such as those we use to bound the expected value obtained by an optimal adaptive policy.

In the scheduling literature to date and also in practice, the "deadline" of a job refers to the time by which a job must be completed. We introduce a new model, called the start deadline model, in which the deadline of a job instead governs the time by which we must start the job. While there is no difference between this model and the standard "completion

deadline" model in a deterministic setting, we show that for our stochastic problems, one can generally obtain much stronger approximation results with much simpler analyses in the start deadline model.

The simplest problem variant we consider is the so-called stochastic knapsack problem, where all jobs share a common deadline and we schedule them on a single machine. The most general variant we consider involves scheduling jobs with individual deadlines on a set of "unrelated" parallel machines, where the value of a job and its processing time distribution can vary depending on the machine to which it is assigned. We also discuss algorithms based on dynamic programming for stochastic scheduling problems and their relatives in a discrete-time setting (where processing times are small integers), and we show how to use a new technique from signal processing called zero-delay convolution to improve the running time of dynamic programming algorithms for some of these problems.

Thesis Supervisor: Michel X. Goemans
Title: Professor of Applied Mathematics

Acknowledgments

There are many people I would like to thank for their support and friendship during my final years at MIT.

Foremost, I thank my advisor Michel Goemans, a true mentor from whom I have learned a great deal not only about computer science and mathematics, but also about how to be an upstanding member of the academic community. Anyone who has worked with Michel will comment on his eye for mathematical elegance — he taught me always to strive for simpler ways of explaining concepts, whether it is for the purpose of teaching a class or publishing a research paper. He has given me the freedom to pursue a broad range of my own interests as well as ample guidance to help me along the way. Michel is largely responsible for the fact that my graduate career at MIT has been perhaps the most enjoyable and rewarding time in my entire life.

I wish to thank the other members of my thesis committee: Piotr Indyk, David Karger, and Andreas Schulz. You have all given me helpful feedback, advice, and support throughout my graduate studies.

In my six terms as a teaching assistant, I have worked with and benefitted from the insight of many members of the MIT faculty, including Michel Goemans, Andreas Schulz, Piotr Indyk, Erik Demaine, Bruce Tidor, Dan Spielman, Shafi Goldwasser, Nancy Lynch, and Srini Devadas. I also wish to thank my former students, without whom teaching would not have been such an enjoyable and integral part of my graduate experience.

I especially would like to thank my good friend Jan Vondrák, with whom I have had the privilege of collaborating on much of the material in this dissertation. Without the benefit of Jan's keen insight, many of the results in this dissertation would be much weaker.

In my ten years at MIT, I have accumulated an amazing group of friends and colleagues from the Computer Science and AI Lab (CSAIL), the math department, the operations research center, my undergraduate dormitory Random Hall, and the Grodzinsky and Ortiz labs with which my wife is affiliated. I am extremely grateful to all of you for your friendship and support, and I have you to thank for many of the positive experiences I've at MIT.

I have benefitted substantially from my involvement in the USA Computing Olympiad, a

program that has allowed me to teach and interact with the brightest high school computer science students across the USA for the past eight years. The enthusiasm and intellect of these students has played a large role in the development of my proficiency and passion for algorithms and my desire to pursue an academic career in computer science. I wish to thank Don Piele, the director of the program, for his support, as well as the other coaches: Rob Kolstad, Hal Burch, Russ Cox, Greg Galperin, and Percy Liang.

Finally, I wish to thank my family. I acknowledge with gratitude my late grandmother, Virginia Ash, as well as my uncles Jeffrey Ash and Cameron Dean, for helping to recognize and support my interest in computers at an early age. My parents Rodney and Barbara always supported me both in my non-academic and academic ventures. They have always been there to encourage me, especially when my brilliant proofs occasionally turned out to be less than brilliant. I definitely would not be where I am today without them. My sister Sarah and her husband Jake have been a constant source of support and friendship. The "west coast" support team, consisting of my parents in law Christian and Catherine Le Cocq and my sister in law Cécile, have also instrumental in encouraging me and my wife Delphine throughout our studies. Finally, to Delphine, whom I started dating exactly eight years to the day prior to the defense of this thesis: you have always been there for me, believed in me, taken care of me, and supported me in all of my endeavors. Thank you.

Contents

1. Introduction

In this dissertation we study a broad class of *stochastic scheduling problems* characterized by the presence of hard deadline constraints. Scheduling problems in general involve the assignment of a set of jobs to a set of machines over time. By combining different machine environments, different objectives, and different side constraints, one can conceive of hundreds of useful scheduling problems. As a result of this abundance of problems as well as constant attention from researchers in the computer science, operations research, and optimization communities over the past several decades, the scheduling literature has become quite extensive, and scheduling theory has evolved into a rather mature discipline.

In a stochastic scheduling problem, the processing times of jobs are known in advance only with uncertainty. The processing time of each job is initially described in terms of a probability distribution, and the exact processing time of a job becomes known, or is "instantiated", only after the job is completed. As one might imagine, this stochastic model gives us a much more versatile means of expressing real world problems in a more realistic fashion than deterministic models in which the exact processing time of each job must be known in advance. One does not need to look very hard to find examples of large projects that have far exceeded their original deadlines as a result of poor planning in the face of uncertainty in the initial project parameters. Unfortunately, stochastic scheduling problems (and stochastic optimization problems in general) tend to be far more difficult to approach from both a theoretical and computational point of view than their deterministic counterparts.

Stochastic scheduling has received a fair amount of attention in the literature to date. For instance, there are no fewer than 343 entries in a bibliography of results in stochastic scheduling maintained until 1995 by Richard Weber [59]. However, the vast majority of the stochastic scheduling literature has been concerned with problems in which *all* jobs must be scheduled in a manner that minimizes some objective. Typical choices include minimizing a schedule's *makespan* (latest completion time over all jobs), or minimizing the sum of weighted completion times over jobs. We consider a different, but also natural objective: if we associate with each job a deadline and a value, it may not be possible to schedule all of the jobs successfully by their deadlines, but we would like to devise a schedule that maximizes the expected value of the jobs that are scheduled by their deadlines. In a deterministic setting, such problems (e.g., the NP-hard knapsack, multiple knapsack, and

interval scheduling problems) are fairly well understood; however, as Pinedo [46] writes in his recent book on scheduling theory, "Problems with due dates are significantly harder in a stochastic setting than in a deterministic setting."

In the chapters that follow, we study many variants of deadline-constrained stochastic scheduling problems. The simplest of these involves only a single machine and a set of jobs that all share a common deadline. We call this variant the *stochastic knapsack problem* since its deterministic counterpart is the well-studied 0/1 knapsack problem. In the most general variant we consider, jobs have individual release times and deadlines, there are multiple machines, and the processing time and value for a job can vary as a function of the machine to which it is assigned (this is known in the scheduling literature as the *unrelated parallel machine* environment). All of these problems are at least NP-hard since they contain the deterministic knapsack problem as a special case, so we focus our attention on the development of *approximation algorithms*: polynomial-time algorithms that output solutions that are provably close to an optimal solution. We develop $O(1)$-approximation algorithms for most of the problems we consider; that is, the expected value of our solution will always differ by at most a constant factor from that of an optimal solution. For the remaining handful of problems, we develop *pseudo-approximation algorithms* that obtain at least a constant factor of the expected value of an optimal solution, but that may output a schedule that is slightly infeasible (say, a schedule that would be feasible if all deadlines were doubled).

The solution to a stochastic scheduling problem is a *policy* that specifies the next job to schedule any time a machine becomes available. A scheduling policy is said to be *adaptive* if it makes these decisions based in part on information it has learned about the actual instantiated processing times of previously-scheduled jobs. A *non-adaptive* policy commits to a certain type of schedule in advance (say, an ordering of jobs to be scheduled in sequence) and adheres to it regardless of its performance as time progresses. An interesting facet of our work is the study of the *benefit of adaptivity* in stochastic scheduling problems. We show that for most of the problems considered in this dissertation, adaptivity can only improve the expected value of an optimal solution by at most a constant factor. This is proved constructively, by demonstrating approximation algorithms whose non-adaptive solution policies obtain expected values that fall within a constant factor of the expected values of optimal adaptive policies.

Many of the results in this dissertation, in particular those in Chapters 3 and 4, are based on joint work with Michel Goemans and Jan Vondrák [14, 15], some of which also appears in the Ph.D. thesis of Jan Vondrák [58]. Chapter 7 is based in part on [13].

1.1 Notation, Terminology, and Problem Formulation

The study of scheduling problems is plagued by an abundance of notation and terminology, even more so with stochastic scheduling problems. The majority of this chapter is devoted to a description of our stochastic scheduling models and problems, after which we briefly summarize the results to appear in the following chapters.

Let $[n] := \{1, 2, \ldots n\}$ be the indices for a set of n jobs, and let $[m]$ index a set of m machines. Each job j has an associated processing time p_j, value[1] w_j, and deadline[2] d_j. On occasion, we also associate a release time r_j with each job j, where r_j indicates the time at which job j becomes available for processing.

1.1.1 The Start Deadline and Completion Deadline Models

In the scheduling literature and in practice, the term *deadline* indicates the time by which a job must be completed. In the study of deadline-constrained stochastic scheduling problems, however, we will occasionally find it more natural and convenient to consider deadlines on the *start* times of jobs rather than the *completion* times. Nowhere in the literature does there seem to be a distinction made between these two concepts — primarily, it seems, because they are completely equivalent[3] in a deterministic setting, so there has been no reason to deviate from the standard notion of a deadline on completion time. The only model in the literature that is perhaps somewhat similar is that of scheduling with time windows or "lags" where the start time of one job may be constrained to be sufficiently close to the completion time of another job.

The results in this dissertation can be divided into two main classes depending on whether they apply to the *start deadline* model (a new model we introduce where deadlines are on job start times) or to the *completion deadline* model (the traditional model where deadlines apply to completion times). The reader with a background in scheduling theory, or with project scheduling in the real world, may ask the reasonable question: is it worthwhile to study the start deadline model? Arguably, this model does not coincide with many of the scheduling scenarios encountered in practice, where hard deadlines on completion times are desired. However, there are good reasons to consider the possibility of deadlines on start times. If we make the reasonable assumption in practice that our random job processing times have "well behaved" distributions without too much variability, then there is not so much difference between the start deadline and completion deadline models (and there is no difference in a deterministic setting). Furthermore, since our results in the start deadline model will tend to be simpler and stronger (i.e., we can get closer to the optimal solution), one can argue as with many scientific models that the analytical strength of the model compensates enough for the slight loss in realism.

The introduction of the start deadline model entails somewhat of a challenge with regard to notation and terminology. After careful consideration, we have opted to use the same term,

[1]The scheduling literature almost exclusively uses the term "weight". However, we prefer to use the term "value" for maximization problems and "cost" for minimization problems, since this leaves no doubt as to whether we are maximizing or minimizing. To maintain consistency with standard scheduling notation (described shortly), we use the symbols $w_1 \ldots w_n$ to denote the values/costs/weights of jobs.

[2]In the literature there is sometimes a distinction between a *deadline* $\bar{d}_j$ and a *due date* d_j, with deadlines being hard constraints and due dates being soft constraints that one might consider overrunning at some penalty. In this work, we use only the term deadline and the symbol d_j.

[3]To convert between the two models in a deterministic setting we simply increase or decrease each job's deadline by its deterministic processing time. The only potentially unpleasant side effect of this transformation is that for problems in which all deadlines coincide at a single point in time, this property is not preserved.

deadline, for both a start deadline and a completion deadline. There is really no other word that conveys as clearly the notion of a time by which something must be performed, whether it is the initiation or the completion of a job. Whenever there is potential for ambiguity, we use the terms *start deadline* and *completion deadline*. In the scheduling literature, the symbols $d_1 \ldots d_n$ are well-understood to represent the completion deadlines of our jobs. While it might seem less confusing to use different symbols (perhaps $\delta_1 \ldots \delta_n$ or $d'_1 \ldots d'_n$) for start deadlines, we actually find that it is less awkward to use $d_1 \ldots d_n$ for both purposes. Many of our results are built upon a common framework that applies to both models, in which it is quite cumbersome to maintain two copies of each formula just for the sake of using a different deadline symbol. We clearly indicate throughout our discussion whether we are in the start deadline or completion deadline model, so there should be no chance of confusion resulting from the dual use of $d_1 \ldots d_n$.

1.1.2 Random Processing Times

Job processing times are independent random variables whose distributions are provided to us as input. An important feature of our results is that they do not require "special" types of processing time distributions. By way of contrast, a large majority of the stochastic scheduling literature focuses on special classes of distributions (e.g., exponential, Gaussian, etc.). We generally assume very little about how processing time distributions are described:

- Our algorithms for the completion deadline model assume that we know $\mathbf{Pr}[p_j \leq d_j]$, and one of these (Section 4.2) requires $\mathbf{Pr}[p_j \leq t]$ for any t.
- We also assume for each job j that we know its *mean truncated processing time*, $\mu_j = \mathbf{E}[\min(p_j, d_j)]$. For technical reasons, this turns out to be a more natural way[4] for us to measure the "expected" processing time of job j than $\mathbf{E}[p_j]$. For some problems (starting in Chapter 5) we assume that we know $\mathbf{E}[\min(p_j, t)]$ for certain other values of t than $t = d_j$ — e.g., for $t = d_j - r_j$, or for $t = d_i$ where $i \neq j$.

For the most part, we treat the p_j's and other inputs as abstract real numbers without worrying about the sizes of their binary representations on a digital computer. Most of our algorithms have strongly-polynomial running times, running in polynomial time independent of the magnitude of the numbers in their input, if we assume that all fundamental arithmetic operations take constant time (a standard assumption). Only one of our results (Section 4.2) involves a weakly-polynomial algorithm whose running time is sensitive to the number of bits in the input.

Finally, we assume that $E[p_j] > 0$ for all jobs, since all jobs j with $E[p_j] = 0$ can be scheduled instantaneously at the beginning of time.

[4] To see one case where this is apparent, consider a single-machine scheduling problem in the completion deadline model where all jobs share a common deadline d. In the event that $p_j > d$ for some job j, it does not really matter how much larger than the deadline p_j turns out to be, since we will not be receiving any value from j. Our instance therefore behaves equivalently to one in which p_j has a support in $[0, d] \cup \{\infty\}$, in which case the standard measure of expected value might be infinite.

1.1.3 Assumption: No Cancellation or Preemption

The strongest and perhaps most objectionable assumption we make in this dissertation is that once a job is started, it must be allowed to run to completion. Cancellation or preemption (putting a job on hold to be resumed later) are forbidden, even in the completion deadline model when we reach the deadline of a job. One can certainly argue against these assumptions as being overly restrictive compared to our typical options in practice. For example, as a job j undergoes processing, we learn more information about p_j over time since we can condition on the fact that $p_j \geq t$ for increasing values of t. It is certainly conceivable that we may reach a point where we know for sure that j will not complete by its deadline, or that j will take an unreasonable amount of additional time to complete, so in practice we might want to cancel j at this point. Another objectionable case occurs in the completion deadline model when we reach the deadline of a job j but find ourselves unable to terminate j immediately even though it is now a worthless job. This case is not problematic if all jobs share a common deadline (and this is a rather common case), but it can make a noticeable difference if jobs have individual deadlines.

We impose our non-cancellation assumption since it seems necessary to simplify our models to the point where we can achieve good approximation guarantees. Consider, for example, a single-machine instance with n unit-value jobs whose processing times are either 0 (probability 1/2) or ∞ (probability 1/2), where all jobs share a common deadline at time $t = 1$. By canceling or preempting jobs that do not complete immediately, we can successfully schedule roughly $n/2$ jobs in expectation, whereas otherwise we could only schedule 1 job in expectation in the completion deadline model or 2 jobs in expectation in the start deadline model. If we are allowed to terminate jobs at their deadlines in the completion deadline model, then we can also schedule roughly $n/2$ jobs in expectation. One might think this is somehow a degenerate case that occurs when we allow processing times of zero, but we can also modify the instance above by spacing deadlines out an infinitesimal amount around time $t = 1$ so that processing times need not be zero. We conclude from this example that if cancellation/preemption is allowed, our scheduling policies *must* somehow take advantage of this fact or else we can only hope to obtain an $O(1/n)$ fraction of the expected value of an optimal schedule in the worst case. At the present time, we do not know how to utilize this extra power.

1.1.4 Random Job Values

Although we typically assume that job values $w_1 \ldots w_n$ are deterministic, we can also accommodate job values that are random with known distributions (instantiated when a job is scheduled). The only requirement here is that $w_1 \ldots w_n$ must be mutually independent as well as independent from $p_1 \ldots p_n$ (in the start deadline model, dependence between p_j and w_j is permitted). In this case, we simply replace $w_1 \ldots w_n$ (random) with $\mathbf{E}[w_1] \ldots \mathbf{E}[w_n]$ (deterministic) and thereby reduce our instance to one having deterministic job values.

We claim that the reduction above (i) does not change the expected value obtained by our algorithms, and (ii) also does not change the expected value obtained by an optimal

scheduling policy. To see this, let X_j be an indicator random variable that tells us whether or not job j is successfully scheduled. Given our independence assumptions above, w_j and X_j must be independent, since X_j depends only on random choices and events prior to the decision to schedule job j (including, in the completion deadline model, the outcome of j's own processing time), and none of these have any influence or dependence on the instantiation of w_j. Now consider the objective we are trying to maximize:

$$\text{Maximize } \mathbf{E}\left[\sum_{j=1}^{n} w_j X_j\right] = \sum_{j=1}^{n} \mathbf{E}[w_j]\mathbf{Pr}[\text{job } j \text{ scheduled successfully}].$$

The $\mathbf{E}[w_j]$ terms are of course not affected when we replace the w_j's with their deterministic expectations. For any optimal policy, $E[X_j] = \mathbf{Pr}[\text{job } j \text{ successfully scheduled}]$ is not affected due to independence. The change also does not affect $\mathbf{E}[X_j]$ for the algorithms in this dissertation, since the only property of the distribution of w_j they use is $\mathbf{E}[w_j]$.

1.1.5 Scheduling Notation

For completeness, we briefly review the standard notation for scheduling problems used in the literature. Scheduling problems are typically described using a three-field $\alpha \mid \beta \mid \gamma$ notation initially popularized by Lawler, Lenstra, and Rinnooy Kan [26]. The α field represents the machine environment. In this dissertation, we concern ourselves with the following common choices:

- **1 (single machine).** In this environment, a schedule simply corresponds to a linear ordering of the jobs to be processed in sequence.
- **P (identical parallel machines).** Here, we have m machines available for processing, all running at the same speed.
- **Q (uniformly related parallel machines).** In this case, our m machines can run at different speeds $s_1 \ldots s_m$.
- **R (unrelated parallel machines).** In this model, the processing time of a job can vary depending on the machine to which it is assigned. Our processing times are of the form p_{ij} for $(i, j) \in [n] \times [m]$. This contains the uniformly related model as a special case, and also allows us to prohibit certain (job, machine) pairings by assigning them infinite processing times.

The β field indicates any additional constraints that might be present in our problem. Typical entries in this field include:

- $p_j \sim stoch$ **(stochastic processing times).** This entry will be present in all of the problems we consider.
- $d_j = d$ **(common deadlines).** All jobs share the same deadline d. This generally simplifies our problems quite a bit.

- r_j **(release times).** Jobs have associated release times (also called release dates). Without this constraint, we assume jobs are all released at time $t = 0$. When release times are involved, we assume an *off-line* model where all jobs (even those not released at time $t = 0$) are known in advance. By contrast, in *on-line* scheduling models, we know nothing about a job until the point in time at which it is released.

- **prec (precedence constraints).** Here, we specify along with our jobs a directed acyclic graph (DAG) whose edges (i, j) indicate precedence relations of the form "job j can only be scheduled after job i completes". We do not consider precedence constraints in any of our problems, and in general it seems very difficult to mix precedence constraints and deadlines (this issue is discussed further in Chapter 8).

- **pmtn (preemption).** Without this field set, jobs must be scheduled in a non-preemptive (i.e., contiguous, or "integral") fashion. If preemption is allowed, we can interrupt a job and resume working on it later. Preemption typically reduces the computational complexity of a scheduling problem, just as removing integrality constraints typically reduces the complexity of an optimization problem. In this dissertation, we only consider preemption in deterministic models (without $p_j \sim stoch$), since as we mentioned, preemption gives us too much power in the stochastic case.

Finally, γ indicates the objective. Common objectives include:

- C_{max} **(minimize makespan).** The completion time of job j is typically denoted C_j (by convention, uppercase letters are used to denote the output of a scheduling algorithm, and lowercase letters its input). The makespan of a schedule, C_{max}, is the maximum completion time over all jobs. In a stochastic setting we write $\mathbf{E}[C_{max}]$, since we wish to minimize expected makespan (this convention applies to all the objectives below).

- $\sum w_j C_j$ **(minimize sum of weighted completion times.)** This is one of the more common objectives in the scheduling literature. For problems with nonzero release times, one often considers instead the objective of minimizing the sum of weighted *flow times*: $\sum w_j F_j$, where $F_j = C_j - r_j$. This objective is equivalent (at least when computing an exact, rather than approximately optimal solution) to minimizing $\sum w_j C_j$, since the two differ by an additive amount of $\sum_j w_j r_j$.

- L_{max} **(minimize maximum lateness.)** The *lateness* of job j is defined as $L_j = C_j - d_j$.

- $\sum w_j T_j$ **(minimize sum of weighted tardiness.)** The *tardiness* of a job is defined as $T_j = \max(0, L_j) = \max(0, C_j - d_j)$.

- $\sum U_j$ **(minimize number of tardy jobs.)** The indicator variable U_j takes the value 1 if j completes after its deadline, and 0 otherwise.

- $\sum w_j \overline{U}_j$ **(maximize weight of successfully scheduled jobs in the completion deadline model.)** This is one of the two objectives on which we focus in this dissertation. The indicator variable $\overline{U}_j$ (sometimes written as $1 - U_j$) takes the value 1

if job j completes no later than its deadline. In terms of computing an exact optimal solution, this objective is equivalent to $\sum w_j U_j$ (minimizing the weight of tardy jobs). However, the two objectives must be considered separately for purposes of approximating an optimal solution. In this dissertation, we focus on the maximization objective.

- $\sum w_j \overline{V}_j$ **(maximize weight of successfully scheduled jobs in the start deadline model.)** In this dissertation, we introduce a new indicator variable V_j that behaves just like U_j except it applies to start deadlines instead of completion deadlines. We have $\overline{V}_j = 1$ only if job j starts no later than its deadline. One can also consider the minimization objective $\sum w_j V_j$, although as above we will focus on the maximization case.

For example, one of the simplest problems we consider is $1 \mid p_j \sim stoch, d_j = d \mid \mathbf{E}[\sum w_j \overline{U}_j]$: we have a single machine, all jobs have stochastic processing times and a common deadline d, and our objective is to maximize the expected total weight of the jobs that complete by their deadlines. We refer to this problem and its analog $1 \mid p_j \sim stoch, d_j = d \mid \mathbf{E}[\sum w_j \overline{V}_j]$ in the start deadline model as variants of the *stochastic knapsack problem*, since these problems both involve a single machine (knapsack) with a common deadline (capacity). One of the more complicated problems we consider is $R \mid p_j \sim stoch \mid \mathbf{E}[\sum w_j \overline{V}_j]$: scheduling on unrelated machines in the start deadline model where each job has its own individual deadline. Remember that if our objective contains U_j or $\overline{U}_j$, we are in the completion deadline model, and if our objective contains V_j or $\overline{V}_j$, then we are in the start deadline model.

1.1.6 Adaptivity

In terms of solution structure, there is a significant difference between deterministic and stochastic scheduling problems. The solution to a deterministic scheduling problem is quite straightforward — it is just an assignment of jobs over time to each machine. The solution to a stochastic scheduling problem, on the other hand, is a scheduling *policy* that decides the next job to schedule any time a machine becomes available. For an *adaptive* policy (also known in the literature as a *dynamic* policy), this decision can use information that is known about the current "state" of execution:

- what is the current time? (i.e., how much time remains before the deadline of each outstanding job?)
- which jobs are currently processing on busy machines, and how long have they been processing?
- which jobs have already been completed? (we now know the true processing time of each of these jobs, but since p_j's are independent this information does not really help us in our future scheduling decisions).

On a single machine, an adaptive policy can be formally defined as a mapping from (time, set of jobs already processed) to the next job to process. In a parallel machine environment, domain of this mapping also includes the index and start time for the job currently pending on each machine that is currently busy.

A *non-adaptive* policy is a policy that cannot make any decisions based on information learned from the instantiation of processing times during scheduling. On a single machine, it specifies in advance an ordering of jobs to schedule in sequence. On a single machine, this is fairly straightforward if all jobs share a common deadline, since in this case we schedule jobs from our ordering until we reach the deadline (and in the completion deadline model, we get no credit for the final job that is still processing). If jobs have individual deadlines, then the non-adaptive model is actually somewhat weak since it might ask us to continue scheduling jobs from our ordering even if it turns out that their deadlines have already expired. One might think it is reasonable to give a non-adaptive policy the power to skip over a job if its deadline has already passed, but this can be interpreted as a limited form of adaptivity, since it depends on instantiated processing times of prior jobs. In a few pages we describe somewhat-related partially adaptive models (called *ordered adaptive models*) that adhere to an ordering but are allowed to skip jobs.

In a parallel machine environment, a non-adaptive policy partitions the jobs in advance into disjoint sequences for each of the machines, after which each machine behaves locally as in the single-machine case above. In addition to this type of "pre-assignment" policy, we will also be considering in this dissertation policies based on a single "global" sequence of jobs. In this case, any time a machine becomes available we schedule on it the next feasible job from this sequence. In deterministic scheduling, algorithms of this type are called *list scheduling* algorithms, and in the stochastic scheduling literature they are sometimes known as *static list* policies. One should note that these types of policy are actually adaptive (albeit in a very limited way) according to our definition.

Adaptive versus non-adaptive types of solutions have been studied in different contexts from our stochastic scheduling models. For example, Borodin et al. [4] and Davis and Impagliazzo [12] develop and analyze "adaptive" and "non-adaptive" models of greedy algorithms for deterministic optimization problems (including deterministic scheduling problems).

1.1.7 Representing an Adaptive Policy

It is considerably easier to describe a non-adaptive policy than an adaptive policy, since a non-adaptive policy is just a sequence of jobs to schedule on each machines, while an adaptive policy is defined in terms of a complicated mapping as described in the previous section. If our processing time distributions are discrete, we can visualize an adaptive policy in terms of a decision tree, as shown in Figure 1-1. The figure provides a sample instance of the problem $1 \mid p_j \sim stoch, d_j = d \mid \mathbf{E}[\sum w_j \overline{U}_j]$ where all jobs have unit value and a common deadline at time $d = 10$, and we are in the completion deadline model. Each node in the tree corresponds to a state (t, J), with t being the current time and J the set of jobs that have already completed. Our adaptive policy specifies, at each non-leaf node (t, J), the job $j \in [n] \backslash J$ to schedule next, and the edges emanating from the node correspond to

Job	w_j	d_j	p_j
1	1	10	2 (Pr. 1/2) or 6 (Pr. 1/2)
2	1	10	8 (Pr. 1)
3	1	10	4 (Pr. 1/2) or 9 (Pr. 1/2)

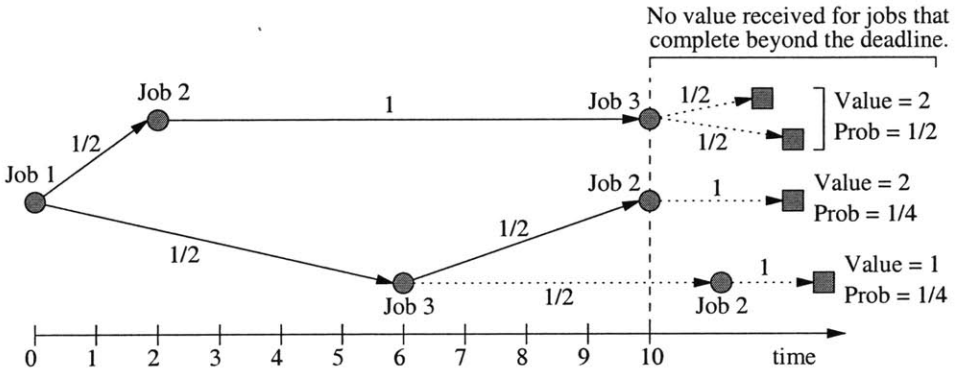

Figure 1-1: Illustration of an adaptive policy in terms of a decision tree, drawn on top of a timeline. Each node in the decision tree corresponds to a particular time and a particular set of jobs we have already scheduled, and specifies the next job to schedule. This particular example is in the completion deadline model, since we obtain no credit for a job whose processing time runs over the deadline. Note that there is no harm in assuming that our policy continues to schedule jobs beyond the deadline — these jobs of course receive no value.

the possible instantiations of p_j. The expected value obtained by the policy is obtained by summing over its leaves. In our example, we succeed in completing 2 jobs by the deadline with probability 3/4, and 1 job with probability 1/4, so $\mathbf{E}[\sum w_j \overline{U}_j] = 7/4$. If we happen to have a parallel machine environment (with discrete processing times), we can still picture an adaptive policy in terms of a decision tree, although the states represented by nodes will be slightly more complicated. In the example above where we have a common deadline, we can effectively think of our policy as terminating at the deadline; however it will occasionally be simpler to assume that our adaptive policy is defined in a way so that it continues to schedule jobs (even beyond their deadlines) until all jobs are scheduled. For example, this lets us speak clearly about the notion of the expected time at which a job is scheduled. Of course, we still only receive value for those jobs scheduled by their respective deadlines.

Even if the p_j's have relatively simple discrete distributions, it is not clear exactly how much space might be required to represent the decision tree corresponding to an optimal, or even approximately-optimal adaptive policy. This is one of the reasons we typically choose to compute non-adaptive solutions rather than adaptive solutions — since the complexity of the output is substantially lower. When we do produce adaptive solutions as output (in Section 4.2), we encode them implicitly in terms of a polynomial-time algorithm that takes a state (t, J) and outputs the next job to schedule.

Job	w_j	d_j	p_j
1	ε	1	0 (Pr. 1/2) or 1 (Pr. 1/2)
2	ε	1	1 (Pr. 1)
3	1	1	0 (Pr. ε) or ∞ (Pr. $1-\varepsilon$)

Figure 1-2: An instance of $1 \mid p_j \sim stoch, d_j = d \mid \mathbf{E}[\sum w_j \overline{U}_j]$ having adaptivity gap arbitrarily close to 5/4. An optimal adaptive policy schedules job 1 followed by jobs 2 and 3 (if $p_1 = 0$) or job 3 (if $p_1 = 1$). An optimal non-adaptive policy schedules jobs in the order 1, 2, 3.

Job	w_j	d_j	p_j
1	$\sqrt{2}-1$	1	0 (Pr. $2-\sqrt{2}$) or 1 (Pr. $\sqrt{2}-1$)
2	$\sqrt{2}-1$	1	1 (Pr. 1)
3	1	1	∞ (Pr. 1)

Figure 1-3: An instance of $1 \mid p_j \sim stoch, d_j = d \mid \mathbf{E}[\sum w_j \overline{V}_j]$ with adaptivity gap $4 - 2\sqrt{2} > 1.171$. An optimal adaptive policy schedules job 1, followed by jobs 2 and 3 (if $p_1 = 0$) or job 3 (if $p_1 = 1$), and the orderings 1, 2, 3 and 2, 3, 1 are both optimal non-adaptive policies.

1.1.8 Adaptivity Gap

What is the best non-adaptive policy for the sample instance shown in Figure 1-1? It is impossible to schedule all 3 jobs so they all complete by the deadline, and for any ordering of jobs, there is at most a 1/2 probability that we succeed in scheduling 2 jobs (the first job always completes successfully). Therefore, the optimal expected value obtained by a non-adaptive policy is 3/2. For example, we could schedule jobs in the order 1, 2, 3. Perhaps not surprisingly, adaptivity allows us to achieve a higher expected value. To quantify how much better the adaptive solution can be (i.e., to measure the *benefit of adaptivity*), we advocate the use of a natural measure we call the *adaptivity gap*.

Definition. The *adaptivity gap* for a problem instance is the ratio between the expected value obtained by an optimal adaptive solution divided by the expected value obtained by an optimal non-adaptive solution. The adaptivity gap for a problem is the maximum such gap over all possible instances.

For the instance shown in Figure 1-1, the adaptivity gap is 7/6. Figure 1-2 gives another instance with adaptivity gap arbitrarily close to 5/4. This gap of 5/4 carries over to every other problem we study in this dissertation in the completion deadline model, since they all contain $1 \mid p_j \sim stoch, d_j = d \mid \mathbf{E}[\sum w_j \overline{U}_j]$ as a special case. In the start deadline model, Figure 1-3 shows an instance with adaptivity gap $4 - 2\sqrt{2} > 1.171$, and this carries over to all other start deadline problems we study in this dissertation, since they all contain $1 \mid p_j \sim stoch, d_j = d \mid \mathbf{E}[\sum w_j \overline{V}_j]$ as a special case.

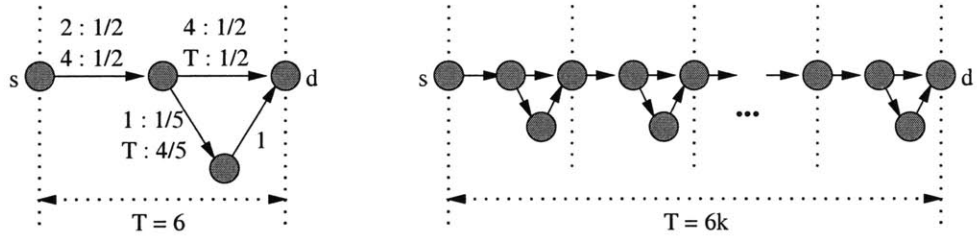

Figure 1-4: (a) An instance of the stochastic shortest path problem for which a better probability of arrival by a deadline T is possible if adaptive routing is allowed. The best non-adaptive path (the straight edges across the top of the graph) arrives on time with probability $1/4$, where the best adaptive route arrives on time with probability at least $7/20$. By concatenating k copies of (a), we see in (b) an instance where the non-adaptive and adaptive probabilities are $(1/4)^k$ and at least $(7/20)^k$, so our adaptivity gap is exponentially large factor in the size of the graph.

We wish to stress that the notion of adaptivity gap can be applied not only to stochastic scheduling, but also to any stochastic optimization problem in which solutions are sequentially constructed. For example, consider the *stochastic shortest path problem*, where the length of every edge in a graph is an independent random variable (with known distribution), and we would like to travel from a designated source node s to a designated destination node d along a path that maximizes our probability of arriving by some deadline T. A non-adaptive policy for this problem would select a static path to follow, while an adaptive policy would select an edge outgoing from the source, follow it, and then decide (based on the instantiated length of the edge) which edge to follow next. The adaptivity gap of this problem can be exponentially large in the size of the graph (Figure 1-4). By way of contrast, most of the stochastic scheduling problems we consider in this dissertation will be shown to have only a constant adaptivity gap.

The notion of an adaptive scheduling policy is quite prevalent in the stochastic scheduling literature. However, our notion of adaptivity gap (largest possible ratio of the expected value of an optimal adaptive solution to that of an optimal non-adaptive solution) does not seem to have received much explicit attention thus far in the literature. The work of Möhring, Schulz, and Uetz [41] (which we discuss in more detail in Chapter 2) implicitly considers the adaptivity gap of the problems $1 \mid p_j \sim stoch, prec \mid \mathbf{E}[\sum w_j C_j]$, $1 \mid p_j \sim stoch, r_j, prec \mid \mathbf{E}[\sum w_j C_j]$, and $P \mid p_j \sim stoch \mid \mathbf{E}[\sum w_j C_j]$, since it analyzes the performance guarantee for several non-adaptive scheduling policies with respect to an optimal adaptive policy.

1.1.9 The "Fixed Set" and "Ordered" Models

In this section we describe several other special models for adaptive and non-adaptive policies, which we consider in more detail in Chapter 7. Let us focus our attention here on the single-machine problem with common deadlines (i.e., the stochastic knapsack problem). A non-adaptive policy for this problem is a sequence of jobs to schedule, and we get credit for whatever prefix of this sequence we manage to schedule by the deadline. A slightly weaker model is the following: suppose we must choose as output a set of jobs (not a sequence), and we only receive the value for the jobs in the set if they all manage to start/complete prior to the deadline (so the order in which the jobs are scheduled does not matter). We call this the *fixed set* model. The maximum expected value we can obtain in the fixed set model is at most the expected value we can obtain with an optimal non-adaptive policy, since it is a feasible non-adaptive policy to schedule (in any order) the jobs belonging to an optimal fixed set, and whereas the value for the fixed set model is received on an "all or nothing" basis, in the non-adaptive model we get partial credit if we only succeed in scheduling some of the jobs in our set.

We also introduce the notion of an *ordered* adaptive policy. Suppose we fix an ordering of the jobs. As we proceed down this ordering in sequence, we can either schedule each job (deadline permitting) or skip over it permanently. If the initial ordering of jobs is fixed and given to us as input, we call this the *inflexible ordered adaptive* model. If our policy is allowed to choose the ordering as a preprocessing step, we call this the *flexible ordered adaptive* model. For any instance, the maximum possible expected value we can obtain is from an optimal adaptive policy, followed by an optimal flexible ordered adaptive policy and then by either an optimal non-adaptive policy or an optimal inflexible ordered adaptive policy. All of these policies will be at least as good as the optimal fixed set solution.

With the flexible ordered adaptive model, we encounter the somewhat interesting question of whether or not one can easily compute an ordering of jobs that gives the maximum expected performance when used to construct an optimal ordered adaptive policy. We currently do not know the answer to this question, although we will show in Chapter 7 that regardless of the ordering of the jobs, the expected value obtained by an optimal ordered adaptive policy always falls within a constant factor (8 in the start deadline model, and 9.5 in the completion deadline model) of the expected value obtained by a general optimal adaptive policy. Another interesting question is the following: consider the order in which jobs are scheduled by an optimal adaptive policy. How many such possible orderings might there be? This number can actually be quite large, as we see from the following example. Suppose we have n unit-value jobs that share a large integral deadline $d > n$. Jobs come in two flavors. Half of the jobs are type A, for which p_j takes the values 1, 2, or 3 each with probability ε, and $5, 7, 9, \ldots, 3 + (1 - 3\varepsilon)/\varepsilon$ each with probability 2ε. The remaining half of the jobs are of type B, and have processing time 1 or 2 each with probability ε, and $4, 6, 8, \ldots, 2 + (1 - 2\varepsilon)/\varepsilon$ each with probability 2ε. If ε is sufficiently small, then the optimal policy uses a type A job (if available) if the remaining capacity has odd parity and a type B job if the remaining capacity has even parity. Depending on the random outcomes of the processing times, there are at least $\binom{n}{n/2} \geq 2^n/n = \Omega(2^n)$ different orderings of the jobs that can be produced by an optimal adaptive policy.

1.1.10 Approximation Algorithms

We assume the reader is familiar with the complexity classes P, NP, #P, and PSPACE, and with the terms *NP-hard*, *#P-hard*, and *PSPACE-hard*. An *α-approximation algorithm* is a polynomial-time algorithm whose solution is always within a factor of α of an optimal solution. The factor α is called its *performance guarantee*, and we adopt the convention that $\alpha > 1$ for maximization problems, so what we call a 2-approximation might be called a $\frac{1}{2}$-approximation elsewhere in the literature. A particular class of interest in approximation algorithms is the *polynomial-time approximation scheme* (PTAS). An algorithm (parameterized by some number ε) is a PTAS if it runs in polynomial time as long as ε is a fixed constant, and delivers a $(1+\varepsilon)$-approximate solution. If the running time only depends polynomially on $1/\varepsilon$, then the algorithm is called a *fully-polynomial-time approximation scheme* (FPTAS). For the purposes of computing a performance guarantee, we always compare the expected value obtained by a policy to that obtained by an optimal adaptive policy.

In Chapter 5 some of our results will be *pseudo-approximation algorithms*. These deliver solutions that are "nearly feasible" and close in objective value to an optimal "feasible" solution. We say that a policy is an (α, β)-approximation algorithm if it delivers a solution whose expected value is at most a factor of α different from that of an optimal adaptive policy, where our policy has the extra advantage of being evaluated in an environment where all deadlines are stretched out by a factor of β. By this, we mean that if job j has original release time r_j and deadline d_j, then we use the quantity $r_j + \beta(d_j - r_j)$ as the new "extended deadline" for j. We will generally only investigate pseudo-approximation algorithms if it otherwise seems very difficult to obtain a good performance guarantee for a standard approximation algorithm.

1.1.11 Complexity

All of the stochastic scheduling problems considered in this thesis are at least NP-hard since their deterministic variants are NP-hard (they contain the 0/1 knapsack problem as a special case). The true complexity of the stochastic variants is currently open and may be somewhat worse than NP-hard. There is some evidence to support this: in [14, 58], it is shown that for $1 \mid p_j \sim stoch, d_j = d \mid \mathbf{E}[\sum w_j \overline{U}_j]$, the problem of computing a policy maximizing the probability that some job completes *exactly* at the deadline is PSPACE-hard. Also, in [15, 58] it is shown that the *stochastic set packing* generalization of $1 \mid p_j \sim stoch, d_j = d \mid \mathbf{E}[\sum w_j \overline{U}_j]$ in which processing times are vector-valued (in two dimensions or higher) is PSPACE-hard.

Several stochastic optimization problems related to ours are known to be #P-hard. For example, the problem of computing the probability that a given set of jobs as a whole will complete before a global deadline is known to be #P-hard [35]. In addition, if our jobs have precedence constraints and we have an infinite number of parallel machines, then for any instantiation of the job processing times, we can compute a minimum-makespan schedule by solving a longest path problem in a DAG. If C^* denotes the optimal makespan for a random instantiation of the processing times, then the problem of computing even a single

point on the cumulative distribution of C^* is #P-hard [27].

1.2 Models for Analyzing On-Line Algorithms

It is possible to consider our stochastic scheduling problems within the framework of *on-line* computation, since we are trying to compute the best possible solution despite limitations on the initial information we know about our instance. Several models have been proposed in the literature for evaluating the performance of an on-line algorithm, and it is worth considering whether any of these are potentially well-suited for our problems.

The standard notion of *competitive analysis* used to analyze an on-line algorithm asks us how much better we could have done had we known, in advance, what future conditions would be (i.e., what are the true processing times of all jobs). In other words, we would be comparing the expected value obtained by one of our solution policies to an *omniscient* policy that knows the true processing times of all jobs in advance. More formally, let I denote an instance of one of our scheduling problems, and let π denote a particular vector of realizations for our processing times for the instance I. We denote by $P(I,\pi)$ the value obtained by a particular adaptive scheduling policy P running on instance I given that our processing times instantiate to π (note that P doesn't know the vector π in advance, although it will discover some of its entries as time progresses). Similarly, we denote by $OPT(I,\pi)$ the value obtained by an optimal omniscient policy. The competitive ratio of the policy P can now be written as

$$\max_I \max_\pi \frac{OPT(I,\pi)}{P(I,\pi)}.$$

Unfortunately, this ratio can be as large as $\Omega(n)$ since an omniscient policy simply has too much power. For example, suppose we have an instance of $1 \mid p_j \sim stoch, d_j = d \mid \mathbf{E}[\sum w_j \overline{U}_j]$ with n unit-value jobs with processing times equal to 0 (probability 1/2) or ∞ (probability 1/2). Here, an optimal omniscient policy achieves expected value $n/2$ while an optimal adaptive policy cannot achieve expected value more than 1.

Several alternative models have been proposed in the on-line computation literature to try and "fix" the shortcomings of competitive analysis (namely, that it compares our performance against an adversary that has too much power). Coffmann and Gilbert [11] and Scharbrodt et al. [49] advocate a measure known as the *expected competitive ratio*,

$$\max_I \mathbf{E}_\pi \left[\frac{OPT(I,\pi)}{P(I,\pi)}\right].$$

For the problem $P \mid p_j \sim stoch \mid \sum C_j$ with exponentially distributed processing times, it is shown in [49] that the expected competitive ratio for the *shortest expected processing time* (SEPT) policy (taking jobs in the order $\mathbf{E}[p_1] \leq \mathbf{E}[p_2] \leq \ldots \leq \mathbf{E}[p_n]$) is $O(1)$ even though the traditional competitive ratio is $\Omega(n)$. Unfortunately, it does not seem possible to make any sort of similar statement for our deadline-constrained problems, especially since we seek

results that hold for arbitrary types of processing time distributions (intuitively, it seems that the expected competitive ratio should grow rapidly once we introduce distributions that have a significant chance of deviating from their means).

Koutsoupias and Papadimitriou [38] offer two more alternatives. The first is what they call the *diffuse adversary model*, where we apply standard competitive analysis but assume that π can be drawn (by a malicious adversary) from one of a limited class of input distributions. The second is a model they call *comparative analysis*, which compares two classes of algorithms A and B, typically where $A \subseteq B$, and typically in order to show that B is a broader, more powerful class than A. For example, by taking A to be the class of on-line algorithms and B to be the class of all off-line algorithms we would end up back at the traditional notion of competitive analysis. For further details, the reader is referred to [38].

In the evaluation model we use in this dissertation, our "adversary" is an optimal adaptive policy that plays by the same rules as the policy P we are trying to evaluate. It knows only probability distributions in advance, and learns the true processing times of jobs as they are instantiated during the course of execution. Letting $ADAPT(I, \pi)$ denote the value obtained by an optimal adaptive policy for an instance I and a realization π, we measure the performance guarantee of P simply as

$$\max_I \frac{\mathbf{E}_\pi[ADAPT(I,\pi)]}{\mathbf{E}_\pi[P(I,\pi)]},$$

This is arguably a very reasonable measure for the quality of a scheduling policy P, and since our adversary is sufficiently limited in power, we will be able to prove $O(1)$ performance guarantees for almost all of the scheduling policies in this dissertation.

As a final note, a completely different way for on-line analysis to potentially creep into our problems is to allow for jobs to have non-zero release times. In this case, we could assume an on-line model where we do not know what jobs will be arriving in the future until the time at which they arrive. However, to simplify matters, we always consider problems with non-zero release times in an off-line model where we know the properties of all jobs in advance.

1.3 Chapter Outline and Summary of Results

The structure of this dissertation is as follows.

In the next chapter we conduct a thorough review of relevant results from the stochastic optimization and stochastic scheduling literature.

Chapters 3 and 4 contain many of the core results of the dissertation. They study the so-called "stochastic knapsack problem", variants of which include the problems $1 \mid p_j \sim stoch, d_j = d \mid \mathbf{E}[\sum w_j \overline{V}_j]$ (start deadline model) and $1 \mid p_j \sim stoch, d_j = d \mid \mathbf{E}[\sum w_j \overline{U}_j]$ (completion deadline model). In Chapter 3 we focus on analyses based on linear programming (LP) relaxations, and in Chapter 4 we discuss techniques that are not explicitly based on linear programming. Chapter 3 contains discussion on a variety of LP-based approaches

Problem	Guarantee	Section
$1 \mid p_j \sim stoch, d_j = d \mid \mathbf{E}[\sum w_j \overline{V}_j]$	4	3.2.1
	2	4.1 (also [58])
$1 \mid p_j \sim stoch \mid \mathbf{E}[\sum w_j \overline{V}_j]$	8	5.1.2
$1 \mid p_j \sim stoch, r_j, d_j = d \mid \mathbf{E}[\sum w_j \overline{V}_j]$	$(8,2)$	5.2
$P \mid p_j \sim stoch, d_j = d \mid \mathbf{E}[\sum w_j \overline{V}_j]$	4	6.2.1, 6.3.1
$R \mid p_j \sim stoch, d_j = d \mid \mathbf{E}[\sum w_j \overline{V}_j]$	4	6.5.1
$R \mid p_j \sim stoch \mid \mathbf{E}[\sum w_j \overline{V}_j]$	8	6.5.2
$1 \mid p_j \sim stoch, d_j = d \mid \mathbf{E}[\sum w_j \overline{U}_j]$	7	3.3.2
	4.572	3.4 (also [58])
	4	4.1 (also [58])
	$3+\varepsilon$	4.2
	$2+\varepsilon$ (*)	4.3
$1 \mid p_j \sim stoch \mid \mathbf{E}[\sum w_j \overline{U}_j]$	$8/(1-\varepsilon)^2$ (**)	5.1.3
$P \mid p_j \sim stoch, d_j = d \mid \mathbf{E}[\sum w_j \overline{U}_j]$	9.831	6.2.2
	4.572	6.3.1
$R \mid p_j \sim stoch, d_j = d \mid \mathbf{E}[\sum w_j \overline{U}_j]$	9.143	6.3.1

Figure 1-5: Results described in this dissertation. The upper half of the table describes results for the start deadline model and the lower half describes results for the completion deadline model. Note that for some problems we have multiple results with different performance guarantees. Results citing [58] are primarily discussed in the Ph.D. dissertation of Jan Vondrák [58]. The result with a guarantee of $(8,2)$ is a pseudo-approximation algorithm. The result (*) only applies if the processing time distribution p_j of each job j satisfies the condition that $(\max p_j)/\mu_j = O(1)$, and the result (**) only applies if $\mu_j \leq \varepsilon$ for all j (note that these are our only results that makes any assumptions about the structure of our processing time distributions).

and shows how simple greedy non-adaptive policies can be used to achieve performance guarantees of 4 in the start deadline model, and ranging from $6 + 4\sqrt{2} < 11.657$ down to 7 in the completion deadline model. Approaches that have the strongest known performance guarantees for the stochastic knapsack problem are primarily discussed in the Ph.D. thesis of Jan Vondrák [58]. These include

- a non-adaptive randomized policy (whose analysis is LP-based) for the completion deadline model with performance guarantee $32/7 < 4.572$, and
- non-adaptive greedy policies with performance guarantees of 2 in the start deadline model and 4 in the completion-deadline model.

We omit details of the former policy, but discuss the latter in Chapter 4, where we show that it delivers a performance guarantee of $2+\varepsilon$ for "small" jobs. By combining this policy with an adaptive $(1+\varepsilon)$-approximate policy for "large" jobs, we will be able to obtain a

performance guarantee of $(3+\varepsilon)$ for any constant $\varepsilon > 0$ in the completion deadline model. We also show, in Chapter 4, how to obtain a performance guarantee of $2+\varepsilon$ if all processing time distributions are "well-behaved" in that for every job j we have $(\max p_j)/\mu_j = O(1)$. Many common distributions satisfy this property — for example, the uniform distribution and all symmetric distributions.

Chapter 5 removes the $d_j = d$ constraint and considers problem variants in which jobs have individual deadlines and release times. We first focus our attention on the start deadline model and use randomized rounding to obtain an 8-approximate non-adaptive policy for $1 \mid p_j \sim stoch \mid \mathbf{E}[\sum w_j \overline{V}_j]$. Adding release times to the picture, we develop an $(8,2)$-pseudo-approximation algorithm for $1 \mid p_j \sim stoch, r_j, d_j = d \mid \mathbf{E}[\sum w_j \overline{V}_j]$. In the completion deadline model our results are substantially weaker, as we can only obtain reasonable performance guarantees when jobs are "small". All of the results in this chapter and the next are LP-based, and depend on the results in Chapter 3 as a prerequisite.

In Chapter 6 we consider parallel machine environments, starting with the problems $P \mid p_j \sim stoch, d_j = d \mid \mathbf{E}[\sum w_j \overline{V}_j]$ and $P \mid p_j \sim stoch, d_j = d \mid \mathbf{E}[\sum w_j \overline{U}_j]$. Recall that there are two types of non-adaptive strategies in this setting: (i) "global" list scheduling based on a single sequence of jobs, and (ii) pre-assignment of disjoint sequences of jobs to each of the machines. For type (i), we show how to achieve a performance guarantee of 4 in the start deadline model and $5.5+2.5\sqrt{3} < 9.831$ in the completion deadline model. For type (ii), we give algorithms with guarantees of 4 (start deadline model) and $32/7 < 4.572$ (completion deadline model). We then consider the unrelated parallel machine model, focusing on policies of type (ii). We give a 4-approximation algorithm for $R \mid p_j \sim stoch, d_j = d \mid \mathbf{E}[\sum w_j \overline{V}_j]$ and a $64/7 < 9.143$-approximation algorithm for $R \mid p_j \sim stoch, d_j = d \mid \mathbf{E}[\sum w_j \overline{U}_j]$. Finally, we allow jobs to have individual deadlines, and we generalize our techniques from the preceding chapter to give an 8-approximation algorithm for $R \mid p_j \sim stoch \mid \mathbf{E}[\sum w_j \overline{V}_j]$. These algorithms are all heavily based upon our existing LP-based algorithms from the single-machine case.

Chapter 7 considers single-machine, common-deadline problems that involve discrete, integer-valued processing time distributions. In this case, one can use dynamic programming (DP) to compute optimal *ordered* adaptive solutions (for a fixed ordering of jobs provided as input, which we called the inflexible ordered adaptive model). If we start with the ordering of jobs suggested by our best non-adaptive algorithms for the stochastic knapsack problem (performance guarantee of 2 in the start deadline model and 4 in the completion deadline model), then we know that the resulting ordered adaptive policy must deliver at least as good of an approximation guarantee. Moreover, we show that for *any* ordering of the input jobs, the ordered adaptive policy computed by our DP algorithm will obtain an expected value at least a 1/8-fraction (in the start deadline model), or a 1/9.5-fraction (in the completion deadline model) of that of an optimal adaptive policy. We do this by demonstrating a approximation algorithms with these guarantees for the fixed set model (recall that the fixed set model is strictly weaker than the ordered adaptive models). Additionally, we show how to apply a technique from signal processing, known as *zero-delay convolution* to speed up the running time of DP algorithms for single-machine stochastic scheduling and other related stochastic optimization problems.

Finally, in Chapter 8 we discuss open problems and potential directions for future research.

2. Literature Review

There do not appear to be any results to date in the literature concerning approximation algorithms for deadline-constrained stochastic scheduling problems, particularly those with arbitrary processing time distributions. However, there are quite a few areas of study to which our work is related. In this chapter, we survey these areas and highlight connections with our models and results. We particularly emphasize related results from stochastic scheduling literature.

2.1 Stochastic Optimization

Optimization problems involving stochastic parameters (e.g., coefficients in the objective function or constraints that are random variables with known distributions) have been studied quite extensively by many different academic communities. One can find many texts on stochastic optimization and stochastic programming — see for example [3].

One reasonable way to classify stochastic optimization problems is based on levels of recourse, or equivalently, on the sequence of when we must make decisions and when random variables in our input are instantiated. At one end of the spectrum, we have problems involving no recourse. One could perhaps describe these using the term *average case analysis*: we know that the parameters in our input are generated by some underlying probability distribution and we want to know the expected value of an optimal solution. For example, if we know that all edge lengths in a graph are uniformly chosen from $[0, 1]$, we might want to know the expected diameter of the graph. In this case, we *first* instantiate all the random parameters in our instance and *then* compute the optimal solution value, Z, by solving the resulting deterministic instance. The optimal solution value Z is a random variable (over the probability space of our random input parameters), and we might want to know properties of Z's distribution, such as its expectation. This model is quite different from the stochastic scheduling models we consider in this dissertation, since in our models we do not know the instantiated processing time of a job until after we actually commit to scheduling the job (recall again our comments regarding on-line models in Section 1.2). For further information on "average case" behavior of the knapsack problem (the deterministic analog of our fundamental problem $1 \mid p_j \sim stoch, d_j = d \mid \mathbf{E}[\sum w_j \overline{U}_j]$), see [6, 39, 24].

Moving away from average case analysis, we find our way to problems involving a single level of recourse. These are probably the most popular problems in the stochastic optimization literature in which there is some limited form of adaptivity present. Here, we must first commit to a partial solution before witnessing the instantiation of any random parameters. Afterwards, we see the results of the random instantiations resulting from our initial decisions and we must react appropriately by completing our solution in an optimal manner, or in a manner that repairs feasibility. For example, in a stochastic bin packing problem (with random item sizes), we might be allowed to purchase bins in advance for a low price, but after finally witnessing the instantiated sizes of our items we may as a recourse need to order additional bins (at a much higher price). Some examples of approximation algorithms for stochastic combinatorial optimization problems with a single stage of recourse include [47, 32, 51].

It is also natural to consider models involving several levels of recourse, where in each level we make some decisions and then witness any random instantiations that result. Our stochastic scheduling problems lie at the extreme in this direction, since they allow for unlimited levels of recourse. Generally, problems that explicitly involve multiple levels of recourse can be quite computationally challenging to solve (this is partly why the single stage recourse models are so popular). The same is true for our problems — for instance, it seems quite difficult to compute an optimal adaptive policy explicitly. For this reason, we tend to compute only non-adaptive solutions and then analyze their performance by comparing against a hypothetical optimal adaptive solution that can take advantage of unlimited levels of recourse.

2.1.1 Chance-Constrained Mathematical Programming

Traditional mathematical programming asks us to compute an optimal solution that is feasible with respect to a set of constraints. A *chance-constrained* mathematical program involves stochastic parameters in its constraints, and asks for an optimal solution that has at least some probability q of being feasible. This is sometimes expressed using a set of *chance constraints*, each of the form $\mathbf{Pr}[\text{constraint } j \text{ violated}] \leq q_j$.

Two recent papers due to Kleinberg et al. [35] and Goel and Indyk[23] consider a chance-constrained formulation of the stochastic knapsack problem (somewhat analogous to our problem $1 \mid p_j \sim stoch, d_j = d \mid \mathbf{E}[\sum w_j \overline{U}_j]$) motivated by the application of assigning potentially bursty data connections to capacitated links in a communication network. Their model is somewhat similar to our fixed set model. They consider items (jobs) with deterministic values and random sizes (processing times), and the objective is to compute a maximum-value set of items whose probability of overflowing the knapsack is at most some specified limit q. Kleinberg et al. consider only the case where item sizes have Bernoulli-type distributions (with only two possible sizes for each item), and for this case they provide a polynomial-time $O(\log 1/q)$-approximation algorithm as well as several pseudo-approximation results. For job sizes that have Poisson or exponential distributions, Goel and Indyk provide a PTAS, and for Bernoulli-distributed items they give a quasi-polynomial approximation scheme whose running time depends polynomially on n and $\log 1/q$.

2.1.2 Markov Decision Processes and Games Against Nature

A Markov process is a stochastic process that moves from state to state according to a matrix of transition probabilities. A Markov Decision Process (MDP) introduces some aspect of control into this model, allowing us to select from a set of valid *actions* in each state. Each (state, action) has an associated vector of transition probabilities, according to which we then move to a new state after an action is selected. By associating a value with each state, MDPs can be used to model a broad range of problems in stochastic optimization and stochastic optimal control, including the problems studied in this dissertation (at least if our probability distributions are discrete). However, in our case the state space is far too large to consider modeling our problems explicitly as MDPs.

A slightly similar notion to that of an MDP is found within the realm of game theory: a "game against nature" [43] is a system where players try to maximize their utility by making moves in alternation with a player called "nature", that behaves according to some specified probabilistic model. We can view the scheduling decisions we make (or the actions taken in an MDP) as our moves, and the resulting job processing time instantiations (or state transitions for an MDP) as the moves for nature. Although this framework also seems too broad to give us any algorithmic advantage, it has proven useful in establishing hardness results. In [14, 15, 58], PSPACE-hardness of certain variants of stochastic knapsack and packing problems are shown to be PSPACE-hard via a reduction from the stochastic satisfiability problem, which behaves like a game against nature as described above.

2.2 Deterministic Scheduling and Packing

The literature on deterministic scheduling is quite extensive. For a comprehensive and well-written introduction to the field of scheduling, the reader is referred to the excellent book of Pinedo [46] (which covers stochastic as well as deterministic results). Other good reference include survey chapters by Kerger, Stein, and Wein [34] and by Hall [28], the latter of which focuses on approximation algorithms for NP-hard deterministic scheduling problems.

2.2.1 Deterministic Packing Problems

The 0/1 knapsack problem $(1 \mid d_j = d \mid \sum w_j \overline{U}_j)$ is the deterministic analog of our simplest stochastic scheduling problems. Ibarra and Kim [31] describe an FPTAS for the knapsack problem, which also gives an FPTAS for the generalization $1 \mid \mid \sum w_j \overline{U}_j$ in which jobs have individual deadlines. By symmetry, this also give us an FPTAS for $1 \mid r_j, d_j = d \mid \sum w_j \overline{U}_j$. Although the start deadline model has not been explicitly considered before in the literature, we can obtain a PTAS for $1 \mid \mid \sum w_j \overline{V}_j$ by converting it into an equivalent instance of $1 \mid \mid \sum w_j \overline{U}_j$ by simply adding p_j to each deadline d_j (thereby converting it from a start deadline into a completion deadline). In general, for all other deterministic problems in the start deadline model, for example "interval scheduling" problems (discussed soon), this same transformation yields an equivalent problem in the completion deadline model. We

also note that NP-hardness of the deterministic problem $1 \mid d_j = d \mid \sum w_j \overline{V}_j$ follows from NP-hardness of the knapsack problem $1 \mid d_j = d \mid \sum w_j \overline{U}_j$. Given any knapsack instance, if we add a "dummy" job of very large value and processing time and then send it to an algorithm for $1 \mid d_j = d \mid \sum w_j \overline{V}_j$, the optimal solution in this case will be to take the dummy job as the "overflowing" job and to fill the remaining time prior to the deadline with an optimal knapsack solution (so we would in effect optimally solve the knapsack problem).

In the parallel machine case, the deterministic problem $Q \mid d_j = d \mid \sum w_j \overline{U}_j$ is known as the *multiple knapsack problem*, and a PTAS for it was recently proposed by Chekuri and Khanna [9]. In the unrelated machine model, $R \mid d_j = d \mid \sum w_j \overline{U}_j$ goes by the name of the *generalized assignment problem*, and a 2-approximation for it is given by Shmoys and Tardos [50].

2.2.2 Interval Scheduling

In the problem $1 \mid r_j \mid \sum w_j \overline{U}_j$, each job j can only be processed within an interval of time $[r_j, d_j]$. This is the simplest variant of what are commonly known as *interval scheduling problems*, in which each job comes with an associated list of disjoint intervals of time during which it may be processed. If $r_j = d_j$ for every job j, we can easily compute an optimal schedule in polynomial time using dynamic programming, even in the stochastic case (here, we ought to use the start deadline model rather than the completion deadline model). For the general deterministic interval scheduling problem, a 2-approximation is given by Bar-Noy et al. [1] using the "local ratio" technique for designing approximation algorithms.

2.2.3 Unknown Deterministic Processing Times

Another interesting model that has some similarity to our stochastic models is an on-line model where processing times are deterministic but unknown. For example, Bender et. al [2] study *stretch scheduling* where the goal is to minimize the maximum or average *stretch* over all jobs. The stretch of a job j is a natural measure defined as $(C_j - r_j)/p_j$ (the ratio of flow time to processing time). In an on-line, preemptive model, they give an algorithm that is $O(1)$-competitive for minimizing average stretch, given that the processing times of all jobs are known to within some constant factor. For completion time-related objectives (C_{max} or $\sum w_j C_j$), any constant-factor approximation algorithm can be applied to the case where we know all processing times to within a constant factor (say, by fixing every processing time at its minimum value), and we will still obtain a constant-factor approximation.

2.3 The Stochastic Knapsack Problem

Stochastic variants of the knapsack problem are of interest to us since the knapsack problem is the deterministic analog of our simplest scheduling problems. Stochastic knapsack problems with deterministic sizes and random values have been studied by several authors [7, 29, 54, 55], all of whom consider the objective of computing a fixed set of items fitting in

the knapsack that has maximum probability of achieving some target value (in this setting, maximizing expected value is a much simpler, albeit still NP-hard, problem since we can just replace every item's value with its expectation and obtain an equivalent deterministic knapsack problem). Several heuristics have been proposed for this variant (e.g. branch-and-bound, preference-order dynamic programming), and adaptivity is not considered by any of the authors.

Another somewhat related variant, known as the stochastic and dynamic knapsack problem [36, 44], involves items that arrive on-line according to some stochastic process — we do not know the exact characteristics of an item until it arrives, at which point in time we must irrevocably decide to either accept the item and process it, or discard the item. We consider this variant briefly in Chapter 7.

Derman et al. [16] consider the adaptive stochastic knapsack problem where multiple copies of items are permitted, as well as the related *knapsack cover* problem in this same setting, where the goal is to cover the capacity of the knapsack with a minimum-cost set of items (for this variant, we *must* allow multiple copies of items since otherwise it might not be possible to cover the knapsack). A prototypical application of the stochastic knapsack cover problem is keeping a machine running for a certain duration, where the machine depends on a critical part (e.g. a light bulb) that periodically fails and must be replaced. The different items correspond to potential replacements, each having a deterministic cost and an uncertain lifetime. Derman et al. provide dynamic programming formulations for these problems, and also prove that if item sizes are exponentially distributed, both problems (as well as the stochastic knapsack problem without multiple copies allowed) are solved by greedily scheduling items according to value or cost divided by expected size.

2.4 Stochastic Scheduling

Stochastic scheduling problems have been studied quite extensively in the literature (although not nearly so much as deterministic scheduling problems). Consult the Ph.D. thesis of Marc Uetz [56] for a comprehensive survey of stochastic scheduling results and problems. We provide a brief summary of some of the most relevant results below.

2.4.1 Stochastic Analogs of Deterministic Algorithms

One of the earliest results in stochastic scheduling dates back to 1966, when Rothkopf [48] gave a simple proof that $1 \mid p_j \sim stoch \mid \mathbf{E}[\sum w_j C_j]$ is optimally solved by greedily ordering according to the *weighted shortest expected processing time* (WSEPT) policy:

$$\frac{w_1}{\mathbf{E}[p_1]} \geq \frac{w_2}{\mathbf{E}[p_2]} \geq \cdots \geq \frac{w_n}{\mathbf{E}[p_n]}.$$

This policy is so-named because if all weights are equal, it becomes the *shortest expected processing time* (SEPT) policy where we always execute the job having shortest expected processing time first (not too surprisingly, there is also an analogous LEPT policy where

we execute the job with longest expected processing time first).

Rothkopf's result is particularly nice considering that the deterministic problem $1 \mid\mid \sum w_j C_j$ is optimally solved by scheduling jobs in the order $\frac{w_1}{p_1} \geq \frac{w_2}{p_2} \geq \ldots \geq \frac{w_n}{p_n}$ (this is called the *weighted shortest processing time* (WSPT) policy, or also "Smith's rule" [53]). Therefore, to solve the stochastic variant, we simply replace each job with its expectation and then solve the resulting deterministic variant. Stochastic problems that can be optimally solved via this sort of trivial reduction to an equivalent deterministic problem are rare. Another example is the problem $1 \mid p_j \sim stoch \mid \mathbf{E}[L_{max}]$, which is optimally solved just like its deterministic counterpart by ordering jobs so that $d_1 \leq d_2 \leq \ldots \leq d_n$.

2.4.2 Special Classes of Distributions

A significant majority of all stochastic scheduling results to date consider only special types of processing time distributions, most notably jobs with exponential processing times. We summarize some of these results below, saving for later the results that deal with deadline-constrained problems.

Bruno, Downey, and Fredrickson [5] show that $P \mid p_j \sim stoch \mid \mathbf{E}[C_{max}]$ with exponential distributions is optimally solved by list scheduling according to the LEPT policy, and that $P \mid p_j \sim stoch \mid \mathbf{E}[\sum C_j]$ with exponential distributions is optimally solved by list scheduling according to the SEPT policy (the latter result is also due to Glazebrook [22]). These results hold even if preemption is allowed. Due to the "memoryless" nature of the exponential distribution, the expected remaining processing time of a job conditioned on the fact that it has already been executing for some length of time is the same as its original expected processing time. Therefore, a policy that performs list scheduling based only on expected processing times will be unaffected by the availability of preemption. The LEPT result here is particularly noteworthy since the LPT policy[1] is not necessarily optimal for the deterministic problem $P \mid\mid C_{max}$ (although a well-known result of Graham [25] states that LPT is a 4/3-approximation for this problem). In this situation the stochastic variant is actually easier to solve, primarily due to the fact that we assume exponential distributions.

The results above are generalized somewhat by Weber [60], who shows that LEPT is optimal for $P \mid p_j \sim stoch \mid \mathbf{E}[C_{max}]$ if processing time distributions have decreasing failure rates, and SEPT is optimal for $P \mid p_j \sim stoch \mid \mathbf{E}[\sum C_j]$ as long as processing time distributions have increasing failure rates. The *failure rate* (also known as *hazard rate*) of a distribution $f(x)$ is given by $\frac{f(x)}{1-F(x)}$, where F denotes the cumulative distribution of f. The exponential distribution is the only distribution with constant failure rate. If p_j has a decreasing failure rate, then $\mathbf{E}[p_j - \tau \mid p_j \geq \tau]$ increases as τ increases. Similarly, for increasing failure rate, the expected remaining processing time decreases over time[2] as we begin to process job j. As a result, the list scheduling approaches above will never want to preempt jobs, since the jobs currently being processed will always remain ahead of any unprocessed jobs on the list.

[1] SPT and LPT are the natural deterministic analogs of SEPT and LEPT.

[2] Distributions for which expected processing time decreases over time (i.e., $\mathbf{E}[x - \tau \mid x \geq \tau] \leq \mathbf{E}[x]$) are called NBUE (new better than used in expectation) distributions.

Weiss and Pinedo [62] also generalize the results above to the case of parallel machines with different speeds, assuming exponential processing times and availability of preemption. Defining the SEPT policy as taking the job with shortest expected processing time and assigning it to the fastest machine, the job with second-shortest expected processing time and assigning it to the second-fastest machine, and so on, they show that this policy is optimal for minimizing $\mathbf{E}[C_{max}]$. Similarly, if we define LEPT as taking the longest job and assigning it to the slowest machine, and so on, they show that this is an optimal policy for minimizing $\mathbf{E}[\sum C_j]$. Note that these policies require preemption, since the completion of any job may require us to re-shuffle the entire current assignment of jobs to machines. For more details on general conditions under which list scheduling policies are optimal for these objectives, the reader is encouraged to consult Kampke [33].

2.4.3 Minimizing Sum of Weighted Completion Times

Some of the strongest approximation results for stochastic scheduling problems to date consider the objective of minimizing $\mathbf{E}[\sum w_j C_j]$. Möhring et al. [41] consider the problems $1 \mid prec \mid \mathbf{E}[\sum w_j C_j]$ and $1 \mid prec, r_j \mid \mathbf{E}[\sum w_j C_j]$, and describe approximation algorithms with performance guarantees of 2 and 3. It is worth noting that these are among the few approximation algorithm results in the stochastic scheduling literature that apply to arbitrary processing time distributions. In the parallel machine case, for $P \mid \mid \mathbf{E}[\sum w_j C_j]$ they show that global list scheduling according to the WSEPT policy yields a performance guarantee of $1 + \frac{m-1}{2m}(\Delta + 1)$, where Δ is an upper bound on the squared coefficients of variation of all job processing times:

$$\max_{j \in [n]} \frac{\mathbf{Var}[p_j]}{\mathbf{E}[p_j]^2} \leq \Delta.$$

If $\Delta \leq 1$ (this is the case for many common distributions such as the uniform, exponential, and Erlang distributions), the performance guarantee simplifies to $2 - \frac{1}{m}$. For the problem $P \mid r_j \mid \mathbf{E}[\sum w_j C_j]$, they provide an algorithm with guarantee $3 - \frac{1}{m} + \max\{1, \frac{m-1}{m}\Delta\}$ (which simplifies to $4 - \frac{1}{m}$ if $\Delta \leq 1$). Megow et al. [40] extend this result to an on-line model (the result of Möhring et al. for release times assumes an off-line model) and improve the performance guarantee to roughly 3.62 for processing times with NBUE distributions.

Skutella and Uetz [52] extend the results above involving precedence constraints to the parallel machine case. They obtain a performance guarantee of $(1+\varepsilon)(1+\frac{m-1}{m\varepsilon}+\max\{1, \frac{m-1}{m}\Delta\})$ for $P \mid prec \mid \mathbf{E}[\sum w_j C_j]$ and a performance guarantee of $(1 + \varepsilon)(1 + \frac{1}{\varepsilon} + \max\{1, \frac{m-1}{m}\Delta\})$ for $P \mid prec, r_j \mid \mathbf{E}[\sum w_j C_j]$ (here, ε is an arbitrary constant and the release time results assume an off-line model). For $\Delta \leq 1$, these performance guarantees can be simplified to $3 + 2\sqrt{2} - (1 + \sqrt{2})/m$ and $3 + 2\sqrt{2}$ respectively.

These results described above are some of the only other results in the domain of stochastic scheduling, besides our own, that use linear programming relaxations to bound the expected value obtained by an optimal adaptive policy.

2.4.4 Deadline-Constrained Stochastic Scheduling

In the literature to date, stochastic scheduling results involving deadlines tend to focus on identifying conditions under which optimal policies can be computed (e.g., for special classes of processing time distributions) rather than on approximation algorithms. We have already mentioned the results of Derman et al. [16] for the stochastic knapsack problem — these show that the WSEPT policy optimally solves $1 \mid p_j \sim stoch \mid \mathbf{E}[\sum w_j \overline{U}_j]$ when processing times are exponential. Pinedo [45] extends this result to show optimality even when job deadlines are independent, identically distributed (i.i.d.) random variables or when all jobs share a common, random deadline.

Emmons and Pinedo [18] give several optimality results. They show optimality of WSEPT in a non-preemptive, identical parallel machine environment when job processing times are i.i.d. and job deadlines are either i.i.d. or all equal to a common random deadline. In addition, they show how to use bipartite assignment to optimally solve $Q \mid p_j = 1, d_j \sim stoch \mid \mathbf{E}[\sum w_j \overline{U}_j]$ where the d_j's have arbitrary distributions (not even necessarily independent), and also $Q \mid p_j \sim stoch, d_j \sim stoch \mid \mathbf{E}[\sum w_j \overline{U}_j]$, where the p_j's are exponential each with expected unit duration, and the d_j's are exponential with arbitrary expectations (not even necessarily independent).

Chang et. al [8] consider preemptive scheduling on parallel machines where jobs have exponentially-distributed processing times. They show that if the WSEPT and LEPT orderings happen to coincide, then global list scheduling in this order optimally solves $P \mid p_j \sim stoch, r_j, d_j \sim stoch \mid \mathbf{E}[\sum w_j \overline{U}_j]$, as long as the d_j's are i.i.d. They discuss similar results for the objective $\mathbf{E}[\sum w_j T_j]$. Pinedo [45] shows that if $\mathbf{E}[p_1] \leq \mathbf{E}[p_2] \leq \ldots \leq \mathbf{E}[p_n]$ and $w_1 \geq w_2 \geq \ldots \geq w_n$, then list scheduling according to this ordering (SEPT) optimally solves $P \mid p_j \sim stoch, d_j = d \sim stoch \mid \mathbf{E}[\sum w_j \overline{U}_j]$ if (i) processing times are independent and exponentially distributed, and (ii) if the common deadline d has a concave distribution function.

The stochastic analog of the problem $1 \mid\mid \sum \overline{U}_j$ is studied by van den Akker and Hoogeveen [57], who show that for some types of processing time distributions, a well-known optimal $O(n \log n)$ algorithm of Moore [42] can be extended to the stochastic case. Rather than focusing on the objective $\mathbf{E}[\sum \overline{U}_j]$ directly, they define a job as being "stochastically on time" if its probability of completion prior to its deadline is above some specified threshold, and they consider maximizing the expected number of jobs that are stochastically on time. For the objective $\mathbf{E}[\sum w_j \overline{U}_j]$, they show how to compute an optimal ordered policy using dynamic programming (as we do in Chapter 7), but only for certain classes of processing time distributions.

3. One Machine with Common Deadlines: LP-Based Results

This chapter develops many of the fundamental results and techniques that we will build upon throughout the remainder of the dissertation. It focuses entirely on the so-called "stochastic knapsack" problem, where jobs are scheduled on a single machine and share a common deadline. All of the problems studied later in the dissertation contain the stochastic knapsack problem as a special case. By appropriate scaling of processing times, we assume without loss of generality that all jobs share a common deadline at time $t = 1$. Therefore, we can write the stochastic knapsack problem as either

- $1 \mid p_j \sim stoch, d_j = 1 \mid \mathbf{E}[\sum w_j \overline{V}_j]$ (in the start deadline model), or
- $1 \mid p_j \sim stoch, d_j = 1 \mid \mathbf{E}[\sum w_j \overline{U}_j]$ (in the completion deadline model).

We study both the start deadline and completion deadline models in this chapter, and in general our results in the start deadline model will be both stronger and simpler than those in the completion deadline model.

In this chapter our analyses all share a common theme in that they are all strongly based on linear programming. In the next chapter, we present stronger results whose analyses are not explicitly based on linear programming (although there do appear to be ways to cast these results in terms of linear programming as well). It should be noted that although the results in this chapter are weaker than those in the next chapter for the case of the stochastic knapsack problem, at the moment only these results seem to extend readily to more complicated scheduling problems such as those with individual deadline constraints (Chapter 5) and parallel machine environments (Chapter 6). These two future chapters (5 and 6) are based on the material in this chapter, and not Chapter 4.

3.1 Bounding an Optimal Adaptive Policy

A feature that is common to most approximation algorithms is some means of obtaining a bound on the value of an optimal solution. In our case, this step is somewhat interesting (and in need of special techniques beyond those typically applied in the literature) since

the solution we are trying to bound — an optimal adaptive policy — is in the form of a decision tree. Since the decision tree for an optimal policy might be quite complex, how does one obtain a bound on the maximum expected value such a policy may obtain?

As an example, suppose we have an unlimited quantity of jobs whose processing times are either 0 (with probability $1-\varepsilon$) or 1 (with probability ε). We assign each job a value equal to its expected processing time, ε. Since we have one unit of time prior to our deadline and each job takes ε units of expected time, one might expect that we can schedule roughly one unit of value. However, it takes 2 jobs of processing time 1 to exceed the deadline, so we actually expect to schedule $2/\varepsilon$ jobs in the start deadline model and $2/\varepsilon - 1$ jobs in the completion deadline model. Since each job has a value of ε, we obtain expected values of 2 and $2-\varepsilon$ in these respective models, and this is nearly twice what we could hope to achieve in the deterministic case. As we shall see in a moment, this example is essentially tight in that the expected value of any adaptive policy can be no larger than the optimal value one can "fractional" schedule in a deterministic instance in which every processing time p_j is replaced with its mean truncated processing time μ_j, and in which our deadline is inflated to time $t = 2$.

The following lemma is a key ingredient in this analysis, and it applies in both the start deadline and completion deadline models.

Lemma 1. *Fix any adaptive policy P. Let J denote the (random) set of jobs that P schedules (i.e., starts) no later than the deadline*[1]*. In the start deadline model, J is the set of jobs actually scheduled by P, and in the completion deadline model J is the set of jobs successfully scheduled plus the single job that completes later than the deadline. Letting $\mu(J) = \sum_{j \in J} \mu_j$, we then have $\mathbf{E}[\mu(J)] \leq 2$.*

Proof. Suppose P finds itself in a situation with a deadline d units of time into the future and a set R of remaining jobs. Let $J(d, R)$ denote the (random) set of jobs that P attempts to schedule from this point on. For any set of jobs R and any random variable $t \leq 1$ independent of the processing times of jobs in R, we argue that $\mathbf{E}[\mu(J(t, R))] \leq \mathbf{E}[t] + 1$ using induction on $|R|$, taking $|R| = 0$ as a trivial base case. The lemma follows if we set $t = 1$ and $R = [n]$. Suppose, now, that P is faced with a deadline t units of time into the future (a random variable) and has a set R of available remaining jobs. If P happens to schedule job $j \in R$ next, then

$$\begin{aligned}
\mathbf{E}[\mu(J(t,R))] &= \mu_j + \mathbf{E}[\mu(J(t-p_j, R\backslash\{j\})) \mid p_j \leq 1]\mathbf{Pr}[p_j \leq 1] \\
&= \mu_j + \mathbf{E}[\mu(J(t-\min(p_j,1), R\backslash\{j\})) \mid p_j \leq 1]\mathbf{Pr}[p_j \leq 1] \\
&= \mu_j + \mathbf{E}[\mu(J(t-\min(p_j,1), R\backslash\{j\}))] \\
&\leq \mu_j + \mathbf{E}[t-\min(p_j,1)] + 1 \quad \text{(by induction)} \\
&= \mathbf{E}[t] + 1,
\end{aligned}$$

[1] A "random set" by itself is not a particularly well-defined notion, so we should technically be considering the random incidence vector of such a set. However, since the notion of a property of a random set like $\mathbf{E}[\mu(J)]$ is fairly clear, we will continue for simplicity of exposition to use the more informal "random set" term throughout this work.

and this completes the proof, since it holds for every job $j \in R$. □

Lemma 1 is tight according to the "Bernoulli" example described above: given an unlimited number of jobs with processing times of 0 (probability $1 - \varepsilon$) or 1 (probability ε), we have $\mathbf{E}[\mu(J)] = 2$.

3.1.1 Analysis Using Martingales

Martingale theory is a branch of probability theory that gives us a powerful and general set of tools with which we can analyze and bound adaptive policies. A *martingale* is a stochastic process $Z_0, Z_1, Z_2, \ldots$ satisfying

$$\mathbf{E}[Z_j \mid Z_0, Z_1, \ldots Z_{j-1}] = Z_{j-1}. \tag{3.1}$$

Equivalently, a martingale $Z_0, Z_1, Z_2, \ldots$ is sometimes defined with respect to a stochastic process $X_0, X_1, X_2, \ldots$, where Z_n is a function of $X_0 \ldots X_n$ and where

$$\mathbf{E}[Z_j \mid X_0, X_1, \ldots X_{j-1}] = Z_{j-1}. \tag{3.2}$$

The simplicity and lack of strong assumptions (e.g., independence, bounded variance) in this definition give martingales sufficient flexibility to model quite a range of useful stochastic processes. As we shall see in a moment, this includes the ability to model an adaptive scheduling policy. Martingales have been studied quite extensively in the literature in the past few decades, due in large part to the initial work of Doob [17]. Historically, martingales originated as a means of describing "unbiased" games of chance, where one's expected earnings after any series of bets are precisely zero. As a trivial example, suppose you flip a fair coin and gain one dollar if it comes up heads ($Z_j = Z_{j-1} + 1$), and lose one dollar if it comes up tails ($Z_j = Z_{j-1} - 1$). In this case, $Z_0, Z_1, Z_2, \ldots$ satisfies the martingale condition since we expect to "break even" at each step. As a consequence of this behavior, all martingales satisfy the following well-known property, which is readily proved by straightforward induction on n.

Lemma 2. *If $Z_0, Z_1, Z_2, \ldots$ is a martingale, then $\mathbf{E}[Z_n] = \mathbf{E}[Z_0]$ for every $n \geq 0$.*

Let us try to cast the stochastic knapsack problem in the "gambling" framework above. Suppose our goal is to maximize $\mathbf{E}[\mu(J)]$, where J denotes the set of jobs we attempt to schedule. To do this, we must select a series of jobs to schedule. At each step, if we choose job j we receive a "reward" of μ_j toward our objective, but we must also "pay" $\min(p_j, 1)$ (a random quantity). We would like to receive as much reward as possible before we pay more than one unit total (at which point we exceed the deadline). Note that this game is unbiased in the sense that the expected reward at each step is the same as the expected amount we pay. More formally, let us fix an adaptive policy P, and let S_j denote the (random) set of the first j jobs that P attempts to schedule. Once we schedule some job j that completes later than the deadline and stop scheduling new jobs, we adopt the

convention that $S_j = S_{j+1} = S_{j+2} = \ldots = S_n$. We now define

$$Z_j = \sum_{i \in S_j} X_i,$$

where $X_i = \mu_i - \min(p_i, 1)$. It is fairly easy to see that $Z_0, Z_1, Z_2, \ldots$ is a martingale with respect to the stochastic process $X_0, X_1, X_2, \ldots$ since it satisfies (3.2),

$$\mathbf{E}[Z_j \mid X_0, X_1, \ldots X_{j-1}] = \mathbf{E}[Z_{j-1} + X_j \mid X_0, X_1, \ldots X_{j-1}] = Z_{j-1},$$

due to the fact that $\mathbf{E}[X_j] = 0$ (this is true both in the initial phase where we schedule jobs as well as in the "stopped" phase where we cease to schedule jobs).

Another way to write Z_j is as $Z_j = \mu(S_j) - \tau(S_j)$, where

$$\tau(S_j) = \sum_{i \in S_j} \min(p_i, 1).$$

Note that $\tau(S_j) \leq 2$ for all j since we stop scheduling immediately after some job i (with $\min(p_i, 1) \leq 1$) completes later than our deadline of $d = 1$. According to Lemma 2, we therefore have $\mathbf{E}[Z_n] = \mathbf{E}[Z_0] = 0$, so $\mathbf{E}[\mu(J)] = \mathbf{E}[\mu(S_n)] = \mathbf{E}[\tau(S_n)] \leq 2$ where J denotes the final (random) set of jobs our policy P attempts to schedule. This gives us an alternative proof of Lemma 1. Actually, one can view our original proof as being essentially equivalent to this one, substituting an inductive argument for Lemma 2.

If the support of every processing time distribution lies in some small bounded range $[0, b]$, then we can strengthen the bound above to $\mathbf{E}[\mu(J)] \leq 1 + b$ since $\mathbf{E}[\tau(S_n)] \leq 1 + b$. We use this fact in Section 4.3 when we study special "well-behaved" classes of distributions.

3.1.2 Fractional Relaxations for the Start Deadline Model

Perhaps the most common technique used to bound the optimal solution to a hard problem is to "relax" it to an easier problem (usually a linear program) by removing integrality constraints. In this section we show that this technique can also be applied, albeit somewhat indirectly, to the stochastic knapsack problem. We show how to bound the optimal value obtained by an adaptive policy for the *stochastic* knapsack problem in terms of the value of a fractional relaxation for a corresponding *deterministic* knapsack problem.

Suppose for a moment that processing times $p_1 \ldots p_n$ are deterministic. In the completion deadline model, this gives us precisely a 0/1 knapsack problem, which we can write as an integer program and relax to a linear program. In the start deadline model, we can still express the problem $1 \mid d_j = 1 \mid \mathbf{E}[\sum w_j \overline{V}_j]$ as an integer program, since in any feasible solution the removal of one job must leave us with a set of jobs whose processing times sum to no more than 1. Letting x be the incidence vector for the set of jobs in our solution and y be the incidence vector for the single job to be removed, we arrive at the following integer program:

$$\begin{array}{rcl} OPT & = & \max \sum_{j=1}^{n} x_j w_j \\ & & \sum_{j=1}^{n} (x_j - y_j) p_j \leq 1 \\ & & \sum_{j=1}^{n} y_j \leq 1 \\ \forall j \in [n] & : & x_j \geq y_j \\ \forall j \in [n] & : & x_j \in \{0,1\}, y_j \in \{0,1\}. \end{array}$$

By relaxing this to a linear program, we can obtain an upper bound on OPT. However, a simpler linear program also suffices for this task. In fact, we can use the same linear programming relaxation as in the completion deadline case, which we parameterize on the deadline d:

$$\begin{array}{rcl} LP(d) & = & \max \sum_{j=1}^{n} x_j w_j \\ & & \sum_{j=1}^{n} x_j \min(p_j, 1) \leq d \\ \forall j \in [n] & : & 0 \leq x_j \leq 1 \end{array}$$

We claim that $OPT \leq LP(2)$. This follows from the fact that any optimal solution x^* is feasible for $LP(2)$ since only one job j (with $\min(p_j, 1) \leq 1$) is allowed to overflow the deadline of $d = 1$. Note that since $LP(1)$ is a fractional relaxation of the knapsack integer program, the optimal solution to a deterministic instance in the completion deadline model is bounded by $LP(1)$.

Moving to the stochastic case, consider the following natural generalization of $LP(d)$:

$$\begin{array}{rcl} \Phi(d) & = & \max \sum_{j=1}^{n} x_j w_j \\ & & \sum_{j=1}^{n} x_j \mathbf{E}[\min(p_j, 1)] \leq d \\ \forall j \in [n] & : & 0 \leq x_j \leq 1 \end{array}$$

Recalling that $\mu_j = \mathbf{E}[\min(p_j, 1)]$, the packing constraint above can be written more concisely as $\sum_j x_j \mu_j \leq d$. It is not too difficult to see that this particular function $\Phi(d)$ is concave. However, since throughout this dissertation we will be utilizing the concavity of this function and several of its more complicated generalizations (all of which are also concave), let us briefly mention a well-known and useful lemma on concavity that will apply to all of our future linear programming relaxations.

Lemma 3. *Consider any function $f(\lambda)$ of the form*

$$f(\lambda) = \max \{c^T x : Ax \leq \lambda b, Cx \leq d\}$$

where x is an n-dimensional vector. Then f is concave over the domain over which $f(\lambda)$ is finite.

Proof. By definition, f is concave if we can show that $\alpha f(\lambda_1) + (1-\alpha)f(\lambda_2) \leq f(\alpha\lambda_1 + (1-\alpha)\lambda_2)$ for any $\alpha \in [0,1]$ and any nonnegative values of λ_1 and λ_2. Therefore, consider any setting for α, λ_1, and λ_2. Let x_1 and x_2 be optimal solutions to the LPs for $f(\lambda_1)$ and $f(\lambda_2)$, and define $x^* = \alpha x_1 + (1-\alpha)x_2$. It is easy to see that x^* is feasible for the LP for $f(\alpha\lambda_1 + (1-\alpha)\lambda_2)$, and moreover, the objective value for x^* for this LP will be precisely equal to $\alpha f(\lambda_1) + (1-\alpha)f(\lambda_2)$. The lemma follows. □

Just as we can bound an optimal solution by $LP(2)$ in the deterministic case, we can bound the expected value obtained by an optimal adaptive policy by $\Phi(2)$ in the stochastic case:

Theorem 1. *Let ADAPT denote the expected value obtained by an optimal adaptive policy. Then* $ADAPT \leq \Phi(2)$ *in both the start deadline and completion deadline models.*

Proof. Fix any adaptive policy P. Let J denote the (random) set of jobs that P attempts to schedule. Denote by $w(J)$ the total value of the jobs in J and by $\mu(J)$ the sum of μ_j for all $j \in J$. Note $w(S) \leq \Phi(\mu(S))$ for any deterministic set S of jobs, since the incidence vector $x(S)$ for the jobs in S is a feasible solution to the linear program for $\Phi(\mu(S))$. Now,

$$\begin{aligned} ADAPT &\leq \mathbf{E}[w(J)] \\ &\leq \mathbf{E}[\Phi(\mu(J))] \\ &\leq \Phi(\mathbf{E}[\mu(J)]) \quad \text{(Jensen's inequality)} \\ &\leq \Phi(2). \end{aligned}$$

The last step follows from monotonicity of $\Phi(d)$ and Lemma 1. □

3.1.3 Fractional Relaxations for the Completion Deadline Model

Although $ADAPT \leq \Phi(2)$ holds in the completion deadline model, this bound is not tight enough to allow us to prove useful approximation bounds. In fact, it can be arbitrarily loose: consider a deterministic instance with a single job j with $p_j = 1 + \varepsilon$ and $w_j = 1$. Here, $ADAPT = 0$ but $\Phi(2) \approx 1$. For exactly the same reason, although $LP(1)$ is an upper bound on the optimal solution to a deterministic knapsack problem, it is an arbitrarily loose upper bound. Fortunately, we can remedy these problems with a slight addition to the objective function that prevents us from getting too much "credit" for jobs of processing time larger than 1:

$LP^*(d)$	=	$\max \sum_{j=1}^{n} x_j w_j I(p_j, 1)$	$\Phi^*(d)$	=	$\max \sum_{j=1}^{n} x_j w_j \mathbf{Pr}[p_j \leq 1]$
		$\sum_{j=1}^{n} x_j \min(p_j, 1) \leq d$			$\sum_{j=1}^{n} x_j \mathbf{E}[\min(p_j, 1)] \leq d$
$\forall j \in [n]$	:	$0 \leq x_j \leq 1$	$\forall j \in [n]$	:	$0 \leq x_j \leq 1$

The indicator function $I(x, y)$ takes the value 1 if $x \leq y$, and 0 otherwise. With this included in the objective, the integrality gap for $LP^*(1)$ for the deterministic knapsack problem drops

to 2. To see the integrality gap really can be arbitrarily close to 2, consider an instance with multiple jobs of unit value and processing time $1/2 + \varepsilon$. In this case, $OPT = 1$ while $LP^*(1) \approx 2$. Finally, note that $LP^*(d) \leq LP(d)$ and $\Phi^*(d) \leq \Phi(d)$.

Theorem 2. *Let $ADAPT$ denote the expected value obtained by an optimal adaptive policy. Then $ADAPT \leq \Phi^*(2)$ in the completion deadline model.*

Proof. Fix an optimal adaptive policy P. Let x_j denote the probability that P attempts to schedule job j no later than the deadline, and let y_j denote the probability that P succeeds in scheduling job j so it completes prior to the deadline. Regardless of when we attempt to schedule j, there is at most a probability of $\mathbf{Pr}[p_j \leq 1]$ that it will complete by the deadline. Therefore, we have $y_j \leq x_j \mathbf{Pr}[p_j \leq 1]$, and the expected value obtained by P is

$$ADAPT = \sum_{j=1}^{n} w_j y_j \leq \sum_{j=1}^{n} w_j x_j \mathbf{Pr}[p_j \leq 1],$$

which is precisely the objective of $\Phi^*(\cdot)$. Letting J denote the random set of jobs that P attempts to schedule, we use Lemma 1 to obtain

$$\sum_{j=1}^{n} x_j \mu_j = \mathbf{E}[\mu(J)] \leq 2.$$

The vector x is therefore have a feasible solution to the linear program for $\Phi^*(2)$ of objective value at least $ADAPT$, so $ADAPT \leq \Phi^*(2)$. □

The quantity $w_j \mathbf{Pr}[p_j \leq 1]$ is something we will commonly encounter in the completion deadline model, more commonly in fact that w_j by itself. We call this the *adjusted value* of job j, and denote it by $w'_j := w_j \mathbf{Pr}[p_j \leq d_j] = w_j \mathbf{Pr}[p_j \leq 1]$. In terms of adjusted values, we can write $\Phi^*(\cdot)$ more concisely as

$$\Phi^*(d) = \max \left\{ \sum_{j=1}^{n} x_j w'_j : \sum_{j=1}^{n} x_j \mu_j \leq d, 0 \leq x_j \leq 1 \right\},$$

which differs from $\Phi(d)$ only in the use of w'_j in the objective rather than w_j.

3.1.4 Limitations of LP-Based Analysis

We have now established the bounds $ADAPT \leq \Phi(2)$ and $ADAPT \leq \Phi^*(2)$ (in the completion deadline model). These bounds are fairly good, but it is worth noting that there are inherent limitations on the approximation guarantees we can prove for any policy whose expected value we compare against $\Phi(2)$ or $\Phi^*(2)$. Consider a deterministic instance with four jobs having $p_j = 1/2 + \varepsilon$ and $w_j = 1$. In the start deadline model, $ADAPT = 2$ while $\Phi(2) \approx 4$. In the completion deadline model, $ADAPT = 1$ while $\Phi^*(2) \approx 4$. Therefore, using only the bounds above, we should not expect any of our LP-based analyses to give us

approximation guarantees better than 2 in the start deadline model, and 4 in the completion deadline model. On the positive side, we will show later in this chapter how to obtain a performance guarantee of 4 in the start deadline model and $32/7 < 4.572$ in the completion deadline model. Both of these can be improved using somewhat stronger methods of analysis (not explicitly based on the LPs introduced above) — in Chapter 4, we derive a performance guarantee of 2 for the start deadline model and $3 + \varepsilon$ for the completion deadline model.

3.1.5 Solving the Linear Programs

It is well-known that the linear programs for $LP(d)$, $LP^*(d)$, $\Phi(d)$ and $\Phi^*(d)$ can all be solved using greedy algorithms in $O(n \log n)$ time. These greedy algorithms select jobs one by one to include in our solution in decreasing order of "value density":

- For $LP(d)$: $\frac{w_1}{\min(p_1,1)} \geq \frac{w_2}{\min(p_2,1)} \geq \ldots \geq \frac{w_n}{\min(p_n,1)}$.
- For $LP^*(d)$: $\frac{w_1 I(p_1,1)}{\min(p_1,1)} \geq \frac{w_2 I(p_2,1)}{\min(p_2,1)} \geq \ldots \geq \frac{w_n I(p_n,1)}{\min(p_n,1)}$.
- For $\Phi(d)$: $\frac{w_1}{\mu_1} \geq \frac{w_2}{\mu_2} \geq \ldots \geq \frac{w_n}{\mu_n}$.
- For $\Phi^*(d)$: $\frac{w'_1}{\mu_1} \geq \frac{w'_2}{\mu_2} \geq \ldots \geq \frac{w'_n}{\mu_n}$.

For each job j selected in sequence by our greedy algorithm we increase x_j in our LP solution as much as possible. This usually means increasing x_j to 1, except for the last job we select (when we reach the deadline), which may only be fractionally included. Therefore, we can always find an optimal solution to the LPs above in which $0 < x_j < 1$ for at most one job j.

In the deterministic case ($LP(d)$ and $LP^*(d)$), the greedy algorithms above are known in the scheduling literature as WSPT (weighted shortest processing time) policies, since if all values $w_1 \ldots w_n$ are equal, the greedy orderings above tell us to schedule the job of shortest processing time first. In the stochastic case ($\Phi(d)$ and $\Phi^*(d)$), they are known as WSEPT (weighted shortest expected processing time) policies. When we speak of "the" WSEPT policy, the actual ordering of jobs depends on whether we are in the start deadline or completion deadline model. In the start deadline model, where $\Phi(\cdot)$ is our LP relaxation of choice, WSEPT stands for the ordering $\frac{w_1}{\mu_1} \geq \frac{w_2}{\mu_2} \geq \ldots \geq \frac{w_n}{\mu_n}$. In the completion deadline model, we take the ordering $\frac{w'_1}{\mu_1} \geq \frac{w'_2}{\mu_2} \geq \ldots \geq \frac{w'_n}{\mu_n}$ as the WSEPT ordering.

3.1.6 An Interesting Special Case: Zero Capacity

Our bounds of $ADAPT \leq \Phi(2)$ and $ADAPT \leq \Phi^*(2)$ (completion deadline model) assume that $d = 1$, which is an assumption we can make without loss of generality by scaling processing times. Without rescaling, the bounds are $ADAPT \leq \Phi(2d)$ and $ADAPT \leq \Phi^*(2d)$ (completion deadline model), using the following "unscaled" versions of our linear programs:

$\Phi(\lambda)$	$=$	$\max \sum_{j=1}^{n} x_j w_j$	$\Phi^*(\lambda)$	$=$	$\max \sum_{j=1}^{n} x_j w_j \mathbf{Pr}[p_j \leq d]$
		$\sum_{j=1}^{n} x_j \mathbf{E}[\min(p_j, d)] \leq \lambda$			$\sum_{j=1}^{n} x_j \mathbf{E}[\min(p_j, d)] \leq \lambda$
$\forall j \in [n]$	:	$0 \leq x_j \leq 1$	$\forall j \in [n]$	:	$0 \leq x_j \leq 1$

This holds for all deadlines $d \geq 0$, even $d = 0$. However, the bound for $d = 0$ is particularly weak since the packing constraints above deteriorate and allow us to set $x_j = 1$ for all j.

For $d = 0$, it turns out that in both the start deadline and completion deadline model we can compute an optimal policy in polynomial time, and that this policy will be non-adaptive. In the completion deadline model this is analogous to a simple greedy problem known as the *quiz problem*. In the quiz problem, we are given a set of n questions, each with an associated value and probability that we answer correctly, and we must decide on an optimal ordering in which to receive the questions for an oral exam, so that we maximize our expected value. The exam proceeds until the first incorrect answer, and no value is obtained from the question answered incorrectly.

Theorem 3. *In the start deadline model with deadline $d = 0$, an optimal policy is to non-adaptively schedule jobs according to the ordering*

$$\frac{w_1}{\mathbf{Pr}[p_1 > 0]} \geq \frac{w_2}{\mathbf{Pr}[p_2 > 0]} \geq \ldots \geq \frac{w_n}{\mathbf{Pr}[p_n > 0]}. \tag{3.3}$$

In the completion deadline model with deadline $d = 0$, the ordering

$$\frac{w_1 \mathbf{Pr}[p_1 = 0]}{\mathbf{Pr}[p_1 > 0]} \geq \frac{w_2 \mathbf{Pr}[p_2 = 0]}{\mathbf{Pr}[p_2 > 0]} \geq \ldots \geq \frac{w_n \mathbf{Pr}[p_n = 0]}{\mathbf{Pr}[p_n > 0]} \tag{3.4}$$

is optimal. Recall that we assume that $\mathbf{Pr}[p_j > 0] \neq 0$ for all jobs j, since any job j for which $\mathbf{Pr}[p_j > 0] = 0$ can be immediately scheduled with no risk at the beginning of time.

Proof. The theorem follows from a straightforward greedy exchange argument. For notational simplicity, let $\pi_j = \mathbf{Pr}[p_j = 0]$. If we schedule jobs in order of their indices $1, 2, \ldots, n$, our total expected value in the start deadline model is

$$\begin{aligned} \mathbf{E}[\text{total value}] &= \sum_{j=1}^{n} w_j \mathbf{Pr}[\text{jobs } 1 \ldots j-1 \text{ take zero processing time}] \\ &= w_1 + w_2\pi_1 + w_3\pi_1\pi_2 + w_4\pi_1\pi_2\pi_3 + \ldots \end{aligned}$$

In the completion deadline model our expected value is similarly

$$\begin{aligned} \mathbf{E}[\text{total value}] &= \sum_{j=1}^{n} w_j \mathbf{Pr}[\text{jobs } 1 \ldots j \text{ take zero processing time}] \\ &= w_1\pi_1 + w_2\pi_1\pi_2 + w_3\pi_1\pi_2\pi_3 + w_4\pi_1\pi_2\pi_3\pi_4 + \ldots \end{aligned}$$

Now suppose we take an adjacent pair of jobs $(j, j+1)$ and exchange them in the ordering.

The net change in expected value is

$$[w_{j+1} + w_j\pi_{j+1} - (w_j + w_{j+1}\pi_j)] \cdot \pi_1\pi_2 \dots \pi_{j-1}$$

in the start deadline model, and

$$[w_{j+1}\pi_{j+1} + w_j\pi_j\pi_{j+1} - (w_j\pi_j + w_{j+1}\pi_j\pi_{j+1})] \cdot \pi_1\pi_2 \dots \pi_{j-1}$$

in the completion deadline model. These quantities are strictly positive precisely when

$$\frac{w_j}{1-\pi_j} < \frac{w_{j+1}}{1-\pi_{j+1}}$$

in the start deadline model, and when

$$\frac{w_j\pi_j}{1-\pi_j} < \frac{w_{j+1}\pi_{j+1}}{1-\pi_{j+1}}$$

in the completion deadline model. Therefore, if we use different orderings than (3.3) and (3.4), we can always find an adjacent pair of jobs to exchange so as to increase our expected value. □

3.2 The Greedy WSEPT Policy

For the remainder of this chapter, we discuss simple non-adaptive "greedy" policies that give $O(1)$ performance guarantees. The natural greedy non-adaptive policy is the WSEPT ordering, in which we schedule jobs in decreasing order of value (adjusted value, in the completion deadline model) divided by mean truncated processing time. Assuming our jobs are indexed according to some ordering (e.g., WSEPT), we define

$$M_j = \sum_{i=1}^{j} \mu_i.$$

Lemma 4. *Suppose we schedule our jobs non-adaptively according to the order in which they are indexed. Then the probability that jobs $1 \dots j$ are successfully scheduled is at least $1 - M_{j-1}$ in the start deadline model, and at least $1 - M_j$ in the completion deadline model.*

Proof. In the start deadline model, jobs $1 \dots j$ are successfully scheduled as long the total processing time of jobs $1 \dots j-1$ does not exceed the deadline. Therefore,

$$\begin{aligned} \mathbf{Pr}[\text{jobs } 1 \dots j \text{ scheduled successfully}] &= 1 - \mathbf{Pr}\left[\sum_{i=1}^{j-1} p_i > 1\right] \\ &= 1 - \mathbf{Pr}\left[\sum_{i=1}^{j-1} \min(p_i, 1) > 1\right] \end{aligned}$$

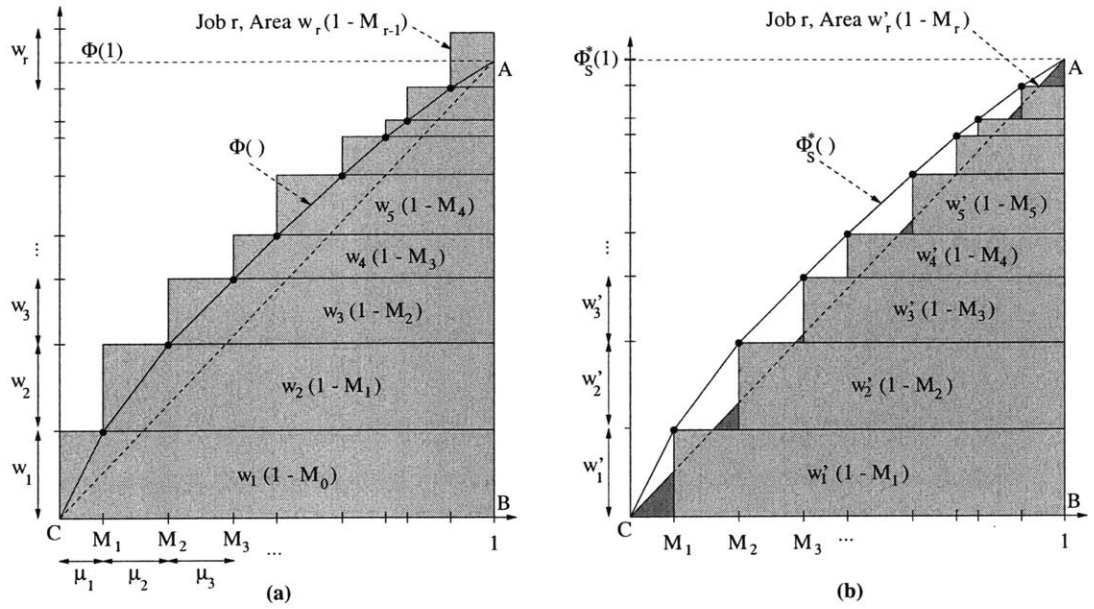

Figure 3-1: Graphs of (a) $\Phi(d)$ and (b) $\Phi^*_S(d)$ used to analyze the WSEPT policy. The expected value obtained by WSEPT (operating only on small jobs in (b)) is at least the area of the shaded rectangles.

$$\begin{aligned} &\geq 1 - \mathbf{E}\left[\sum_{i=1}^{j-1} \min(p_i, 1)\right] \quad \text{(Markov's inequality)} \\ &= 1 - M_{j-1}. \end{aligned}$$

In the completion deadline model, the proof is identical except $j-1$ is replaced with j, since job j is successfully scheduled only if jobs $1 \ldots j$ all complete prior to the deadline. □

3.2.1 Analysis in the Start Deadline Model

WSEPT is somewhat easier to analyze and provides stronger guarantees in the start deadline model than in the completion deadline model. In this section, we show that WSEPT is a 4-approximation algorithm in the start deadline model, and later in Chapter 4 we show how to tighten the analysis to prove a performance guarantee of 2.

Lemma 5. *In the start deadline model, WSEPT obtains expected value at least* $\Phi(1)/2$.

Proof. We use a simple geometric argument. Let r be the largest index such that $M_{r-1} \leq 1$, and let J denote the set of jobs successfully scheduled by WSEPT. Using Lemma 4, the

expected value we obtain is therefore

$$\begin{aligned}\mathbf{E}[w(J)] &= \sum_{j=1}^{n} w_j \mathbf{Pr}[\text{jobs } 1 \ldots j \text{ scheduled successfully}] \\ &\geq \sum_{j=1}^{r} w_j(1 - M_{j-1}).\end{aligned}$$

Consider now Figure 3-1(a), which plots the concave, piecewise linear function $\Phi(d)$. Each segment of the function corresponds to some job j in the WSEPT ordering and has slope w_j/μ_j. The total area of the shaded rectangles is $\sum_{j=1}^{r} w_j(1 - M_{j-1})$. Due to the concavity of $\Phi(d)$, this area is always at least as large as the area of the triangle ABC, which is $\Phi(1)/2$. □

Corollary 1. *The WSEPT policy delivers a performance guarantee of 4 for the problem* $1 \mid p_j \sim stoch, d_j = 1 \mid \mathbf{E}[\sum w_j \overline{V}_j]$.

Proof. Let J denote the (random) set of jobs successfully scheduled by WSEPT. Since $\Phi(d)$ is concave, we have $ADAPT \leq \Phi(2) < 2\Phi(1) \leq 4\mathbf{E}[w(J)]$. □

Since the WSEPT policy is non-adaptive, this also proves that the adaptivity gap of $1 \mid p_j \sim stoch, d_j = d \mid \mathbf{E}[\sum w_j \overline{V}_j]$ (the stochastic knapsack problem in the start deadline model) is at most 4. A similar statement regarding adaptivity gap can be made for each of our ensuing approximation algorithms that involve non-adaptive policies. Finally, we note that in the deterministic case, the deterministic WSPT policy for $1 \mid d_j = d \mid \mathbf{E}[\sum w_j \overline{V}_j]$ can be easily shown to give a 2-approximation, since the value of an optimal solution is no more than $\Phi(2)$ and WSPT gives us a solution of value at least $\Phi(1)$.

3.3 Greedy Policies for the Completion Deadline Model

For the remainder of this chapter we now focus our attention on the completion deadline model. In this section, we investigate a simple non-adaptive policy called the *best-of-2* policy that achieves a performance guarantee of 7, and in the next section we describe a randomized non-adaptive policy with a performance guarantee of $32/7 < 4.572$. The best-of-2 policy combines two different non-adaptive policies that by themselves can give poor approximation bounds in the completion deadline model: WSEPT, and what we call the *single job* policy, which schedules only the best single job (the job j maximizing w'_j) followed by an arbitrary ordering of the remaining jobs. Recall that the WSEPT ordering in the completion deadline model uses adjusted values w'_j rather than actual values w_j.

In the completion deadline model, it is well-known that WSEPT by itself might not give a good approximation bound in the presence of jobs with large expected processing times. For example, suppose we have a deterministic instance with two jobs, the first having $w_1 = (1 + \varepsilon)p_1$ and $p_1 = \varepsilon$, and the second having $w_2 = p_2 = 1$. The optimal solution in

this case is to schedule only job 2 (for a value of 1), but WSEPT only schedules job 1 and receives an expected value slightly more than ε.

It is also well-known that the single job policy by itself might not give a good approximation bound if all of our jobs have small expected processing times. Suppose we have a deterministic instance in which all jobs have $p_j = \varepsilon$, where we have a large number of jobs of two types: jobs with $w_j = \varepsilon$, and jobs with $w_j = 0$. The single job policy only guarantees that we can schedule one of the valuable jobs, so it gives us only ε value while the optimal solution can schedule nearly 1 unit of value.

Let us call a job j *small* if $\mu_j \leq \varepsilon$, and *large* if $\mu_j > \varepsilon$, for some threshold ε. As we shall see, WSEPT is a good policy for small jobs and the single job policy is good for large jobs. In our best-of-2 policy, we apply either WSEPT to the small jobs, or the single job policy to all jobs (we could apply it only to the large jobs without affecting our analysis). Let us index our jobs so that the small jobs appear first (indexed $1, 2, 3, \ldots$) and in the WSEPT ordering. Letting r be the largest index among the small jobs such that $M_r \leq 1$, we define

$$\begin{aligned} WSEPT &= \sum_{j=1}^{r} w'_j(1 - M_j) \\ SINGLE &= \max_j w'_j \\ B &= \max(WSEPT, SINGLE) \end{aligned}$$

We know that $WSEPT$ is a lower bound on the expected value of the WSEPT policy applied to small jobs (according to Lemma 4) and that $SINGLE$ is a lower bound on the expected value obtained by the single job policy. To combine the two policies, we compute $WSEPT$ and $SINGLE$ and execute the policy corresponding to whichever is larger. The resulting non-adaptive policy, which we call the *best-of-2* policy, obtains an expected value of at least B.

3.3.1 Analysis of the Best-Of-2 Policy

In this section, we show that the best-of-2 policy obtains a performance guarantee of $6 + 4\sqrt{2} < 11.657$ (for suitable choice of ε) by combining separate analyses of WSEPT and the single job policy in a straightforward manner. Although we will improve this analysis in a moment to obtain a performance guarantee of 7, there are several ideas in the simpler analysis that are definitely worthy of discussion, and in addition this gives us an opportunity to analyze WSEPT and the single job policies by themselves.

Given any set J of jobs, we define $\Phi^*_J(d)$, a version of the LP for our fractional relaxation restricted only to the jobs in J, as:

$$\begin{array}{lcl} \Phi^*_J(d) & = & \max \sum_{j \in J} x_j w'_j \\ & & \sum_{j \in J} x_j \mu_j \leq d \\ \forall j \in J & : & 0 \leq x_j \leq 1 \end{array}$$

We define $\Phi_J(d)$ similarly. If $J = A \cup B$ for disjoint sets A and B, then $\Phi_J^*(d) \leq \Phi_A^*(d) + \Phi_B^*(d)$, since the optimal solution x^* to the LP for $\Phi_J^*(d)$ can be decomposed (in a straightforward fashion) into two solution x_A and x_B that are feasible for the LPs for $\Phi_A^*(d)$ and $\Phi_B^*(d)$ whose respective objective values sum to $\Phi_J^*(d)$. Similarly, $\Phi_J(d) \leq \Phi_A(d) + \Phi_B(d)$. For the rest of this dissertation, we let S and L denote the sets of small and large jobs in our instance. We now have

$$ADAPT \leq \Phi^*(2) \leq \Phi_S^*(2) + \Phi_L^*(2).$$

The performance guarantee of $6 + 4\sqrt{2}$ for the best-of-2 policy now follows immediately from the following two lemmas.

Lemma 6. *In the completion deadline model,* $WSEPT \geq \frac{1-\varepsilon}{2}\Phi_S^*(1)$.

Lemma 7. *In the completion deadline model,* $SINGLE \geq \varepsilon\Phi_L^*(1)$.

Corollary 2. *If all jobs are small, then WSEPT is a* $\frac{4}{1-\varepsilon}$*-approximation algorithm in the completion deadline model.*

Corollary 3. *If all jobs are large, then the single job policy is a* $\frac{2}{\varepsilon}$*-approximation algorithm in the completion deadline model.*

Corollary 4. *The best-of-2 policy delivers a performance guarantee of* $6 + 4\sqrt{2}$ *for the problem* $1 \mid p_j \sim stoch, d_j = 1 \mid \mathbf{E}[\sum w_j \overline{U}_j]$, *if we take* $\varepsilon = \sqrt{2} - 1$.

Corollary 2 follows from the fact that if all jobs are small, then $ADAPT \leq \Phi^*(2) = \Phi_S^*(2) \leq 2\Phi_S^*(1) \leq \frac{4}{1-\varepsilon}WSEPT$ and Corollary 3 follows from the fact that if all jobs are large, then $ADAPT \leq \Phi^*(2) = \Phi_L^*(2) \leq 2\Phi_L^*(1) \leq \frac{2}{\varepsilon}SINGLE$. To prove Corollary 4, we note that $B \geq WSEPT$ and $B \geq SINGLE$, and obtain

$$\begin{aligned} ADAPT &\leq \Phi_S^*(2) + \Phi_L^*(2) \\ &\leq \frac{4}{1-\varepsilon}WSEPT + \frac{2}{\varepsilon}SINGLE \\ &\leq \left(\frac{4}{1-\varepsilon} + \frac{2}{\varepsilon}\right)B \\ &= (6 + 4\sqrt{2})B, \end{aligned}$$

if we choose $\varepsilon = \sqrt{2} - 1$.

In the next chapter, we will be able to improve the performance guarantee for this problem substantially by strengthening or replacing Lemmas 6 and 7. Specifically, we improve the performance guarantee for small jobs to $2/(1-\varepsilon)$ and for large jobs we show how to construct a $(1+\varepsilon)$-approximate adaptive policy. Combining the two of these together gives an adaptive policy with a performance guarantee of $3 + \varepsilon$ in the completion deadline model.

We now return to the proofs of Lemmas 6 and 7.

Proof of Lemma 6. Let r be the largest index in the WSEPT ordering such that $M_r \leq 1$. Using Lemma 4, the expected value obtained by WSEPT in the completion deadline model

can be lower bounded as follows:

$$\begin{aligned}\sum_{j=1}^{n} w_j \mathbf{Pr}[\text{jobs } 1 \ldots j \text{ scheduled successfully}] &\geq \sum_{j=1}^{r} w_j(1 - M_j) \\ &\geq \sum_{j=1}^{r} w'_j(1 - M_j) \\ &\geq \frac{\Phi^*_S(1)}{2} - \frac{\varepsilon\Phi^*_S(1)}{2}.\end{aligned}$$

The last inequality follows from a geometric argument similar to the one in the proof of Lemma 5. In Figure 3-1(b), the area of the shaded rectangles ($\sum_{j=1}^{r} w_j(1 - M_j)$) is at least as large as the area of triangle ABC ($\Phi^*_S(1)/2$) minus the area of the darkly-shaded triangles. Each dark triangle has base at most ε (since we only apply WSEPT to small jobs in the completion deadline model), and their combined heights are at most $\Phi^*_S(1)$; hence, the total area of the dark triangles is at most $\varepsilon\Phi^*_S(1)/2$. □

Proof of Lemma 7. Recall that $SINGLE = \max_j w'_j$. Letting $SINGLE(L) = \max_{j \in L} w'_j$, we therefore have $SINGLE \geq SINGLE(L)$. An equivalent way to obtain a value of $SINGLE(L)$ is by solving the following linear program.

$$\begin{array}{rcl} SINGLE(L) & = & \max \sum_{j \in L} x_j w'_j \\ & & \sum_{j \in L} x_j \leq 1 \\ \forall j \in L & : & 0 \leq x_j \leq 1 \end{array}$$

Since all jobs are large, this linear program is a relaxation of the linear program for $\Phi^*_L(\varepsilon)$,

$$\begin{array}{rcll} \Phi^*_L(\varepsilon) & = & \max \sum_{j \in L} x_j w'_j & \\ & & \sum_{j \in L} x_j \mu_j \leq \varepsilon & (*) \\ \forall j \in L & : & 0 \leq x_j \leq 1, & \end{array}$$

since replacing μ_j with ε for each job j only weakens the constraint (*). Therefore, $SINGLE \geq SINGLE(L) \geq \Phi^*_L(\varepsilon) \geq \varepsilon\Phi^*_L(1)$. □

3.3.2 Stronger Analysis of the Best-Of-2 Policy

By more carefully combining the analyses of WSEPT and the single job policy, we can show that the best-of-2 policy delivers an approximation guarantee of 7. Again, our analysis is based on two lemmas, one for small jobs and one for large jobs.

Lemma 8. *If $\varepsilon \leq 1/3$ then for any set J of small jobs, $w'(J) \leq \left(1 + \frac{2\mu(J)}{1-\varepsilon}\right) WSEPT$. Therefore, if we fix any adaptive policy P and let J_S denote the (random) set of small jobs that P attempts to schedule, we have $\mathbf{E}[w'(J_S)] \leq \left(1 + \frac{2\mathbf{E}[\mu(J_S)]}{1-\varepsilon}\right) WSEPT$.*

Lemma 9. *Fix any adaptive policy P, and let J_L denote the (random) set of large jobs that P attempts to schedule. Then $\mathbf{E}[w'(J_L)] \leq \frac{\mathbf{E}[\mu(J_L)]}{\varepsilon} SINGLE$.*

Theorem 4. *The best-of-2 policy delivers a performance guarantee of 7 for the problem $1 \mid p_j \sim stoch, d_j = 1 \mid \mathbf{E}[\sum w_j \overline{U}_j]$, if we set $\varepsilon = 1/3$.*

Proof. Fix an optimal adaptive policy P. Let J_S and J_L denote the (random) sets of small and large jobs that P attempts to schedule, and let $J = J_S \cup J_L$. We argue, as in Theorem 2, that $ADAPT \leq \mathbf{E}[w'(J)]$. Recalling that $B \geq WSEPT$ and $B \geq SINGLE$, we then have

$$\begin{aligned}
ADAPT &\leq \mathbf{E}[w'(J_S)] + \mathbf{E}[w'(J_L)] \\
&\leq \left(1 + \frac{2\mathbf{E}[\mu(J_S)]}{1-\varepsilon}\right) WSEPT + \frac{\mathbf{E}[\mu(J_L)]}{\varepsilon} SINGLE \\
&\leq \left(1 + \frac{2\mathbf{E}[\mu(J_S)]}{1-\varepsilon} + \frac{\mathbf{E}[\mu(J_L)]}{\varepsilon}\right) B \\
&\leq \left(1 + \max\left(\frac{2}{1-\varepsilon}, \frac{1}{\varepsilon}\right) \mathbf{E}[\mu(J_S \cup J_L)]\right) B \\
&\leq \left(1 + 2\max\left(\frac{2}{1-\varepsilon}, \frac{1}{\varepsilon}\right)\right) B \quad \text{(by Lemma 1)} \\
&= 7B
\end{aligned}$$

for $\varepsilon = 1/3$. □

This analysis is actually fairly tight considering that we are comparing B (expected value we can get with best-of-2) to $\Phi^*(2)$ (upper bound on $ADAPT$). Consider a deterministic instance with 6 jobs having $w_j = p_j = 1/3$. Regardless of whether or not these jobs are classified as being small or large, we have $B = 1/3$ while $\Phi^*(2) = 2$, for a gap of 6.

We now prove Lemmas 8 and 9.

Proof of Lemma 8. Let S denote the set of all small jobs. Take any set $J \subseteq S$ and consider three cases:

1. If $\mu(J) \leq \frac{1+\varepsilon}{2}$, then we turn again to a geometric argument. Let $d = \mu(J)$, and let k be the index such that $M_k \leq d < M_{k+1}$, as shown in Figure 3-2. Just as in our previous geometric arguments, $WSEPT$ is the total area of the shaded rectangles. We claim that this area is at least as large as the area of the bold rectangle, $(1-d)\Phi_S^*(d)$. To do this, we show that the area of rectangle B is always at least as large as that of A (the only "missing piece" in the bold rectangle). Equivalently, we show that B extended

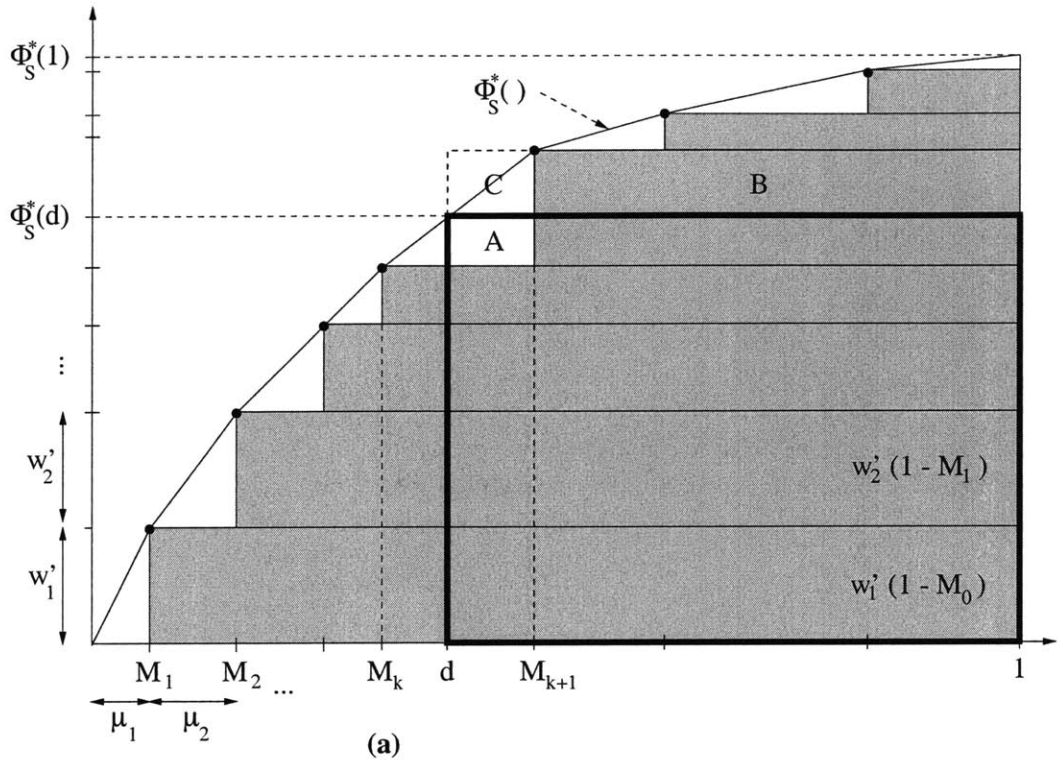

Figure 3-2: Bounding $WSEPT$ below by $(1-d)\Phi_S(d)$.

to the left into C has an area at least as large as A extended upward into C. Letting $\lambda = w_{k+1}/\mu_{k+1}$ be the slope of $\Phi^*_S(\cdot)$ at d, we have

$$\begin{aligned} Area(A+C) &= (M_{k+1}-d)\lambda\mu_{k+1} \\ Area(B+C) &= (1-d)\lambda(M_{k+1}-d), \end{aligned}$$

so the area of B dominates that of A as long as $1-d \geq \mu_{k+1}$, which is true if $1-d \geq \varepsilon$ since all jobs are small. The condition $1-d \geq \varepsilon$ is implied by $\varepsilon \leq 1/3$ and $d = \mu(J) \leq (1+\varepsilon)/2$, since in this case $d+\varepsilon \leq (1+\varepsilon)/2+\varepsilon = 1/2+3\varepsilon/2 \leq 1$. Since $WSEPT \geq (1-d)\Phi^*_S(d)$, we have $w'(J) \leq \Phi^*_S(d) \leq \frac{1}{1-d}WSEPT$. The function $\frac{1}{1-x}$ is convex on the interval $x \in [0,(1+\varepsilon)/2]$, so we can bound it above by its linear interpolation on this interval, which is $1+\frac{2x}{1-\varepsilon}$. Therefore, $w'(J) \leq \left(1+\frac{2\mu(J)}{1-\varepsilon}\right)WSEPT$.

2. If $\mu(J) \in [\frac{1+\varepsilon}{2}, 1]$, then

$$w'(J) \leq \Phi^*_S(\mu(J)) \leq \Phi^*_S(1) \leq \frac{2}{1-\varepsilon}WSEPT \leq \left(1+\frac{2\mu(J)}{1-\varepsilon}\right)WSEPT.$$

3. If $\mu(J) \geq 1$, then by Lemma 6 and concavity of $\Phi_S^*(\cdot)$ we have

$$w'(J) \leq \Phi_S^*(\mu(J)) \leq \mu(J)\Phi_S^*(1) \leq \frac{2\mu(J)}{1-\varepsilon} WSEPT. \qquad \square$$

Proof of Lemma 9. Fix a policy P, let J_L denote the (random) set of large jobs that P attempts to schedule, and let x_j denote the probability that P attempts to schedule job j. We can now write

$$\begin{aligned} \mathbf{E}[w'(J_L)] &= \sum_{j \in J_L} w'_j x_j \\ &\leq \left(\sum_{j \in J_L} x_j \right) \left(\max_{j \in [n]} w'_j \right) \\ &= \mathbf{E}[|J_L|]\ SINGLE \\ &\leq \frac{\mathbf{E}[\mu(J_L)]}{\varepsilon}\ SINGLE \end{aligned}$$

where the final inequality uses the fact that all jobs in J_L are large. Note that we could have also proved this lemma using an argument based on relating two linear programs, just as in the proof of Lemma 7. $\square$

3.4 A Randomized Policy for the Completion Deadline Model

In terms of approximation guarantee, there are several non-adaptive policies with stronger performance than the best-of-2 policy in the preceding section. In this section, we discuss a randomized policy we call the *round-by-value* policy, which achieves a performance guarantee of $32/7 < 4.572$, and in the next chapter we give a different type of "best of 2" policy that delivers an approximation guarantee of 4 (the analysis in this case does not use an LP relaxation to bound $ADAPT$). Both of these policies are primarily developed in the Ph.D. dissertation of Jan Vondrák [58], so we will omit some of the proofs in our following discussion.

Unlike our previous approaches in the completion deadline model, the round-by-value algorithm does not need to distinguish small and large jobs. It proceeds as follows:

1. Solve the linear program for $\Phi^*(1)$. Let $x_1 \ldots x_n$ denote the optimal solution.
2. Select a job j with probability proportional to $x_j w'_j$ and schedule it first.
3. Schedule the remaining jobs according to the WSEPT ordering for the completion deadline model: $\frac{w'_1}{\mu_1} \geq \frac{w'_2}{\mu_2} \geq \ldots \geq \frac{w'_n}{\mu_n}$.

The purpose of the randomization above is to try to alleviate the problem with our previous analyses in the completion deadline model where a job of high value is "cut off" (scheduled

in our fractional relaxation so that it just exceeds the deadline). Recall that our previous "geometric" analyses do not give us credit for any such jobs.

Theorem 5. *In the completion deadline model, the non-adaptive round-by-value policy has a performance guarantee of* $32/7 < 4.572$. *More precisely, given any set of jobs* J, *the round-by-value policy on* J *delivers an expected value of at least* $\frac{7}{32}\Phi^*_J(2)$.

Proof of Theorem 5 appears in [58], where it is also shown that this policy gives us the best possible performance guarantee as long as we bound $ADAPT$ by $\Phi^*(2)$ and use Markov's inequality to lower bound the probability that jobs are successfully scheduled (Lemma 4). Note that we can view the round-by-value policy as a convex combination of non-adaptive policies, one of which must have a performance guarantee of at least 32/7. Hence, the adaptivity gap for $1 \mid p_j \sim stoch, d_j = d \mid \mathbf{E}[\sum w_j \overline{U}_j]$ is at most 32/7. We can also derandomize the round-by-value algorithm by repeating the analysis from the proof (see [58]) for each potential starting job, and taking the one that gives the largest expected value. Finally, note that even though the expected value obtained by the round-by-value algorithm is comparable to $\Phi^*(2)$, the algorithm itself only potentially uses jobs that we expect to start prior to time $t = 1$ in WSEPT ordering. We will exploit this feature in Chapter 6 when we study parallel environments.

4. One Machine with Common Deadlines: Improved Results

In this section we describe non-adaptive and adaptive policies for the stochastic knapsack problem that have stronger performance guarantees than those in the preceding chapter. Although one can interpret some of these results in the context of linear programming just as in the preceding chapter, the analyses in this chapter will be more of a combinatorial nature.

We begin by reproducing (with small modifications) a proof from the Ph.D. thesis of Jan Vondrák [58] showing that the WSEPT policy gives a performance guarantee of 2 in the start deadline model, and that a "best of two" policy (taking the better of WSEPT and a carefully chosen single job) delivers a performance guarantee of 4 in the completion deadline model.

We then develop an $(1+\varepsilon)$-approximate *adaptive* policy for large jobs, and show how to combine this with the preceding analysis (for small jobs) to obtain a $(3+\varepsilon)$-approximate adaptive policy in the completion deadline model. This result is primarily of theoretical interest, however, since the $(1+\varepsilon)$-approximate policy for large jobs can have a very large running time (albeit still polynomial, as long as ε is constant).

Finally, we show how to obtain a $(2+\varepsilon)$-approximate adaptive policy in the event that our processing times distributions are somewhat "well behaved" — in particular, if for every job j the ratio P_j/μ_j is at most a constant, where P_j denotes the maximum value in the support of p_j.

4.1 A New Analysis for WSEPT

In this section we present a much stronger analysis (from [58]) for the expected value obtained by the WSEPT policy. The analysis applies to both the start deadline and completion deadline models, and consists of two main pieces: just as in the preceding chapter, we first discuss lower bounds on $\mathbf{E}[\mu(J)]$ (where J denotes the random set of jobs we manage to schedule using WSEPT) and upper bounds on $\mathbf{E}[\mu(J)]$ (where J denotes the random set of jobs that an optimal adaptive policy attempts to schedule). After this, we modify these

bounds so they relate the expected *value* we obtain from WSEPT to the expected value obtained by an optimal adaptive policy.

4.1.1 Bounding $\mathbf{E}[\mu(J)]$

Suppose we schedule our jobs non-adaptively according to their indices in the WSEPT ordering, and let $M_j = \sum \mu_i$ over all $i \leq j$. In the preceding chapter, we used Lemma 4 (based on Markov's inequality) to lower bound the probability that jobs $1 \ldots j$ are successfully scheduled by $1 - M_{j-1}$ (start deadline model) or $1 - M_j$ (completion deadline model). These bounds are only useful, however, for $M_{j-1} \leq 1$ (or $M_j \leq 1$), which may only hold for a small prefix of the jobs. The following lemma strengthens this statement, when computing the probability that job j is successfully scheduled, by conditioning on jobs $1 \ldots j-1$ having already been successfully scheduled. For example, in the completion deadline model, we can lower bound the probability of job j completing by the deadline by $1 - t_{j-1} - \mu_j$, where t_{j-1} is the expected total processing time of jobs $1 \ldots j-1$ conditioned on the fact that job $j-1$ completes by the deadline. Since t_{j-1} is no larger (and potentially much smaller) than M_{j-1}, the new bound is much stronger.

Lemma 10. *Fix any non-adaptive policy P, and index our jobs according to the scheduling order for P. Let π_j denote the probability that job j completes prior to the deadline, and let J denote the set of jobs successfully scheduled by P. Then in the start deadline model,*

$$\mathbf{E}[\mu(J)] = \sum_{j=1}^{n} \pi_{i-1}\mu_j \geq 1 - \prod_{j=1}^{n}(1-\mu_j). \tag{4.1}$$

In the completion deadline model, if all jobs are small we similarly have

$$\mathbf{E}[\mu(J)] = \sum_{j=1}^{n} \pi_j\mu_j \geq (1-\varepsilon)\left(1 - \prod_{j=1}^{n}(1-\mu_j)\right). \tag{4.2}$$

Proof. Define the random variable $T_j = \sum_{i=1}^{j} \min(p_j, 1)$, and let $t_j = \mathbf{E}[T_j | T_j \leq 1]$. Now,

$$\begin{aligned}
t_{j-1} + \mu_j &= \mathbf{E}[T_{j-1} + \min(p_j, 1) \mid T_{j-1} \leq 1] \\
&= \mathbf{E}[T_j \mid T_{j-1} \leq 1] \\
&= \mathbf{E}[T_j \mid T_j \leq 1]\ \mathbf{Pr}[T_j \leq 1 \mid T_{j-1} \leq 1] + \mathbf{E}[T_j \mid T_j > 1]\ \mathbf{Pr}[T_j > 1 \mid T_{j-1} \leq 1] \\
&\geq t_j \frac{\pi_j}{\pi_{j-1}} + 1\left(1 - \frac{\pi_j}{\pi_{j-1}}\right) \\
&= 1 - (1-t_j)\frac{\pi_j}{\pi_{j-1}}.
\end{aligned}$$

As a result,

$$\frac{\pi_j}{\pi_{j-1}} \geq \frac{1 - t_{j-1} - \mu_j}{1 - t_j}. \tag{4.3}$$

This holds even for $j = 1$ if we adopt the convention that $\pi_0 = 1$ and $t_0 = 0$. Multiplying together (4.3) from 1 to j we obtain

$$\begin{aligned}
\pi_j &\geq \frac{1-\mu_1}{1-t_1}\,\frac{1-t_1-\mu_2}{1-t_2}\,\frac{1-t_2-\mu_3}{1-t_3}\cdots\frac{1-t_{j-1}-\mu_j}{1-t_j} \\
&= \frac{1-\mu_1}{1-t_j}\left(1-\frac{\mu_2}{1-t_1}\right)\left(1-\frac{\mu_3}{1-t_2}\right)\cdots\left(1-\frac{\mu_j}{1-t_{j-1}}\right) \\
&= \frac{1-\mu_1}{1-t_j}\prod_{i=2}^{j}(1-z_i) \\
&= \frac{1}{1-t_j}\prod_{i=1}^{j}(1-z_i)\,, \qquad (4.4)
\end{aligned}$$

where $z_j = \mu_j/(1-t_{j-1})$. In the start deadline model,

$$\begin{aligned}
\sum_{j=1}^{n}\mu_j\pi_{j-1} &\geq \sum_{j=1}^{n}\frac{\mu_j}{1-t_{j-1}}\prod_{i=1}^{j-1}(1-z_i) \\
&= \sum_{j=1}^{n} z_j\prod_{i=1}^{j-1}(1-z_i) \\
&= 1-\prod_{j=1}^{n}(1-z_j)\,. \qquad (4.5)
\end{aligned}$$

Consider now two cases. If $\mu_j \leq 1 - t_{j-1}$ for all $j \in [n]$, then for all $j \in [n]$ we have $\mu_j \leq z_j \leq 1$ so $1 - z_j \geq 0$. Hence (4.5) can be written as

$$1-\prod_{j=1}^{n}(1-z_j) \geq 1-\prod_{j=1}^{n}(1-\mu_j)\,. \qquad (4.6)$$

Otherwise, if $\mu_k > 1 - t_{k-1}$ for some $k \in [n]$, then consider the smallest such k. We then have $\mu_j \leq z_j \leq 1$ for all $j < k$ and by (4.4),

$$\pi_{k-1}\mu_k > \pi_{k-1}(1-t_{k-1}) \geq \prod_{j=1}^{k-1}(1-z_j). \qquad (4.7)$$

Therefore,

$$\begin{aligned}
\sum_{j=1}^{n}\pi_{j-1}\mu_j &\geq \sum_{j=1}^{k}\pi_{j-1}\mu_j \\
&= \pi_{k-1}\mu_k + \sum_{j=1}^{k-1}\pi_{j-1}\mu_j
\end{aligned}$$

$$\geq \left(\prod_{j=1}^{k-1}(1-z_j)\right) + \left(1 - \prod_{j=1}^{k-1}(1-z_j)\right) \quad \text{(Using (4.7) and (4.5))}$$
$$= 1.$$

This concludes the proof for the start deadline model. In the completion deadline model with small jobs, we have

$$\begin{aligned}
\sum_{j=1}^{n}\pi_j\mu_j &= \sum_{j=1}^{n}\pi_{j-1}\mu_j - \sum_{j=1}^{n}(\pi_{j-1}-\pi_j)\mu_j \\
&\geq 1 - \prod_{j=1}^{n}(1-z_j) - \sum_{j=1}^{n}(\pi_{j-1}-\pi_j)\varepsilon \quad \text{(Using (4.5))} \\
&= 1 - \prod_{j=1}^{n}(1-z_j) - (\pi_0 - \pi_n)\varepsilon \\
&= (1-\varepsilon) - \prod_{j=1}^{n}(1-z_j) + \varepsilon\pi_n. \qquad (4.8)
\end{aligned}$$

Consider again two cases. If $\mu_j \leq 1 - t_{j-1}$ for all $j \in [n]$, then for all $j \in [n]$ we have $\mu_j \leq z_j \leq 1$ so $1 - z_j \geq 0$. Using (4.4), we can therefore bound (4.8) as follows:

$$\begin{aligned}
(1-\varepsilon) - \prod_{j=1}^{n}(1-z_j) + \varepsilon\pi_n &\geq (1-\varepsilon) - \prod_{j=1}^{n}(1-z_j) + \frac{\varepsilon}{1-t_n}\prod_{j=1}^{n}(1-z_j) \\
&\geq (1-\varepsilon) - (1-\varepsilon)\prod_{j=1}^{n}(1-z_j) \\
&\geq (1-\varepsilon)\left(1 - \prod_{j=1}^{n}(1-\mu_j)\right). \qquad (4.9)
\end{aligned}$$

Otherwise, if $\mu_{k+1} > 1 - t_k$ for some k, then consider the smallest such k. We then have $\mu_j \leq z_j \leq 1$ for all $j \leq k$ and by (4.4),

$$\prod_{j=1}^{k}(1-z_j) \leq (1-t_k)\pi_k \leq \mu_{k+1}\pi_k \leq \varepsilon\pi_k. \qquad (4.10)$$

Therefore,

$$\begin{aligned}
\sum_{j=1}^{n}\pi_j\mu_j &\geq \sum_{j=1}^{k}\pi_j\mu_j \\
&\geq (1-\varepsilon) - \prod_{j=1}^{k}(1-z_j) + \varepsilon\pi_k \quad \text{(Using (4.8))}
\end{aligned}$$

$$\begin{aligned} &\geq (1-\varepsilon) - \varepsilon\pi_k + \varepsilon\pi_k \quad \text{(Using (4.10))} \\ &= 1-\varepsilon. \end{aligned}$$

This concludes the proof. □

Just as Lemma 10 improves the "$\mathbf{E}[\mu(J)]$" lower bound for WSEPT, we now demonstrate a corresponding improvement for the upper bound on ADAPT, which can be viewed as a strengthening of Lemma 1.

Lemma 11. *Fix any adaptive policy P. Letting J denote the random set of jobs that P attempts to schedule, we have*

$$\mathbf{E}[\mu(J)] \leq 2\left(1 - \prod_{j=1}^{n}(1-\mu_j)\right). \tag{4.11}$$

Proof. We follow the same inductive structure as in the proof of Lemma 1. Suppose P finds itself in a situation with a deadline d units of time into the future and a set R of remaining jobs. Let $J(d,R)$ denote the (random) set of jobs that P manages to schedule from this point on. Using induction on $|R|$, we argue that for any set of jobs R and any random variable $t \leq 1$ independent of the processing times of jobs in R,

$$\mathbf{E}[\mu(J(t,R))] \leq (\mathbf{E}[t]+1)\left(1 - \prod_{j\in R}(1-\mu_j)\right).$$

The lemma follows if we set $t=1$ and $R=[n]$. We take $|R|=0$ as a trivial base case. Suppose, now, that P is faced with a deadline t units of time into the future (a random variable) and has a set R of available remaining jobs. If P happens to schedule job $j \in R$ next, then

$$\begin{aligned} \mathbf{E}[\mu(J(t,R))] &= \mu_j + \mathbf{E}[\mu(J(t-p_j, R\backslash\{j\})) \mid p_j \leq 1]\mathbf{Pr}[p_j \leq 1] \\ &= \mu_j + \mathbf{E}[\mu(J(t-\min(p_j,1), R\backslash\{j\})) \mid p_j \leq 1]\mathbf{Pr}[p_j \leq 1] \\ &= \mu_j + \mathbf{E}[\mu(J(t-\min(p_j,1), R\backslash\{j\}))] \\ &\leq \mu_j + (\mathbf{E}[t-\min(p_j,1)]+1)\left(1 - \prod_{i\in R\backslash\{j\}}(1-\mu_i)\right) \quad \text{(by induction)} \\ &= \mu_j + (\mathbf{E}[t]-\mu_j+1)\left(1 - \prod_{i\in R\backslash\{j\}}(1-\mu_i)\right) \\ &= (\mathbf{E}[t]+1) - (\mathbf{E}[t]-\mu_j+1)\prod_{i\in R\backslash\{j\}}(1-\mu_i) \\ &\leq (\mathbf{E}[t]+1) - (\mathbf{E}[t]+1)(1-\mu_j)\prod_{i\in R\backslash\{j\}}(1-\mu_i) \end{aligned}$$

$$= (\mathbf{E}[t]+1)\left(1-\prod_{i\in R}(1-\mu_i)\right),$$

and completes the proof since it holds for every job $j \in R$. □

Observe that Lemmas 10 and 11 are actually tight, if we consider an instance in which the processing time of each job is either 0 (probability $1-\varepsilon$) or 1 (probability ε).

4.1.2 Bounds on Expected Value

We now show how to obtain better lower bounds on the expected value obtained by WSEPT and better upper bounds on *ADAPT*. Recall that ordering of jobs used by the WSEPT policy depends on whether we are in the start deadline or completion deadline model. If we define W_j as w_j if we are in the start deadline model, or as w'_j in the completion deadline model, then the WSEPT ordering is $\frac{W_1}{\mu_1} \geq \frac{W_2}{\mu_2} \geq \ldots \geq \frac{W_n}{\mu_n}$. By defining W_j in this fashion, we enable the proofs that follow to work in both the start deadline model and the completion deadline model. We now define

$$V = \sum_{j=1}^{n} W_j \prod_{i=1}^{j-1}(1-\mu_i). \tag{4.12}$$

Lemma 12. *In the start deadline and completion deadline models, $ADAPT \leq 2V$.*

Proof. Fix an optimal adaptive policy P, and let x_j denote the probability that P attempts to schedule job j. In both the start deadline and completion deadline models, we have $ADAPT \leq \sum_{j=1}^{n} W_j x_j$. This clearly holds with equality in the start deadline model, (where $W_j = w_j$), and follows as a consequence of Theorem 2 in the completion deadline model (where $W_j = w'_j$). Adopting the convention that $W_{n+1}/\mu_{n+1} = 0$, we have

$$\begin{aligned}
ADAPT &\leq \sum_{j=1}^{n} W_j x_j \\
&= \sum_{j=1}^{n} \frac{W_j}{\mu_j} x_j \mu_j \\
&= \sum_{j=1}^{n} \left(\frac{W_j}{\mu_j} - \frac{W_{j+1}}{\mu_{j+1}}\right) \sum_{i=1}^{j} x_i \mu_i \\
&\leq 2\sum_{j=1}^{n} \left(\frac{W_j}{\mu_j} - \frac{W_{j+1}}{\mu_{j+1}}\right) \left(1-\prod_{i=1}^{j}(1-\mu_i)\right) \quad \text{(Lemma 11)} \\
&\leq 2\sum_{j=1}^{n} \frac{W_j}{\mu_j} \left(\prod_{i=1}^{j-1}(1-\mu_i) - \prod_{i=1}^{j}(1-\mu_i)\right)
\end{aligned}$$

$$
\begin{aligned}
&= 2\sum_{j=1}^{n} \frac{W_j}{\mu_j} \left(\prod_{i=1}^{j-1} (1-\mu_i) \right) (1-(1-\mu_j)) \\
&= 2\sum_{j=1}^{n} W_j \prod_{i=1}^{j-1} (1-\mu_i) \\
&= 2V
\end{aligned}
$$

and this completes the proof. □

Lemma 13. *In the start deadline model, the expected value obtained by WSEPT is at least V. In the completion deadline model, if all jobs are small then WSEPT obtains an expected value of at least $(1-\varepsilon)V$.*

Corollary 5. *The WSEPT policy delivers a performance guarantee of 2 for the problem $1 \mid p_j \sim stoch, d_j = 1 \mid \mathbf{E}[\sum w_j \overline{V}_j]$, and if all jobs are small, then WSEPT delivers a performance guarantee of $2/(1-\varepsilon)$ for $1 \mid p_j \sim stoch, d_j = 1 \mid \mathbf{E}[\sum w_j \overline{U}_j]$.*

Proof of Lemma 13. Index the jobs according to the WSEPT ordering, and let π_j denote the probability that job j completes by the deadline. In the start deadline model, the expected value we obtain is

$$
\begin{aligned}
\sum_{j=1}^{n} w_j \pi_{j-1} &\geq \sum_{j=1}^{n} W_j \pi_{j-1} \\
&= \sum_{j=1}^{n} \frac{W_j}{\mu_j} \pi_{j-1} \mu_j \\
&= \sum_{j=1}^{n} \left(\frac{W_j}{\mu_j} - \frac{W_{j+1}}{\mu_{j+1}} \right) \sum_{i=1}^{j} \pi_{i-1} \mu_i \\
&\geq \sum_{j=1}^{n} \left(\frac{W_j}{\mu_j} - \frac{W_{j+1}}{\mu_{j+1}} \right) \left(1 - \prod_{i=1}^{j} (1-\mu_i) \right) \quad \text{(Lemma 10)} \\
&= \sum_{j=1}^{n} \frac{W_j}{\mu_j} \left(\prod_{i=1}^{j-1} (1-\mu_i) - \prod_{i=1}^{j} (1-\mu_i) \right) \\
&= \sum_{j=1}^{n} \frac{W_j}{\mu_j} \left(\prod_{i=1}^{j-1} (1-\mu_i) \right) (1-(1-\mu_j)) \\
&= \sum_{j=1}^{n} W_j \prod_{i=1}^{j-1} (1-\mu_i) \\
&= V.
\end{aligned}
$$

In the completion deadline model with small jobs, the application of Lemma 10 above brings with it an extra factor of $(1-\varepsilon)$, which carries through to the end of argument. □

4.1.3 A 4-Approximate Policy in the Completion Deadline Model

The analysis above shows that WSEPT by itself is a 2-approximate non-adaptive policy in the start deadline model. In the completion deadline model, we obtain a 4-approximate non-adaptive policy by taking the better of two solutions:

- If any job j satisfies $w'_j \geq V/2$, then schedule only this single job.
- Otherwise, schedule all jobs (not just small jobs) according to the WSEPT ordering.

We call this the *improved best-of-2 policy.*

Theorem 6. *The improved best-of-2 policy delivers a performance guarantee of 4 for the problem* $1 \mid p_j \sim stoch, d_j = 1 \mid \mathbf{E}[\sum w_j \overline{U}_j]$.

Proof. If a single job is scheduled, then we obtain an expected value of at least $V/2$ and $ADAPT \leq 2V$, so the performance guarantee of 4 holds in this case. Consider now the case where we use the WSEPT ordering, and let π_j denote the probability that job j completes by the deadline. Taking the jobs to be indexed according to the WSEPT ordering, the expected value we obtain is

$$\begin{aligned}
\sum_{j=1}^{n} w_j \pi_j &\geq \sum_{j=1}^{n} w'_j \pi_j \\
&= \sum_{j=1}^{n} w'_j \pi_{j-1} - \sum_{j=1}^{n} w'_j (\pi_{j-1} - \pi_j) \\
&\geq V - \sum_{j=1}^{n} w'_j (\pi_{j-1} - \pi_j) \quad \text{(Lemma 13)} \\
&\geq V - \left(\max_{j \in [n]} w'_j \right) \sum_{j=1}^{n} (\pi_{j-1} - \pi_j) \\
&\geq V - (V/2)(\pi_0 - \pi_n) \\
&\geq V/2,
\end{aligned}$$

which gives a performance guarantee of 4. □

4.2 Exhaustive Enumeration for Large Jobs

We focus on the completion deadline model for the remainder of the chapter. As usual, let S and L denote the sets of small and large jobs in our instance, and let $ADAPT(S)$ and $ADAPT(L)$ denote the expected value obtained by an optimal adaptive policy restricted to these sets. Taking only the small jobs and indexing them according to the WSEPT ordering

(based on w'_j rather than w_j since we are in the completion deadline model), let us define

$$SMALL = (1-\varepsilon)\sum_{j=1}^{n} w'_j \prod_{i=1}^{j-1}(1-\mu_i).$$

We know from the preceding section that the WSEPT policy obtains at least an expected value of $SMALL$, and that $ADAPT(S) \leq \frac{2}{1-\varepsilon} SMALL$. In this section, we construct an adaptive policy for large jobs that delivers an expected value of at least $LARGE$, where $ADAPT(L) \leq (1+\varepsilon)LARGE$. Combining these two policies (taking the better of the two), we obtain a $(3+\varepsilon)$-approximate policy.

Theorem 7. *Let $\varepsilon' = \varepsilon/5$ be used to define the boundary between small and large jobs. Applying either WSEPT for small jobs (if $SMALL \geq LARGE$) or the adaptive policy described in this section for large jobs (if $LARGE > SMALL$), we obtain a $(3+\varepsilon)$-approximate adaptive policy for $1 \mid p_j \sim stoch, d_j = 1 \mid \mathbf{E}[\sum w_j \overline{U}_j]$.*

Proof. Let $B = \max(SMALL, LARGE)$ denote the expected value obtained by the policy described in the theorem. Using the fact that $\frac{1}{1-\varepsilon'} \leq 1 + 2\varepsilon'$ for $\varepsilon' \leq 1/2$, we then have

$$\begin{aligned} ADAPT &\leq ADAPT(S) + ADAPT(L) \\ &\leq \frac{2}{1-\varepsilon'} SMALL + (1+\varepsilon')LARGE \\ &\leq (3+5\varepsilon')\, B \\ &\leq (3+\varepsilon)\, B, \end{aligned}$$

which is the desired performance guarantee. □

We proceed now to describe our $(1+\varepsilon)$-approximate policy for large jobs. An interesting feature of this policy is that it is adaptive — given a set of remaining large jobs J and a deadline d, it spends polynomial time (assuming ε is constant) and computes the next job to schedule in a $(1+\varepsilon)$-approximate policy. It also estimates the value it will eventually obtain, thereby computing a $(1+\varepsilon)$-approximation to $ADAPT(L)$ (i.e., the value of $LARGE$).

The policy we describe shortly is the first result we encounter in this dissertation with a weakly-polynomial running time. Let K be an upper bound on the number of bits required to represent any job value w_j, instantiated processing time for p_j, or probability value obtained by evaluating the cumulative distribution for p_j. Assuming ε is a constant, our running times will be polynomial in n and K.

Let us think of an adaptive policy as a decision tree, where at a depth-l node we have a choice from among $|L| - l$ jobs to schedule next (and this decision will be made based on remaining deadline). We first claim that by restricting our policy to have at most a constant depth (depending on ε), we only forfeit a small (ε-fraction) amount of expected value. Hence, our decision tree will contain at most a polynomial number of nodes, and we can exhaustively compute an approximately-optimal adaptive policy for each node in a bottom-up fashion — at each depth-l node in the decision tree, our algorithm for determining the appropriate job

to schedule next will make a polynomial number of calls to algorithms at depth level $l+1$. Since the resulting polynomial running time can have extremely high degree, we stress that this is a result of mainly theoretical interest (as is true for PTAS's for many other NP-hard problems that use exhaustive enumeration to solve instances comprised of "large" items).

Let us define the function $F_{J,k}(t)$ to give the maximum expected value one can achieve if t units of time have already been used, we have already scheduled jobs $J \subseteq L$, and we may only schedule k more jobs. For example, $ADAPT(L) = F_{\emptyset,|L|}(0)$. The analysis of our adaptive policy relies primarily on the following technical lemma.

Lemma 14. *For any constant $\delta \in [0,1]$, any $t \in [0,1]$, any set of large jobs $J \subseteq L$ and any $k = O(1)$, there exists a polynomial-time algorithm (which we call $A_{J,k,\delta}$) that computes a job in $L \backslash J$ to schedule first, which constitutes the beginning of an adaptive policy obtaining expected value in the range $[F_{J,k}(t)/(1+\delta), F_{J,k}(t)]$. The algorithm also computes the expected value of this policy, which we denote by the function $G_{J,k,\delta}(t) \in [F_{J,k}(t)/(1+\delta), F_{J,k}(t)]$.*

We prove the lemma shortly, but consider for a moment its implications. Using the lemma, we use $A_{\emptyset,k,\delta}$ to compute an adaptive policy that obtains an expected value of at least $LARGE = G_{\emptyset,k,\delta}(0)$ where $\delta = \varepsilon/3$ and $k = \frac{12}{\varepsilon^2}$. Then

$$F_{\emptyset,k}(0) \leq (1+\varepsilon/3)LARGE.$$

Letting J_L denote the (random) set of jobs successfully scheduled by an optimal adaptive policy, Lemma 1 tells us that $\mathbf{E}[\mu(J_L)] \leq 2$, so by Markov's inequality (and the fact that $\mu_j \geq \varepsilon$ for all $j \in L$) implies that $\mathbf{Pr}[|J_L| \geq k] \leq \varepsilon/6$. For any k, we can now decompose the expected value obtained by $ADAPT(L)$ into the value from the first k jobs, and the value from any jobs after the first k. The first quantity is bounded by $F_{\emptyset,k}(0)$ and the second quantity is bounded $ADAPT(L)$ even when we condition on the event $|J_L| > k$. Therefore,

$$ADAPT(L) \leq F_{\emptyset,k}(0) + ADAPT(L)\mathbf{Pr}[|J_L| > k] \leq F_{\emptyset,k}(0) + \frac{\varepsilon}{6}ADAPT(L),$$

and assuming that $\varepsilon \leq 3$ we have

$$ADAPT(L) \leq \frac{F_{\emptyset,k}(0)}{1-\varepsilon/6} \leq (1+\varepsilon/3)F_{\emptyset,k}(0) \leq (1+\varepsilon/3)^2 LARGE \leq (1+\varepsilon)LARGE.$$

Proof of Lemma 14. We use induction on k, taking $k = 0$ as a trivial base case. Assume the lemma now holds up to some value of k, so for every set $J \subseteq L$ of large jobs we have a polynomial-time algorithm $A_{J,k,\delta/3}$. We use $\delta/3$ as the constant for our inductive step since $(1+\delta/3)^2 \leq 1+\delta$ for $\delta \in [0,1]$ (that is, our argument below will fall short by 2 consecutive factors of $1+\delta/3$). Note that this decreases our constant δ by a factor of 3 for every level of induction, but since we only carry the induction out to a constant number of levels, we can still treat δ as a constant at every step of the induction.

We now describe to construct the algorithm $A_{\bullet,k+1,\delta}$ using a polynomial number of "recursive" calls to the polynomial-time algorithm $A_{\bullet,k,\delta}$. Again, since we only use a constant number of levels of induction (recall that the final value of k is constant), at each of which

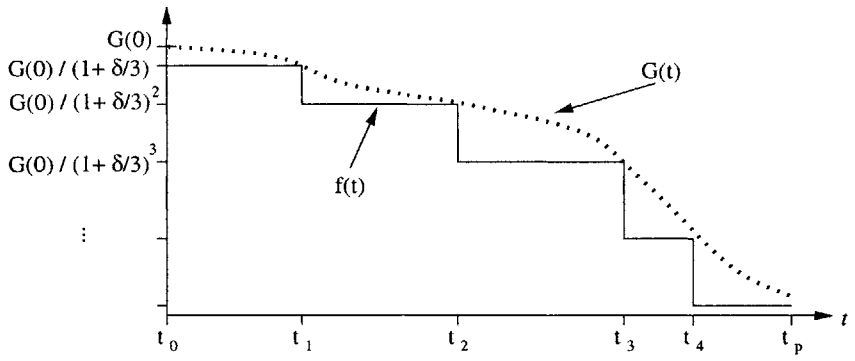

Figure 4-1: Approximation of the function $G(t) = G_{J\cup\{j\},k,\delta/3}(t)$ (shown by a dotted line) by a piecewise constant function $f(t)$ so that $f(t) \in [G(t)/(1+\delta/3), G(t)]$.

we make only a polynomial number of function calls at the preceding level, the overall running time of $A_{\bullet,k+1,\delta}$ will be polynomial (albeit with potentially very high degree).

The algorithm $A_{J,k+1,\delta}$ must decide which job in $L\backslash J$ to schedule first, given that we are currently at time t, in order to launch a $(1+\delta)$-approximate policy for scheduling at most $k+1$ more jobs. To do this, we approximate the expected value we get with each job $j \in L\backslash J$ and take the best job. To estimate the expected value if we start with job j, we might try to use random sampling: sample a large number of instances of p_j and for each one we call $A_{J\cup\{j\},k,\bullet}$ to approximate the expected value $F_{J\cup\{j\},k}(t+p_j)$ obtained by the remainder of an optimal policy starting with j. However, this approach does not work due to the "rare event" problem often encountered with random sampling. If p_j has exponentially small probability of taking very small values for which $F_{J\cup\{j\},k}(t+p_j)$ is very large value, we will likely miss this contribution to the aggregate expected value.

To remedy the problem above, we a sort of "assisted sampling" that first determines the "interesting" ranges of values of p_j we should consider. We approximate $F_{J\cup\{j\},k}(\cdot)$ by a piecewise constant function $f(\cdot)$ with a polynomial number of breakpoints denoted $0 = t_0 \ldots t_p = 1$. As with f, let us assume implied subscripts for the moment and let $G(t)$ denote $G_{J\cup\{j\},k,\delta/3}(t)$. Initially we compute $f(t_0) = G(0)/(1+\delta/3)$ by a single invocation of $A_{J\cup\{j\},k,\delta/3}$. We then use binary search to compute each successive breakpoint $t_1 \ldots t_{p-1}$. More precisely, once we have computed t_{i-1}, we determine t_i to be the minimum value of t such that $G(t) < f(t_{i-1})$. We illustrate the construction of $f(\cdot)$ in Figure 4-1.

The maximum number of steps required by the binary search will be polynomial in n and K, since any aggregate sum of n item sizes requires $poly(n,K)$ bits to represent, and each iteration of binary search fixes one of these bits. Each step of the binary search makes one call to $A_{J\cup\{j\},k,\delta/3}$ to evaluate $G(t)$. Note that even though $F_{J\cup\{j\},k}(t)$ is a non-increasing function of t, its approximation $G(t)$ may not be monotonically non-increasing. However, if we ever evaluate $G(t)$ and notice its value is larger than $G(t')$, where $t' < t$ is a point at which we formerly evaluated the function (or similarly, if we notice that $G(t) < G(t')$ and

$t < t'$), then it suffices to use the value of $G(t')$ instead of the value of $G(t)$ — this still yields a $(1+\delta/3)$-approximation to $F_{J\cup\{j\},k}(t)$, and allows us to binary search properly.

Each breakpoint of $f(\cdot)$ marks a drop in $G(\cdot)$ by at least a $(1+\delta/3)$ factor. The value of G can drop by a $(1+\delta/3)$ factor at most a polynomial number of times before it eventually reaches zero. We can argue this inductively, by adding to our inductive hypothesis the claim that G evaluates to a quantity represented by a polynomial number of bits. Since f is a $(1+\delta/3)$-approximation to G, which is in turn a $(1+\delta/3)$-approximation to $F_{J\cup\{j\},k}$, and since $(1+\delta/3)^2 \leq 1+\delta$, we know that $f(t) \in [F_{J\cup\{j\},k}(t)/(1+\delta), F_{J\cup\{j\},k}(t)]$.

Assume for a moment now that j is the best first job to schedule (the one that would be scheduled first by an adaptive policy optimizing $F_{J,k+1}(t)$, we can write $F_{J,k+1}(t)$ as

$$F_{J,k+1}(t) = w_j \mathbf{Pr}[p_j \leq 1-t] + \int_{\tau=0}^{1-t} F_{J\cup\{j\},k}(t+\tau) h_j(\tau) d\tau.$$

where $h_j(\cdot)$ is the probability density function for p_j. Since we are willing to settle for a $(1+\delta)$-approximation, we can use f inside the integral:

$$\begin{aligned} G_{J,k+1,\delta}(t) &= w_j \mathbf{Pr}[p_j \leq 1-t] + \int_{\tau=0}^{1-t} f(t+\tau) h_j(\tau) d\tau \\ &= w_j \mathbf{Pr}[p_j \leq 1-t] + \sum_{i=1}^{p} f(t+t_{i-1}) \mathbf{Pr}[t_{i-1} \leq p_j \leq t_i]. \end{aligned}$$

Maximizing over all potential first items $j \in L\backslash J$ gives the final value of $G_{J,k+1,\delta}(t)$, which is a $(1+\delta)$-approximation to $F_{J,k+1}(t)$. Note that any value of $G_{J,k+1,\delta}(t)$ must be representable by a polynomial number of bits as long as the same is true of $G_{\bullet,k,\delta/3}(t)$ (since we only carry the induction out for a constant number of levels). In total, the algorithm $A_{J,k+1,\delta}$ above makes only a polynomial number of calls to $A_{\bullet,k,\delta/3}$. □

4.3 Special Classes of Distributions

In this section we derive stronger performance guarantees for WSEPT if processing times have particularly "well behaved" distributions. From the previous section, we know how to construct a $(1+\varepsilon)$-approximate policy if all jobs are large. The real difficulty lies with the small jobs, and specifically small jobs with high variability. One way to escape this difficulty is simply to assume that such jobs do not exist. If we assume the support of every processing time p_j lies in some small bounded interval $[0, b]$, then according to the central limit theorem, a sum of such independent random variables should be tightly concentrated about its mean. In this case, we can use appropriate tail inequalities to show that WSEPT obtains an expected value close to $\Phi(1)$ (for sufficiently small b). Note that if all processing times are bounded, then $\mu_j = \mathbf{E}[p_j]$ and $w_j = w'_j$ for all jobs j and also $\Phi^*(\lambda) = \Phi(\lambda)$ for all values of λ.

Let $R = \max_j P_j/\mu_j$, where P_j denotes the maximum value in the support of p_j. We focus

our attention in this section on instances for which $R = O(1)$, since this guarantees that a job j with small μ_j has a bounded support in the small interval $[0, R\mu_j]$. Several common processing time distributions satisfy this condition, including for example the uniform distribution and all symmetric distributions.

Lemma 15. *If $R = O(1)$, then for any small constant $\varepsilon > 0$, there exists a constant $b \in (0, \varepsilon]$ such that WSEPT is a $(1+\varepsilon)$-approximate policy when we restrict our instance to only the set J of jobs with $p_j \in [0, b]$. This result holds in both the start deadline and completion deadline models.*

To prove the lemma, we use a well-known tail inequality due to Hoeffding [30]:

Theorem 8 (Hoeffding). *Let $X_1 \ldots X_n$ be independent random variables each satisfying $X_i \in [a_i, b_i]$. If $X = X_1 + \ldots + X_n$, then*

$$\mathbf{Pr}[X > \mathbf{E}[X] + t] \leq e^{-2t^2/c} \quad \textit{and} \quad \mathbf{Pr}[X < \mathbf{E}[X] - t] \leq e^{-2t^2/c}$$

where $c = \sum_i (b_i - a_i)^2$.

Proof of Lemma 15. We choose $\delta > 0$ to be the constant satisfying $(1+\delta)/(1-\delta) = 1+\varepsilon$. Choose $b \in (0, \varepsilon]$ to be a small enough constant so that

$$e^{-2\left(\frac{\delta}{2}-b\right)^2/bR\left(1-\frac{\delta}{2}+b\right)} \leq \frac{\delta}{2}. \tag{4.13}$$

This is always possible since the left hand side tends to zero in the limit as b decreases. Let J denote the set of all jobs j satisfying $p_j \in [0, b]$. We restrict our focus to the WSEPT ordering of only the jobs in J. Let $F = 1 - \frac{\delta}{2} + b$, and let j be the smallest index in our ordering such that $\sum_{i \leq j} \mu_i \in [F - b, F]$. Applying Hoeffding's bound (note that $c \leq bRF$), we have

$$\begin{aligned}
\mathbf{Pr}[\text{jobs } 1 \ldots j \text{ complete by deadline}] &= 1 - \mathbf{Pr}\left[\sum_{i=1}^{j} p_j > 1\right] \\
&\geq 1 - e^{-2(1-F)^2/c} \\
&\geq 1 - e^{-2(1-F)^2/(bRF)} \\
&\geq 1 - \frac{\delta}{2}. \quad \text{(Using (4.13))}
\end{aligned}$$

The value we obtain if jobs $1 \ldots j$ succeed in completing by the deadline is at least

$$\Phi_J(F - b) \geq (F - b)\Phi_J(1) \geq \left(1 - \frac{\delta}{2}\right)\Phi_J(1).$$

Note that $\Phi_J(1) = \Phi_J^*(1)$ given that our processing time distributions are bounded, so this holds in both the start deadline model and completion deadline model. Therefore, the total

expected value obtained by WSEPT for the jobs in J is at least

$$\left(1-\frac{\delta}{2}\right)^2 \Phi_J(1) \geq (1-\delta)\Phi_J(1),$$

In terms of bounding $ADAPT$, we return to our martingale argument from Section 3.1.1. Since our processing times are bounded by b, we know that $\tau(S_n) \leq 1+b$ (rather than 2), which implies that $\mathbf{E}[\mu(J')] \leq 1+b$, where J' denotes the (random) set of jobs that any given adaptive policy attempts to schedule. Jensen's inequality (see also the proof of Theorem 1) then gives us $ADAPT(J) \leq \Phi_J(1+b) \leq (1+\delta)\Phi_J(1)$, where $ADAPT(J)$ is the expected value obtained by an optimal adaptive policy for just the jobs in J. Therefore, if $WSEPT(J)$ denotes the expected value for WSEPT on the set J, we have

$$ADAPT(J) \leq \frac{1+\delta}{1-\delta} WSEPT(J) \leq (1+\varepsilon)WSEPT(J),$$

which completes the proof. □

Note that the proof above gives us a means of estimating, to within a $(1+\varepsilon)$ factor, the expected value obtained by WSEPT for the jobs in J.

Theorem 9. *If $R = O(1)$, then one can compute an adaptive $(2+\varepsilon)$-approximate policy for any small constant $\varepsilon > 0$ for the problems $1 \mid p_j \sim stoch, d_j = 1 \mid \mathbf{E}[\sum w_j \overline{V}_j]$ and $1 \mid p_j \sim stoch, d_j = 1 \mid \mathbf{E}[\sum w_j \overline{U}_j]$.*

Proof. We invoke Lemma 15 (using $\varepsilon' = \varepsilon/2$ instead of ε) to obtain a constant b and a set J with $p_j \in [0,b]$ for all $j \in J$. Let $ADAPT(J)$ denote the expected value obtained by an optimal adaptive policy for only the jobs in J, and let $WSEPT(J)$ denote the expected value obtained by the WSEPT policy for jobs in J. Then $ADAPT(J) \leq (1+\varepsilon')WSEPT(J)$. Now consider the jobs in $\overline{J}$: these all have support that extends beyond b. We know that $\mu_j \geq b/R$ with $b/R \leq \varepsilon' = O(1)$ for all $j \in \overline{J}$. Let $LARGE(\overline{J})$ denote the expected value for the adaptive policy from Section 4.2 for jobs that are "large" with respect to the constant b/R. We then get $ADAPT(\overline{J}) \leq (1+\varepsilon')LARGE(\overline{J})$. Taking the better of WSEPT for the set J and the adaptive policy for $\overline{J}$, we obtain an expected value $B \geq \max\{WSEPT(J), LARGE(\overline{J})\}$, and

$$\begin{aligned} ADAPT &\leq ADAPT(J) + ADAPT(\overline{J}) \\ &\leq (1+\varepsilon')WSEPT(J) + (1+\varepsilon')LARGE(J) \\ &\leq 2(1+\varepsilon')B \\ &= (2+\varepsilon)B, \end{aligned}$$

which is the desired performance guarantee. □

5. One Machine with Individual Deadlines and Release Times

We begin this chapter by considering stochastic scheduling problems in which all jobs have individual deadlines but a common release time at $t = 0$:

- $1 \mid p_j \sim stoch \mid \mathbf{E}[\sum w_j \overline{V}_j]$ (in the start deadline model), and
- $1 \mid p_j \sim stoch \mid \mathbf{E}[\sum w_j \overline{U}_j]$ (in the completion deadline model).

We then consider the somewhat symmetric problem where jobs have a common deadline at time $t = 1$ but individual release times:

- $1 \mid p_j \sim stoch, r_j, d_j = d \mid \mathbf{E}[\sum w_j \overline{V}_j]$ (in the start deadline model), and
- $1 \mid p_j \sim stoch, r_j, d_j = d \mid \mathbf{E}[\sum w_j \overline{U}_j]$ (in the completion deadline model).

Note that these problems are equivalent due to symmetry in the deterministic case, but they are now quite different in our stochastic setting.

Just as with the stochastic knapsack problem we have considered for the past 2 chapters, our results in this chapter will be simpler and stronger in the start deadline model than in the completion deadline model. In particular, for we will only be able to obtain reasonable results in the completion deadline model if all jobs are small.

5.1 Individual Deadlines

Deterministic scheduling problems in which jobs have individual deadlines have received quite a bit of attention in the literature. It is well known that one can extend the FPTAS for knapsack [31] to obtain an FPTAS for $1 \mid\mid \sum w_j \overline{U}_j$ (and we can extend this to an FPTAS for $1 \mid\mid \sum w_j \overline{V}_j$ by increasing the deadline of each job j by p_j). In the stochastic case, we show how to use a straightforward randomized rounding procedure to obtain an 8-approximate non-adaptive policy for $1 \mid p_j \sim stoch \mid \mathbf{E}[\sum w_j \overline{V}_j]$.

5.1.1 Fractional Relaxations

Consider the following natural generalizations of the linear programming relaxations for $\Phi(\cdot)$ and $\Phi^*(\cdot)$ from Chapter 3 (recall that we define $w'_j = w_j \mathbf{Pr}[p_j \leq d_j]$):

$\Phi(\lambda)$	$=$	$\max \sum_{j=1}^{n} w_j x_j$	$\Phi^*(\lambda)$	$=$	$\max \sum_{j=1}^{n} w'_j x_j$
$\forall j \in [n]$	:	$\sum_{i:d_i \leq d_j} x_i \mathbf{E}[\min(p_i, d_j)] \leq \lambda d_j$	$\forall j \in [n]$	:	$\sum_{i:d_i \leq d_j} x_i \mathbf{E}[\min(p_i, d_j)] \leq \lambda d_j$
$\forall j \in [n]$	:	$0 \leq x_j \leq 1$	$\forall j \in [n]$	:	$0 \leq x_j \leq 1$

Note that the generalized functions $\Phi(\cdot)$ and $\Phi^*(\cdot)$ above are completely equivalent to $\Phi(\cdot)$ and $\Phi^*(\cdot)$ from Chapter 3 in the event that all deadlines coincide. As in Chapter 3, we can use these functions to upper bound the expected value obtained by an optimal adaptive policy. Note that in the completion deadline model, our bound below depends crucially on our assumption that jobs cannot be canceled immediately at their deadlines. Also note that $\Phi(\cdot)$ and $\Phi^*(\cdot)$ are concave due to Lemma 3.

Lemma 16. *In the start deadline model, $ADAPT \leq \Phi(2)$, and in the completion deadline model, $ADAPT \leq \Phi^*(2)$.*

Proof. We use the same argument as in the proof of Theorem 2. Fix an optimal adaptive policy P, and let x_j denote the probability that P attempts to schedule job j. The expected value obtained by P is $\sum x_j w_j$ (start deadline model) or at most $\sum x_j w'_j$ (completion deadline model), and these are precisely the objective functions for $\Phi(2)$ and $\Phi^*(2)$. We now need only to argue that $x_1 \ldots x_n$ is a feasible solution. Consider, for some j, one of the packing constraints from either LP (these constraints are identical for both LPs):

$$\sum_{i:d_i \leq d_j} x_i \mathbf{E}[\min(p_i, d_j)] \leq 2d_j.$$

This is equivalent to the single packing constraint for the stochastic knapsack problem. Since all jobs i with $d_i \leq d_j$ must be launched prior to time d_j, Lemma 1 establishes that this constraint would be valid for a problem in which all jobs in $\{i : d_i \leq d_j\}$ share a common deadline of d_j. In our case, some of these jobs may have earlier deadlines, but this does not affect validity of the constraint — if anything, these earlier deadlines could only further strengthen the constraint. □

5.1.2 Randomized Rounding

We now work toward an 8-approximate non-adaptive policy for $1 \mid p_j \sim stoch \mid \mathbf{E}[\sum w_j \overline{V}_j]$. As an upper bound on the expected value of an optimal adaptive policy, we use $ADAPT \leq \Phi(2)$ from Lemma 16. Starting with a fractional solution $x_1 \ldots x_n$ for $\Phi(1)$, we consider each job j in sequence (ordered by their deadlines), flip a coin, and with probability $x_j/2$ we

schedule j. Since we do not pay any attention to j's deadline, it is entirely possible that we may schedule j after its deadline has already expired. However, since we can perform the coin flips in advance, the resulting policy (which we call the *randomized rounding* policy) is truly non-adaptive, so we establish a bound of 8 on the adaptivity gap of $1 \mid p_j \sim stoch \mid \mathbf{E}[\sum w_j \overline{V}_j]$. If we are allowed to use a limited amount of adaptivity and avoid scheduling jobs whose deadlines have already expired, then our results can only improve.

Theorem 10. *Take any feasible fractional solution $x_1 \ldots x_n$ for $\Phi(1)$. In the start deadline model, the randomized rounding policy described above obtains an expected value of at least one quarter of the LP objective value for x — that is, at least $\sum x_j w_j/4$.*

Proof. Let us index our jobs according to their deadlines (the ordering used by the randomized rounding policy). We extend our definition of M_j as follows:

$$M_j = \sum_{i=1}^{j} x_i \mathbf{E}[\min(p_i, d_j)].$$

Note that $M_j \leq d_j$ since $x_1 \ldots x_n$ is feasible for the linear program for $\Phi(1)$. Recall that for the stochastic knapsack problem with common deadline at time $t = d$, Lemma 4 tells us that job j is successfully scheduled in the start deadline model with probability at least $1 - M_{j-1}/d$. Here, we modify this statement to say that job j *would* be scheduled by its deadline (that is, if we were to include j according to its coin flip) with probability at least $1 - M_{j-1}/(2d_j) \geq 1/2$. To show this, let P_j be a random variable whose value is the sum of $\min(p_i, d_j)$ for all jobs $i \leq j$ that are selected via coin flip by the randomized rounding policy. Let X_j be an indicator random variable that takes the value 1 if job j *would* be scheduled prior to its deadline (if the randomized rounding policy were to select j via coin flip). Then,

$$\begin{aligned}
\mathbf{E}[X_j] &= 1 - \mathbf{Pr}[P_{j-1} > d_j] \\
&\geq 1 - \mathbf{E}[P_{j-1}]/d_j \quad \text{(Markov's inequality)} \\
&\geq 1 - M_{j-1}/(2d_j) \\
&\geq 1/2,
\end{aligned}$$

since $\mathbf{E}[P_{j-1}] \leq M_{j-1}/2$. Let W_j be a random variable taking the value 1 (with probability $x_j/2$, if the coin flip for job j says to schedule j) or 0 (with probability $1 - x_j/2$, if the coin flip tells us to leave out j). Job j will be successfully scheduled only if both W_j and X_j deliver positive indications. Since W_j and X_j are independent (this is true only in the start deadline model!), we can write the expected value obtained by the randomized rounding policy as

$$\mathbf{E}\left[\sum_{j=1}^{n} w_j W_j X_j\right] = \sum_{j=1}^{n} w_j \mathbf{E}[W_j]\mathbf{E}[X_j] \geq \frac{1}{4}\sum_{j=1}^{n} w_j x_j,$$

and this is precisely one quarter of the objective value for $\Phi(1)$ for x. □

Corollary 6. *The non-adaptive randomized rounding policy is an 8-approximation algorithm for* $1 \mid p_j \sim stoch \mid \mathbf{E}[\sum w_j \overline{V}_j]$.

Using the method of conditional expectations with pessimistic estimators, we can derandomize the argument above and obtain a deterministic 8-approximation algorithm.

5.1.3 Difficulties with the Completion Deadline Model

The completion deadline model seems more difficult to accommodate in the case of individual deadlines. In particular, our familiar LP-based analysis based on $\Phi^*(\cdot)$ no longer seems powerful enough to give us a strong approximation guarantee. According to the following lemma, one cannot hope for a better approximation guarantee than $\Omega(n)$ in the worst case if we base our analysis on $\Phi^*(\cdot)$.

Lemma 17. *For the deterministic problem* $1 \mid\mid \sum w_j \overline{U}_j$ *the linear program for* $\Phi^*(1)$ *has an integrality gap of* $\Theta(k)$*, where* k *denotes the number of distinct deadlines among all jobs.*

Proof. To show that the integrality gap is at least k, consider a deterministic instance with k jobs having $p_j = d_j = 2^j$ and unit value. In this case, an optimal integral schedule includes only one job (which blocks all others from being scheduled), but $\Phi^*(1) \geq n/2$ since $x_j = 1/2$ for all $j \in [k]$ is a feasible solution. The integrality gap is at most $k+1$ since we can take an optimal fractional solution for $\Phi^*(1)$ and upper bound its value by the value of a union of $k+1$ feasible integral solutions. Let us assume without loss of generality that we start with an optimal basic fractional solution for $\Phi^*(1)$ in which $x_j = 0$ if $p_j > d_j$, so any single job j for which $x_j > 0$ is guaranteed not to exceed its deadline when scheduled in isolation. Since the LP for $\Phi^*(1)$ has effectively k inequality constraints, any basic feasible solution x will contain at most k fractional variables $0 < x_j < 1$. We take each of these k fractional variables x_j and turn each one into a "single job" schedule that includes only job j by itself. That gives us k feasible integer solutions. For the last solution, we take the set of all jobs j for which $x_j = 1$. The total value of all of these $k+1$ integral solutions is clearly an upper bound on $\Phi^*(1)$. □

The large integrality gap for the LP for $\Phi^*(1)$ seems to only be a problem for large jobs. For small jobs, we can still apply randomized rounding somewhat successfully.

Theorem 11. *Suppose that for all jobs* j *we have* $\mu_j \leq \varepsilon d_j$ *(i.e., that all jobs are "small" with respect to a cutoff* ε *compared to their own deadlines). Let* $q = (1-\varepsilon)/2$*, and let us perform randomized rounding by taking the jobs in order of their deadlines and including each job* j *independently with probability* qx_j*, where* $x_1 \ldots x_n$ *is an optimal solution to* $\Phi^*(1)$*. This gives us a performance guarantee of* $8/(1-\varepsilon)^2$ *for the problem* $1 \mid p_j \sim stoch \mid \mathbf{E}[\sum w_j \overline{U}_j]$.

Proof. We generalize slightly the proof of Theorem 10. Define the indicator random variable W_j to be 1 with probability $x_j/2$ and zero otherwise, and as before, let the indicator random

variable X_j tell us whether or not job j *would* be successfully scheduled (that is, if the randomized rounding policy chooses to include j). Now,

$$\begin{aligned}
\mathbf{E}[X_j] &= 1 - \mathbf{Pr}[P_{j-1} + p_j] > d_j] \\
&= 1 - \mathbf{Pr}[P_{j-1} + \min(p_j, d_j)] > d_j] \\
&\geq 1 - \frac{\mathbf{E}[P_{j-1}] + \mathbf{E}[\min(p_j, d_j)]}{d_j} \quad \text{(Markov's inequality)} \\
&= 1 - \frac{qM_{j-1}}{d_j} - \frac{\mu_j}{d_j} \\
&\geq 1 - q - \varepsilon.
\end{aligned}$$

Since W_j and our modified version of X_j are independent, the total expected value we obtain is

$$\begin{aligned}
\mathbf{E}\left[\sum_{j=1}^{n} w'_j W_j X_j\right] &= \sum_{j=1}^{n} w'_j \mathbf{E}[W_j]\mathbf{E}[X_j] \\
&\geq \sum_{j=1}^{n} q w'_j x_j (1 - q - \varepsilon) \\
&\geq q(1 - q - \varepsilon)\Phi^*(1) \\
&\geq \frac{q(1 - q - \varepsilon)}{2} ADAPT.
\end{aligned}$$

By optimally setting $q = (1 - \varepsilon)/2$, we obtain the desired performance guarantee. □

5.2 Individual Release Times

Let us now consider problems with individual release times $r_1 \ldots r_j$ and a common deadline at time $t = d$. As before, we consider only the start deadline model, so the problem under consideration is $1 \mid p_j \sim stoch, r_j, d_j = d \mid \mathbf{E}[\sum w_j \overline{V}_j]$. The natural LP relaxation for this problem is symmetric to the one from the previous section:

$$\begin{array}{lcl}
\Phi(\lambda) & = & \max \sum_{j=1}^{n} w_j x_j \\
\forall j \in [n] & : & \sum_{i:r_i \geq r_j} x_i \mathbf{E}[\min(p_i, d - r_j)] \leq \lambda(d - r_j) \\
\forall j \in [n] & : & 0 \leq x_j \leq 1
\end{array}$$

By a symmetric argument to the one given in the proof of Lemma 16, we have $ADAPT \leq \Phi(2)$. Unfortunately, whereas in the previous section we can achieve at least $\Phi(1)/4$ in expected value by randomized rounding, here we find that even in the deterministic case, the linear program for $\Phi(1)$ has an integrality gap of $\Omega(n)$. Consider a deterministic instance with n unit-value jobs where $d = 2^n$ and for each $j \in [n]$ we have $p_j = 2^j$ and $r_j = d - 2^j$.

An optimal solution can schedule at most two jobs (one of which is scheduled right at its deadline), while the feasible fractional solution that assigns $x_j = 1/2$ for all jobs obtains a value of $n/2$.

5.2.1 Pseudo-Approximation Algorithms

The integrality gap of the LP for $\Phi(1)$ rules out the possibility of obtaining any reasonable approximation guarantee by rounding a fractional solution to the LP. However, it does not rule out the possibility of developing pseudo-approximation algorithms. Suppose we "stretch" the deadline of each job by a factor of β; that is, we assign each job j a new deadline $d'_j = r_j + \beta(d_j - r_j)$. We say that a policy is an (α, β)-approximation algorithm if, using the new deadlines $d'_1 \ldots d'_n$, it can obtain an expected value that is at least an α fraction of $ADAPT$ (where $ADAPT$ always denotes the optimal expected value we can obtain using our *original* deadlines, which in this case all coincide at time $t = d$).

Theorem 12. *Randomized rounding of jobs in non-increasing order of release time, using a fair coin, gives a non-adaptive policy with an expected performance guarantee of* $(8, 2)$.

Proof. As long as our jobs initially share a common deadline, the act of stretching out deadlines gives us much more slack for jobs with earlier release times. In fact, if we stretch deadlines by a factor of $\beta = 2$, then the new deadlines $d'_1 \ldots d'_n$ become the "mirror image" of the release times $r_n \ldots r_1$ with respect to the original deadline $t = d$. Therefore, if we focus on times $t \geq d$ in the stretched instance, we find ourselves faced with an instance of $1 \mid p_j \sim stoch \mid \mathbf{E}[\sum w_j \overline{V}_j]$, since all jobs are available at time $t = d$ and each job has an individual deadline d'. Due to symmetry, the optimal solution $x_1 \ldots x_n$ for the LP for $\Phi(1)$ for the original problem will be an optimal solution to the LP for $\Phi(1)$ for the new problem, so by randomized rounding (starting at time $t = d$) according to the order $d'_1 \leq \ldots \leq d'_n$ (i.e., non-increasing order of release times), Theorem 10 ensures that we obtain at least an expected value of $\Phi(1)/4$, which is at least $ADAPT/8$. The randomized rounding approach described above starts scheduling at time $t = r_n$ rather than $t = d$, but this can only serve to improve our expected value. □

We note that the preceding approach also gives a performance guarantee of $\left(\frac{8}{(1-\varepsilon)^2}, 2\right)$ in the completion deadline model if all jobs are small.

5.2.2 Interval Scheduling

The natural problem to consider next is $1 \mid p_j \sim stoch, r_j \mid \mathbf{E}[\sum w_j \overline{V}_j]$, where jobs have individual release times and deadlines. In fact, one can generalize this further to allow each job to have an associated set of intervals of time during which it can be launched. In a deterministic setting, such "interval scheduling" problems can be approximated to within a factor of $(2 + \varepsilon)$ [1]. However, in the stochastic case we currently do not know how to obtain approximation algorithms (or even pseudo-approximation algorithms) with good

performance guarantees for these problems. One can still bound $ADAPT$ using a linear programming relaxation, as we have done before. The difficulty seems to be on the other side of the equation, in analyzing the expected value obtained by a "rounded" policy.

6. Parallel Machines

In this chapter we generalize the LP-based non-adaptive policies from Chapter 3 to parallel machine models. One might ideally like to generalize the combinatorial results from Chapter 4, since these give us stronger performance guarantees in the single machine case; however, at the present time there does not seem to be a straightforward way of doing this.

The simplest parallel machine problems we consider are those involving identical machines (we again assume by scaling that $d = 1$ throughout this chapter whenever we have common deadlines):

- $P \mid p_j \sim stoch, d_j = 1 \mid \mathbf{E}[\sum w_j \overline{V}_j]$ (in the start deadline model), or
- $P \mid p_j \sim stoch, d_j = 1 \mid \mathbf{E}[\sum w_j \overline{U}_j]$ (in the completion deadline model).

We consider two types of policies for these problems: "global" list scheduling policies that sequentially assign jobs to machines based on a single global ordering of all jobs, and non-adaptive "pre-assignment" policies that initially construct disjoint orderings of jobs to schedule on each of the machines.

Moving along from identical machines, we then consider pre-assignment policies for problems in the unrelated parallel machine environment:

- $R \mid p_j \sim stoch, d_j = 1 \mid \mathbf{E}[\sum w_j \overline{V}_j]$ (in the start deadline model), or
- $R \mid p_j \sim stoch, d_j = 1 \mid \mathbf{E}[\sum w_j \overline{U}_j]$ (in the completion deadline model).

Finally, we remove the $d_j = 1$ constraint and generalize our pre-assignment policies for both identical and unrelated machines to the case where jobs have individual deadlines.

6.1 Identical Parallel Machines: Fractional Relaxations

Just as in the single machine case, linear programming will serve as an important tool for bounding the performance of an optimal adaptive policy in parallel environments. Letting m denote the number of machines in our environment, consider the following linear programs:

$\Psi(\lambda)$	$=$	$\max \sum_{j=1}^{n} w_j x_j$	$\Psi^*(\lambda)$	$=$	$\max \sum_{j=1}^{n} w'_j x_j$
		$\sum_{j=1}^{n} x_j \mu_j / m \leq \lambda$			$\sum_{j=1}^{n} x_j \mu_j / m \leq \lambda$
$\forall j \in [n]$	:	$0 \leq x_j \leq 1$	$\forall j \in [n]$	:	$0 \leq x_j \leq 1$

The functions $\Psi(\cdot)$ and $\Psi^*(\cdot)$ as defined above are formulated for the identical parallel machine environment; later in the chapter, we will generalize these functions to handle the case of unrelated parallel machines. Just as in Chapter 3, we define $\Psi_J(\cdot)$ and $\Psi^*_J(\cdot)$ by restricting the linear programs above to a particular set of jobs J. Finally, note that $\Psi(\lambda) \geq \Psi^*(\lambda)$ for all $\lambda \geq 0$, that $\Psi(\cdot)$ and $\Psi^*(\cdot)$ are concave (Lemma 3), and that for $m = 1$, these functions reduce back to our familiar $\Phi(\cdot)$ and $\Phi^*(\cdot)$ functions from Chapter 3.

Theorem 13. *For the problem $P \mid p_j \sim stoch, d_j = 1 \mid \mathbf{E}[\sum w_j \overline{V}_j]$, we have $ADAPT \leq \Psi(2)$, and for the problem $P \mid p_j \sim stoch, d_j = 1 \mid \mathbf{E}[\sum w_j \overline{U}_j]$, we have $ADAPT \leq \Psi^*(2)$.*

Proof. Fix an optimal adaptive policy P and let x_j denote the probability that P attempts to schedule job j no later than the deadline. Since our m machines are identical, by symmetry we can assume that x_j/m is the probability that P attempts to schedule job j on some particular machine i. Letting J_i denote the (random) set of jobs that P attempts to schedule on machine i, we know by Lemma 1 that $\sum x_j \mu_j / m = \mathbf{E}[\mu(J_i)] \leq 2$. Therefore, $x_1 \ldots x_n$ is a feasible solution for the LPs for $\Psi(2)$ and $\Psi^*(2)$. The expected value obtained by P in the start deadline model is $\sum x_j w_j$, which is precisely the objective value for the LP for $\Psi(2)$. In the completion deadline model, the expected value obtained by P is at most $\sum x_j w'_j$ (using the same argument as in the proof of Theorem 1), and this is the objective value of the LP for $\Psi^*(2)$. □

6.2 Global List Scheduling

Global list scheduling according to the WSEPT ordering is perhaps the most natural and simplest approach for scheduling on identical parallel machines. In the start deadline model, we show that this gives us a performance guarantee of 4, while in the completion deadline model we obtain a performance guarantee of $5.5 + 2.5\sqrt{3} < 9.831$ by using the better of the global list scheduling policy according to WSEPT ordering (for small jobs) and a simple policy that schedules only a single job on each machine. Recall that the WSEPT ordering differs slightly between the start deadline and completion deadline models. In the start deadline model, we use the ordering $\frac{w_1}{\mu_1} \geq \frac{w_2}{\mu_2} \geq \ldots \geq \frac{w_n}{\mu_n}$, and in the completion deadline model, we use the ordering $\frac{w'_1}{\mu_1} \geq \frac{w'_2}{\mu_2} \geq \ldots \geq \frac{w'_n}{\mu_n}$. Let us now define

$$M_j = \frac{1}{m} \sum_{i=1}^{j} \mu_i,$$

where jobs are indexed according to the WSEPT ordering (in the completion deadline model, we only use WSEPT to schedule small jobs, so in this model the definition above

should be applied only to the WSEPT-indexed ordering of small jobs).

Lemma 18. *The probability that jobs $1 \ldots j$ are successfully scheduled by global list scheduling based on the WSEPT ordering is at least $1 - M_{j-1}$ in the start deadline model, and at least $1 - M_{j-1} - \mu_j$ in the completion deadline model.*

Proof. Let us assume for convenience that our list scheduling policy runs to completion and schedules all jobs — this gives us a clear notion of the expected time at which job j is scheduled. We denote this time by the random variable T_j. We also assume for convenience that the true processing time of each job j is $\min(p_j, 1)$ rather than p_j; note that this has no impact on the scheduling time for any job that starts or completes prior to our deadline $d = 1$. Now fix a job j, and let $t_1 \ldots t_m$ be random variables whose values give the times at which machines $1 \ldots m$ would finish processing if we were to stop our list scheduling policy at job $j - 1$. We then have

$$\mathbf{E}[T_j] = \mathbf{E}\left[\min_{i \in [m]} t_i\right] \leq \mathbf{E}\left[\frac{1}{m}\sum_{i=1}^{m} t_i\right] = \mathbf{E}\left[\frac{1}{m}\sum_{k=1}^{j-1} \min(p_k, 1)\right] = M_{j-1}.$$

In the start deadline model, we fail to schedule job j with probability $\mathbf{Pr}[T_j > 1] \leq \mathbf{E}[T_j] \leq M_{j-1}$, so job j succeeds with probability at least $1 - M_{j-1}$. In the completion deadline model, the probability that job j fails to complete by the deadline $d = 1$ is

$$\begin{aligned} \mathbf{Pr}[T_j + p_j > 1] &= \mathbf{Pr}[T_j + \min(p_j, 1) > 1] \\ &\leq \mathbf{E}[T_j + \min(p_j, 1)] \quad \text{(Markov's inequality)} \\ &\leq M_{j-1} + \mu_j. \end{aligned}$$

Therefore, job j completes by time 1 with probability at least $1 - M_{j-1} - \mu_j$. □

6.2.1 Results in the Start Deadline Model

Just as in the single machine case, the start deadline model allows for a simpler analysis and a better performance guarantee. In fact, our analysis in the parallel case is essentially identical to that of the single machine case.

Lemma 19. *Global list scheduling according to the WSEPT ordering delivers a performance guarantee of 4 for the problem $P \mid p_j \sim stoch, d_j = 1 \mid \mathbf{E}[\sum w_j \overline{V}_j]$.*

Proof. Perhaps the simplest way to prove the lemma is to argue that our analysis in this case is equivalent to the case of applying WSEPT to the single-machine instance of $1 \mid p_j \sim stoch, d_j = 1 \mid \mathbf{E}[\sum w_j \overline{V}_j]$ in which every μ_j is reduced by a factor of m. Note that $\Psi(\lambda)$ is identical to $\Phi(\lambda)$ for this single-machine instance, so our upper bound $ADAPT \leq \Psi(2)$ coincides exactly with the single-machine upper bound $ADAPT \leq \Phi(2)$. Moreover, there is a direct correspondence between the analogous expressions that we use to lower bound the expected value obtained by WSEPT. For the parallel case, Lemma 18 tells us that the

expected value obtained by WSEPT is at least

$$\sum_{j=1}^{r} w_j(1 - M_{j-1}) = \sum_{j=1}^{r} w_j \left(1 - \sum_{k=1}^{j-1} \mu_k/m\right), \tag{6.1}$$

where r is the largest index such that $M_{r-1} = \sum_{k=1}^{r-1} \mu_k/m \leq 1$. In the single machine case, Lemma 4 tells us that WSEPT obtains an expected value of at least

$$\sum_{j=1}^{r'} w_j \left(1 - \sum_{k=1}^{j-1} \mu_k\right), \tag{6.2}$$

where r' is the largest index such that $\sum_{k=1}^{r'-1} \mu_k \leq 1$. The two expressions (6.1) and (6.2) become identical if we define μ_j for the single-machine case as μ_j/m from the parallel case. We now apply the same geometric proof as in Lemma 5 to argue that there is a gap of at most a factor of 4 between these corresponding upper and lower bounds. □

6.2.2 Results in the Completion Deadline Model

In the completion deadline model, just as in the single machine case we need to consider the better of two policies. The first is the WSEPT policy for small jobs (remember the WSEPT ordering is now based on w'_j rather than w_j), and the second is a "single job" policy that assigns one job per machine. As before, each of these policies, taken individually, can be arbitrarily bad in terms of performance guarantee.

The Single Job Policy. We denote by $SINGLE$ the expected value we get by scheduling the best set of m single large jobs, one per machine. Note that in practice, we would use the best m jobs overall, but for our analysis it suffices to consider only large jobs. If the large jobs in our instance (denoted, as usual, by the set L) are ordered so that $w'_1 \geq w'_2 \geq \ldots \geq w'_n$, then we can define $SINGLE = w'_1 + \ldots + w'_m$. An equivalent way to compute $SINGLE$ is using an LP:

$$\begin{array}{rcl} SINGLE & = & \max \sum_{j \in L} w'_j x_j \\ & & \sum_{j \in L} x_j/m \leq 1 \\ \forall j \in L & : & 0 \leq x_j \leq 1. \end{array}$$

Note the strong similarity between this LP and the LP for $\Psi^*(\cdot)$. In fact, if all jobs are large (i.e., $\mu_j \geq \varepsilon$ for all $j \in [n]$), then the LP for $SINGLE$ is a relaxation of the LP for $\Psi^*_L(\varepsilon)$, so $SINGLE \geq \Psi^*_L(\varepsilon)$. Due to the concavity of $\Psi^*(\cdot)$, we see that if all jobs are large, then the single job policy gives a performance guarantee of $2/\varepsilon$ compared to $ADAPT$.

WSEPT. Suppose we use global list scheduling to schedule the small jobs in our instance according to the WSEPT ordering. If we set r be the largest index in the WSEPT ordering

of small jobs so that $M_{r-1} \leq 1 - \varepsilon$, then according to Lemma 18,

$$\begin{aligned} WSEPT &= \sum_{j=1}^{r} w'_j(1 - M_{j-1} - \mu_j) \\ &\geq \sum_{j=1}^{r} w'_j(1 - M_{j-1} - \varepsilon) \end{aligned} \tag{6.3}$$

is a lower bound on the expected value we obtain.

The Best-of-2 Policy. We combine the two policies above by computing $SINGLE$ and $WSEPT$, and by running the policy corresponding to whichever quantity is larger. This gives us an expected value of at least B, where $B = \max(SINGLE, WSEPT)$.

Lemma 20. *For any set J of small jobs,*

$$w'(J) \leq \left(\frac{1 - \varepsilon + 2\mu(J)/m}{(1-\varepsilon)^2}\right) WSEPT. \tag{6.4}$$

As a consequence, if we let J_S denote the (random) set of small jobs scheduled by an adaptive policy P, then

$$\mathbf{E}[w'(J_S)] \leq \left(\frac{1 - \varepsilon + 2\mathbf{E}[\mu(J_S)]/m}{(1-\varepsilon)^2}\right) WSEPT. \tag{6.5}$$

Lemma 21. *Fix any adaptive policy P, and let J_L denote the (random) set of large jobs that P attempts to schedule. Then*

$$\mathbf{E}[w'(J_L)] \leq \left(1 + \frac{\mathbf{E}[\mu(J_L)]}{m\varepsilon}\right) SINGLE. \tag{6.6}$$

Theorem 14. *The best-of-2 policy based on global list scheduling delivers a performance guarantee of $5.5 + 2.5\sqrt{3} < 9.831$ for the problem $P \mid p_j \sim stoch, d_j = 1 \mid \mathbf{E}[\sum w_j \overline{U}_j]$, if we set $\varepsilon = 2 - \sqrt{3}$.*

Proof. Fix an optimal adaptive policy P. Let J_S and J_L denote the (random) sets of small and large jobs that P attempts to schedule, and let $J = J_S \cup J_L$. We argue, as in Theorem 2, that $ADAPT \leq \mathbf{E}[w'(J)]$. Now,

$$\begin{aligned} ADAPT &\leq \mathbf{E}[w'(J_S)] + \mathbf{E}[w'(J_L)] \\ &\leq \left(\frac{1 - \varepsilon + 2\mathbf{E}[\mu(J_S)]/m}{(1-\varepsilon)^2}\right) WSEPT + \left(1 + \frac{\mathbf{E}[\mu(J_L)]}{m\varepsilon}\right) SINGLE \\ &\leq \left(\frac{1}{1-\varepsilon} + 1 + \max\left(\frac{2}{(1-\varepsilon)^2}, \frac{1}{\varepsilon}\right) \mathbf{E}\left[\frac{\mu(J_S \cup J_L)}{m}\right]\right) B \\ &\leq \left(\frac{1}{1-\varepsilon} + 1 + 2\max\left(\frac{2}{(1-\varepsilon)^2}, \frac{1}{\varepsilon}\right)\right) B \quad \text{(Using Lemma 1)} \\ &\leq (5.5 + 2.5\sqrt{3})B \end{aligned}$$

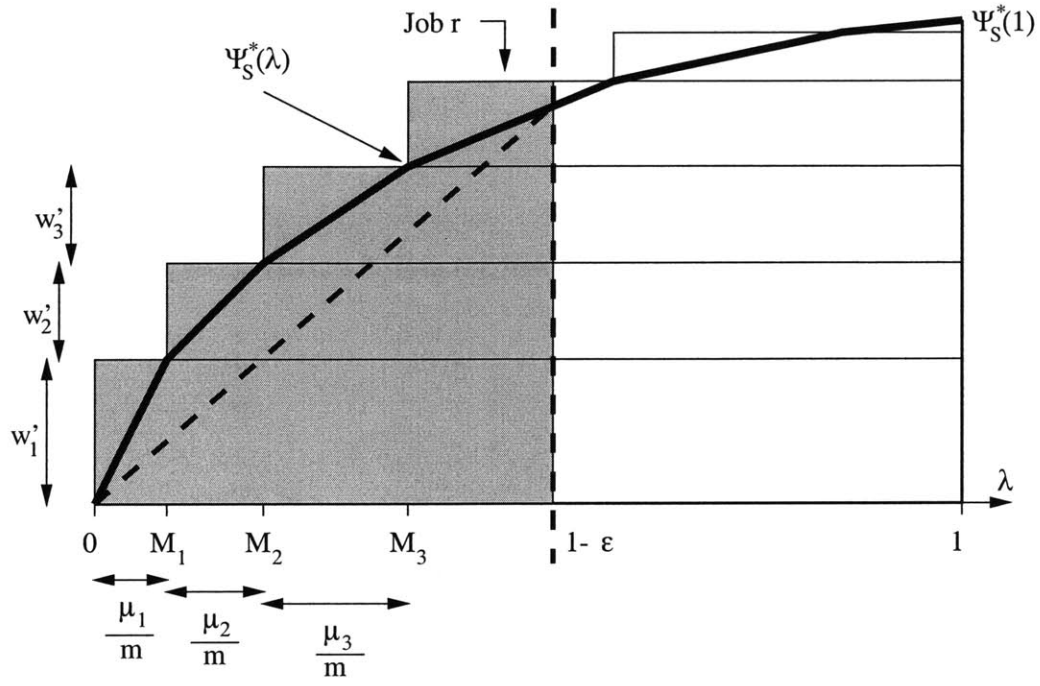

Figure 6-1: The concave, piecewise linear function $\Psi_S^*(\lambda)$. The value of $WSEPT$ is at least the area of the shaded part of the diagram, which in turn is at least as large as the dashed triangle of area $\frac{1-\varepsilon}{2}\Psi_S^*(1-\varepsilon)$.

for $\varepsilon = 2 - \sqrt{3} \approx 0.268$. □

Before proving Lemmas 20 and 21, we need one small auxiliary Lemma.

Lemma 22. $WSEPT \geq \frac{1-\varepsilon}{2}\Psi_S^*(1-\varepsilon)$.

Proof. Consider Figure 6-1. Recall our definition of r as the largest index so that $M_{r-1} \leq 1-\varepsilon$. Also recall from (6.3) that $WSEPT \geq \sum_{j=1}^{r} w_j'(1 - M_{j-1} - \varepsilon)$, and this is precisely the shaded area in the figure, which in turn is at least as large as the dashed triangle of area $\frac{1-\varepsilon}{2}\Psi_S^*(1-\varepsilon)$. □

As a consequence of Lemma 22 we now see that if all jobs are small,

$$ADAPT \leq \Psi^*(2) = \Psi_S^*(2) \leq \frac{2}{1-\varepsilon}\Psi_S^*(1-\varepsilon) \leq \left(\frac{2}{1-\varepsilon}\right)^2 WSEPT,$$

so this implies a performance guarantee of $\left(\frac{2}{1-\varepsilon}\right)^2$ in the case where all jobs are small.

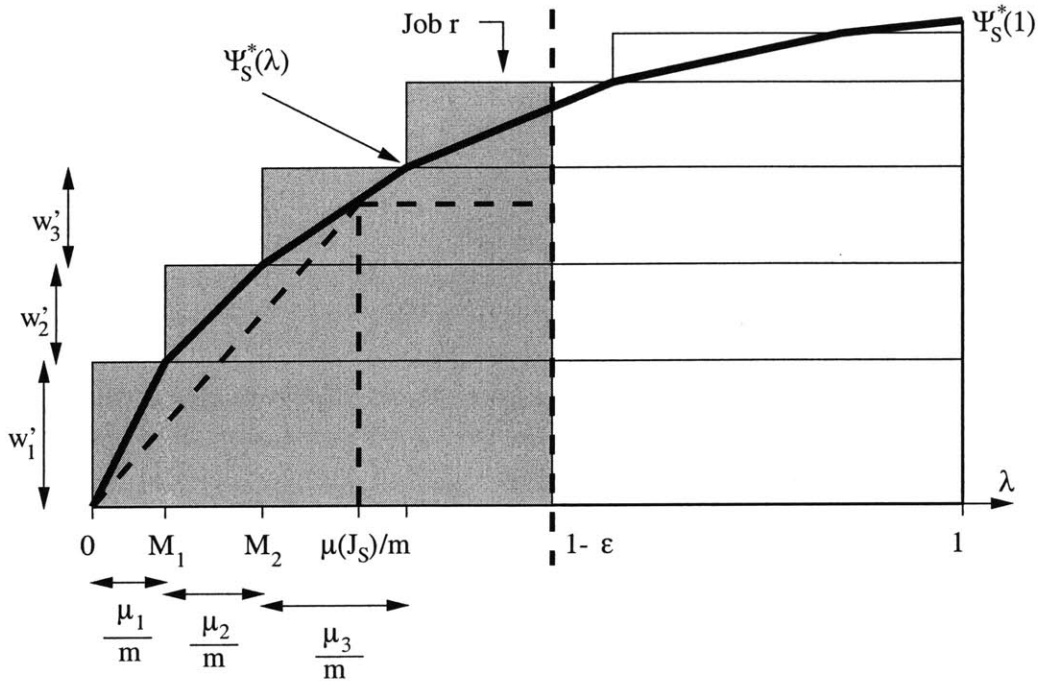

Figure 6-2: Case 2 of the proof of Lemma 20.

Proof of Lemma 20. Let J be any set of small jobs and consider the following three cases:

1. $\mu(J)/m \geq 1-\varepsilon$. In this case, we have

$$w'(J) \leq \Psi_S^*(\mu(J)/m) \leq \frac{\mu(J)/m}{1-\varepsilon}\Psi_S^*(1-\varepsilon) \leq \frac{2\mu(J)/m}{(1-\varepsilon)^2} WSEPT.$$

2. $\mu(J)/m \leq 1-\varepsilon$. Consider the diagram shown in Figure 6-2. By the argument from the proof of Lemma 22, we know that the shaded area is a lower bound on $WSEPT$. We can lower bound this area by the dashed trapezoid of area $[(1-\varepsilon) - \mu(J)/(2m)]\Psi_S^*(\mu(J)/m)$. Therefore,

$$w'(J) \leq \Psi_S^*(\mu(J)/m) \leq \left(\frac{2}{2(1-\varepsilon)-\mu(J)/m}\right) WSEPT.$$

The quantity $\frac{2}{2(1-\varepsilon)-\mu(J)/m}$ is a convex function of $\mu(J)$, so on the interval $\mu(J) \in [0,(1-\varepsilon)m]$ we upper bound it by a linear function:

$$\frac{2}{2(1-\varepsilon)-\mu(J)/m} \leq \frac{1-\varepsilon+\mu(J)/m}{(1-\varepsilon)^2} \leq \frac{1-\varepsilon+2\mu(J)/m}{(1-\varepsilon)^2}.$$

In both cases, the desired bound holds. □

Proof of Lemma 21. Having fixed an adaptive policy P, we let J_L denote the (random) set of large jobs that P attempts to schedule, and x_j the probability that P attempts to schedule job j. Recall from the proof of Theorem 2 that $\mathbf{E}[w'(J_L)] \leq \sum_{j \in L} x_j w'_j$. Therefore,

$$\begin{aligned}
\mathbf{E}[w'(J_L)] &\leq \sum_{j \in L} x_j w'_j \\
&\leq \Psi^*_L \left(\frac{\sum_{j \in L} x_j \mu_j}{m} \right) \\
&= \Psi^*_L \left(\frac{\mathbf{E}[\mu(J_L)]}{m} \right) \\
&= \Psi^*_L \left(\frac{\varepsilon \mathbf{E}[\mu(J_L)]}{m\varepsilon} \right) \\
&\leq \Psi^*_L \left(\varepsilon \max \left(1, \frac{\mathbf{E}[\mu(J_L)]}{m\varepsilon} \right) \right) \\
&\leq \max \left(1, \frac{\mathbf{E}[\mu(J_L)]}{m\varepsilon} \right) \Psi^*_L(\varepsilon) \\
&\leq \max \left(1, \frac{\mathbf{E}[\mu(J_L)]}{m\varepsilon} \right) SINGLE \\
&\leq \left(1 + \frac{\mathbf{E}[\mu(J_L)]}{m\varepsilon} \right) SINGLE,
\end{aligned}$$

and this is the desired bound. □

6.3 Pre-Assignment by List Scheduling

Having investigated "global" list scheduling policies based on a single list, we spend the remainder of this chapter discussing non-adaptive "local" pre-assignment policies that initially partition our jobs into disjoint sets $J_1 \ldots J_m$, one for each machine. Scheduling then proceeds in a non-adaptive fashion on each machine.

In this section, we compute $J_1 \ldots J_m$ by global list scheduling according to the WSEPT ordering. In contrast to the preceding discussion, however, we run global list scheduling not in "real time", but in advance, treating the processing time of each job j as μ_j. More precisely, we consider the jobs in the WSEPT ordering (note this is slightly different for the start deadline and completion deadline models). For each job, we assign it to the set J_i ($1 \leq i \leq m$) for which $\mu(J_i)$ is currently smallest. The process stops when $\mu(J_i) \geq 1$ for all $i \in [m]$ (if we run out of jobs before this condition is met, we can continue to make progress by using *dummy* jobs of zero value), and we let J' denote the set of any "leftover" jobs that have not yet been assigned. Since $\mu_j \leq 1$ for all jobs j, the pre-assignment algorithm terminates with $\mu(J_i) \in [1, 2]$ for all $i \in [m]$.

6.3.1 Identical Parallel Machines

Recall the definitions of the LPs for $\Phi_J(d)$ and $\Phi^*_J(d)$ from Chapter 3:

$\Phi_J(d)$	=	$\max \sum_{j \in J} w_j x_j$	$\Phi^*_J(d)$	=	$\max \sum_{j \in J} w'_j x_j$
		$\sum_{j \in J} x_j \mu_j \leq d$			$\sum_{j \in J} x_j \mu_j \leq d$
$\forall j \in J$	:	$0 \leq x_j \leq 1$	$\forall j \in J$	:	$0 \leq x_j \leq 1$

Lemma 23. *For disjoint sets $J_1 \ldots J_m$ and J' as computed by list scheduling using the WSEPT ordering, we have*

- $\Psi(d) \leq \sum_{i=1}^{m} \Phi_{J_i \cup J'}(d)$ *in the start deadline model, and*
- $\Psi^*(d) \leq \sum_{i=1}^{m} \Phi^*_{J_i \cup J'}(d)$ *in the completion deadline model,*

for any $d \geq 2$.

Proof. Let us focus on the start deadline model, as the same argument applies in the completion deadline model. An optimal fractional solution for $\Psi(d)$ with $d \geq 2$ includes each job in $J_1 \cup \ldots \cup J_m$ to its full extent, as well as the most valuable $md - \mu(J_1 \cup \ldots \cup J_m)$ units worth of "mass" from J'. That is, the solution with respect to jobs in J' can be computed by solving the LP for $\Psi_{J'}(d - \mu(J_1 \cup \ldots \cup J_m)/m)$, which one can view as the "residual" problem once the jobs in $J_1 \cup \ldots \cup J_m$ are fixed. Now let us consider the solutions of $\Phi_{J_i \cup J'}(d) \ldots \Phi_{J_m \cup J'}(d)$. Since $d \geq 2$ and $\mu(J_i) \in [1, 2]$, we know that these solutions together also include all jobs in $J_1 \cup \ldots \cup J_m$ to their full extent, followed by $md - \mu(J_1 \cup \ldots \cup J_m)$ worth of "mass" from J'. However, this case has the advantage of being able to select the most valuable jobs in J' to an extent greater than one by scheduling them fractionally across several machines. Viewed as a "residual" problem once $J_1 \cup \ldots \cup J_m$ are fixed, the solution here in terms of J' is computed by solving the LPs for $\Phi_{J'}(d - \mu(J_1)) \ldots \Phi_{J'}(d - \mu(J_m))$. We claim that the total objective value of these solutions must be at least as large as the objective value in the preceding case, for the LP for $\Psi_{J'}(d - \mu(J_1 \cup \ldots \cup J_m)/m)$. This follows from the fact that an optimal fractional solution to the LP for $\Psi_{J'}(d - \mu(J_1 \cup \ldots \cup J_m)/m)$ can be decomposed into feasible fractional solutions for the LPs for $\Phi_{J'}(d - \mu(J_1)) \ldots \Phi_{J'}(d - \mu(J_m))$ whose total objective values sum to $\Psi_{J'}(d - \mu(J_1 \cup \ldots \cup J_m)/m)$. □

Start Deadline Model. Suppose we take each set J_i and process it non-adaptively on machine i according to the WSEPT ordering.

Theorem 15. *Using WSEPT to process each set J_i gives a performance guarantee of 4 for the problem $P \mid p_j \sim stoch, d_j = 1 \mid \mathbf{E}[\sum w_j \overline{V}_j]$.*

Proof. According to our analysis in Lemma 5 (when we analyzed WSEPT in the single machine case), we obtain an expected value of at least $\Phi_{J_i}(1)/2$ for each machine i. Therefore, we receive a total expected value of at least

$$\frac{1}{2}\sum_{i=1}^{m}\Phi_{J_i}(1) = \frac{1}{2}\sum_{i=1}^{m}\Phi_{J_i\cup J'}(1) \geq \frac{1}{4}\sum_{i=1}^{m}\Phi_{J_i\cup J'}(2) \geq \frac{1}{4}\Psi(2) \geq \frac{1}{4}ADAPT.$$

The initial equality is due to the fact that $\mu(J_i) \geq 1$ for all $i \in [m]$, and also the fact that jobs in J' have smaller "value density" (w_j/μ_j) than jobs in J_i. □

Completion Deadline Model. In the completion deadline model, WSEPT by itself tends not to be a good policy. In this case, we use the randomized round-by-value policy (Section 3.4) independently on each machine i to schedule the jobs in J_i.

Theorem 16. *By using the round-by-value randomized non-adaptive policy to schedule each set J_i, we obtain a performance guarantee of $32/7 < 4.572$ for the problem $P \mid p_j \sim stoch, d_j = 1 \mid \mathbf{E}[\sum w_j\overline{U}_j]$*

Proof. According to Lemma 5, on each machine $i \in [m]$ the round-by-value policy obtains an expected value of at least at least $7\Phi_{J_i\cup J'}(2)/32$. Therefore, our total expected value is at least

$$\frac{7}{32}\sum_{i=1}^{m}\Phi_{J_i\cup J'}(2) \geq \frac{7}{32}\Psi(2) \geq \frac{7}{32}ADAPT.$$

Note that the round-by-value policy only considers the jobs in J_i for scheduling, and not those in J' (so we don't try to schedule the same job on two different machines). □

6.4 Pre-Assignment by Shmoys-Tardos Rounding

We now consider the unrelated parallel machine model, where the value and processing time of a job can depend on the machine to which it is assigned. Let w_{ij} and p_{ij} denote the value and (random) processing time of job j if processed on machine i, and let $w'_{ij} = w_{ij}\mathbf{Pr}[p_{ij} \leq 1]$ and $\mu_{ij} = \mathbf{E}[\min(p_{ij}, 1)]$. We can bound $ADAPT$ in the unrelated machine model using linear programs that generalize $\Psi(d)$ and $\Psi^*(d)$:

$\Psi(d)$	$=$	$\max \sum_{i=1}^{m}\sum_{j=1}^{n} w_{ij}x_{ij}$	$\Psi^*(d)$	$=$	$\max \sum_{i=1}^{m}\sum_{j=1}^{n} w'_{ij}x_{ij}$
$\forall j \in [n]$	:	$\sum_{i=1}^{m} x_{ij} \leq 1$	$\forall j \in [n]$	:	$\sum_{i=1}^{m} x_{ij} \leq 1$
$\forall i \in [m]$	:	$\sum_{j=1}^{n} x_{ij}\mu_{ij} \leq d$	$\forall i \in [m]$	:	$\sum_{j=1}^{n} x_{ij}\mu_{ij} \leq d$
$\forall (i,j)$	:	$0 \leq x_{ij} \leq 1$	$\forall (i,j)$	:	$0 \leq x_{ij} \leq 1$

Just as before, we have $ADAPT \leq \Psi(2)$ in the start deadline model and $ADAPT \leq \Psi^*(2) \leq \Psi(2)$ in the completion deadline model. This is argued, as usual (see for example the proof of Theorem 2) by interpreting x_{ij} as the probability that an optimal policy P attempts to schedule job j on machine i, and by showing that x is a feasible solution to the LPs above (with $d = 2$) whose value is at least the expected value obtained by P. Note also that by Lemma 3, $\Psi(\cdot)$ and $\Psi^*(\cdot)$ are concave.

6.4.1 Unrelated Parallel Machines, Completion Deadline Model

The linear programs for both $\Psi(d)$ and $\Psi^*(d)$ solve a problem whose minimization variant is known as the fractional generalized assignment problem. In terms of assigning n jobs with deterministic processing times $\mu_1 \ldots \mu_n$ to m unrelated parallel machines, the variable x_{ij} indicates the fraction of job j to assign to machine i. The generalized assignment problem is typically phrased in terms of minimizing the cost of the assignment, whereas in our case we are maximizing its value.

We now employ a rounding technique due to Shmoys and Tardos [50]. Restricting our focus to the completion deadline model, suppose we optimally solve the LP for $\Psi^*(1)$. The Shmoys-Tardos procedure takes this fractional solution and rounds it to an integral solution of equal or higher total value. The main benefit of the procedure is that it ensures that for each machine at most a single job will "overflow" the deadline $d = 1$ (treating processing times as being deterministic and set to μ_j). Since $\mu_j \leq 1$ for each job j, this gives us an integer solution of value at least $\Psi^*(1)$ that is feasible for $\Psi^*(2)$. From this we obtain disjoint sets of jobs $J_1 \ldots J_m$ to process on each machine.

Theorem 17. *By using the round-by-value randomized non-adaptive policy to schedule each set J_i, we obtain a performance guarantee of $64/7 < 9.143$ for the problem $R \mid p_j \sim stoch, d_j = 1 \mid \mathbf{E}[\sum w_j \overline{U}_j]$.*

Proof. We use the same argument as in Theorem 16. Starting with the upper bound $ADAPT \leq \Psi^*(2) \leq 2\Psi^*(1)$, we can bound $\Psi^*(1)$ by a sum of single-machine solutions,

$$\Psi^*(1) \leq \sum_{i=1}^{m} \Phi^{(i)}_{J_i}(2),$$

where the notation $\Phi^{(i)}_J$ means the $\Phi^*_J(\cdot)$ function using values $w'_{i,\bullet}$ and mean truncated sizes $\mu_{i,\bullet}$ (i.e., the parameters we get when we view machine i as an isolated single-machine problem by itself). We conclude the proof by using Theorem 5 to argue that on each machine i, the round-by-value policy obtains expected value at least $7\Phi^{(i)}_{J_i}(2)/32$. □

6.5 Pre-Assignment by Randomized Rounding

Our final method for pre-assigning jobs to machines involves the use of randomized rounding. Technically, we have already used randomized rounding in a sense, because the preceding

technique of Shmoys and Tardos is a type of "dependent" randomized rounding (it, as well as the following technique, can also be derandomized). In this section, we discuss the use of "independent" randomized rounding, where we randomly assign each job independently from all other jobs.

6.5.1 Unrelated Parallel Machines, Start Deadline Model

Consider the problem $R \mid p_j \sim stoch, d_j = 1 \mid \mathbf{E}[\sum w_j \overline{V}_j]$. As usual, we solve $\Psi(2)$ to obtain a fractional solution x whose value is an upper bound on $ADAPT$. For our randomized rounding step, we assign job j to machine i with probability $x_{ij}/2$. For each job j, we ensure that it ends up assigned to at most one machine by laying out disjoint segments of lengths $x_{ij}/2$ on a unit interval and randomly choosing a single point on the interval. If you recall the "coin flip" randomized rounding procedure we used in Chapter 5, the current procedure can be interpreted in a similar fashion: we assign job j to machine i with probability x_{ij}, but when it comes time to schedule job j we flip a fair coin to decide whether to schedule or skip over it. After randomized rounding, we are left with disjoint sets of jobs $J_1 \ldots J_m$ to schedule on each machine.

Theorem 18. *Suppose we processes each set J_i in the WSEPT ordering on machine i. This yields a performance guarantee of 4 for the problem $R \mid p_j \sim stoch, d_j = 1 \mid \mathbf{E}[\sum w_j \overline{V}_j]$.*

Proof. Let us index the jobs according to the WSEPT ordering. We then define

$$M_{ij} = \sum_{k=1}^{j} x_{ik}\mu_{ik}.$$

Just as in the case of the "coin flip" randomized rounding from the preceding chapter, we claim by Markov's inequality that the probability job j *would* be successfully scheduled on machine i (not considering whether or not j is actually chosen to be scheduled on machine i) is at least $1 - M_{i,j-1}/2$. Note that this quantity is in $[0,1]$ since x is feasible for the LP for $\Psi(2)$, and hence $M_{ij} \leq 2$ for all $(i,j) \in [m] \times [n]$. Continuing in the style of the randomized rounding argument from the preceding chapter, we define two random variables: W_{ij}, which takes value 1 with probability $x_{ij}/2$ and zero otherwise, and X_{ij} taking the value 1 if it would be possible to schedule job j on machine i (so $\mathbf{E}[X_{ij}] \geq 1 - M_{i,j-1}/2$), or zero otherwise. Job j is scheduled on machine i only of both of these indicators turn out positive: $X_{ij} = 1$ indicates that there is still enough time prior to the deadline to schedule job j on machine i, and $W_{ij} = 1$ indicates that our randomized rounding assigns j to J_i. Since W_{ij} and X_{ij} are independent, the total expected value we obtain is therefore

$$\begin{aligned}
\mathbf{E}\left[\sum_{i=1}^{m}\sum_{j=1}^{n} w_{ij}W_{ij}X_{ij}\right] &= \sum_{i=1}^{m}\sum_{j=1}^{n} w_{ij}\mathbf{E}[W_{ij}]\mathbf{E}[X_{ij}] \\
&\geq \sum_{i=1}^{m}\sum_{j=1}^{n} \frac{w_{ij}x_{ij}}{2}\left(1 - \frac{M_{i,j-1}}{2}\right)
\end{aligned}$$

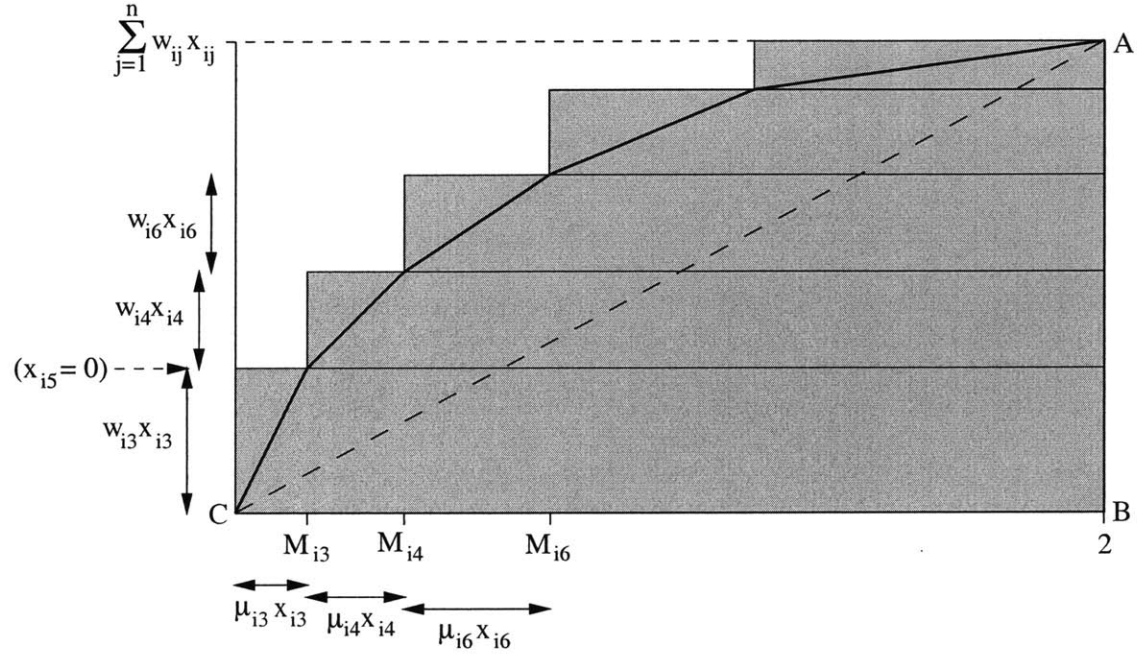

Figure 6-3: Getting rid of the $2 - M_{i,j-1}$ term.

$$\begin{aligned}
&\geq \frac{1}{4} \sum_{i=1}^{m} \sum_{j=1}^{n} w_{ij} x_{ij} (2 - M_{i,j-1}) \\
&\geq \frac{1}{4} \sum_{i=1}^{m} \sum_{j=1}^{n} w_{ij} x_{ij} \\
&= \frac{1}{4} \Psi(2) \\
&\geq \frac{1}{4} ADAPT.
\end{aligned}$$

The only mysterious step above involves the vanishing $(2 - M_{i,j-1})$ term. This follows from the same geometric argument as used in the proof of Lemma 5. Consider a particular machine $i \in [m]$. In Figure 6-3, the area of the shaded rectangles is $\sum_{j=1}^{n} w_{ij} x_{ij} (2 - M_{i,j-1})$, and this is lower bounded by the area of triangle ABC, which is $\sum_{j=1}^{n} w_{ij} x_{ij}$. □

6.5.2 Individual Deadlines, Start Deadline Model

For the problem $R \mid p_j \sim stoch \mid \mathbf{E}[\sum w_j \overline{V}_j]$ where jobs have individual deadlines, randomized rounding gives a performance guarantee of 8 following exactly the same argument as in the single-machine case. Let us first generalize the LP for $\Psi(\cdot)$ for this case:

$$
\begin{array}{rcl}
\Psi(\lambda) & = & \max \sum_{i=1}^{m} \sum_{j=1}^{n} w_{ij} x_{ij} \\
\forall j \in [n] & : & \sum_{i=1}^{m} x_{ij} \leq 1 \\
\forall (i,j) \in [m] \times [n] & : & \sum_{k:d_k \leq d_j} x_{ik} \mathbf{E}[\min(p_{ik}, d_j)] \leq \lambda d_j \\
\forall (i,j) \in [m] \times [n] & : & 0 \leq x_{ij} \leq 1
\end{array}
$$

We still have $ADAPT \leq \Psi(2)$ since the solution x derived from an optimal adaptive policy P is feasible for $\Psi(2)$ and its objective value is the expected value obtained by P. Feasibility follows from the fact that each packing constraint can be considered as its own individual single-machine instance, and by applying Lemma 1.

Suppose we solve the LP for $\Psi(1)$ (rather than $\Psi(2)$ as before) and then apply randomized rounding as before, assigning job j to machine i with probability $x_{ij}/2$. This gives us disjoint sets of jobs $J_1 \ldots J_m$ to process on each machine. However, whereas in the preceding (common deadline) case we processed each set using WSEPT, here we process jobs in order of their deadlines, just as in Chapter 5.

Theorem 19. *Randomized rounding followed by processing of jobs in order of their deadlines gives a performance guarantee of 8 for the problem* $R \mid p_j \sim stoch \mid \mathbf{E}[\sum w_j \overline{V}_j]$.

Proof. We proceed as before, by indexing jobs in order of their deadlines and by defining

$$M_{ij} = \sum_{k=1}^{j} x_{ik} \mu_{ik}.$$

Since we started with a feasible solution x to $\Psi(1)$, we know that $M_{ij} \leq d_j$. Now we define random variables W_{ij} and X_{ij} as before, where W_{ij} takes the value 1 with probability $x_{ij}/2$ and zero otherwise, and X_{ij} takes the value 1 if we *would* be able to schedule job j on machine i based on its deadline (irrespective of whether or not we randomly decide to schedule it). According to the proof of Lemma 10, $\mathbf{E}[X_{ij}] \geq 1 - M_{i,j-1}/(2d_j) \geq 1/2$. Since W_{ij} and X_{ij} are independent, the expected value obtained by our randomized rounding policy is

$$
\begin{array}{rcl}
\mathbf{E}\left[\sum_{i=1}^{m} \sum_{j=1}^{n} w_{ij} W_{ij} X_{ij}\right] & = & \sum_{i=1}^{m} \sum_{j=1}^{n} w_{ij} \mathbf{E}[W_{ij}] \mathbf{E}[X_{ij}] \\
& \geq & \sum_{i=1}^{m} \sum_{j=1}^{n} \frac{w_{ij} x_{ij}}{2} \left(1 - \frac{M_{i,j-1}}{2d}\right) \\
& \geq & \frac{1}{4} \sum_{i=1}^{m} \sum_{j=1}^{n} w_{ij} x_{ij} \\
& = & \frac{1}{4} \Psi(1)
\end{array}
$$

$$\begin{aligned} &\geq \frac{1}{8}\Psi(2) \\ &\geq \frac{1}{8}ADAPT, \end{aligned}$$

and this completes the proof. □

7. Discrete Distributions and Dynamic Programming

This chapter adopts a slightly different outlook from most of the previous chapters, focusing on problems with discrete, integer-valued processing time distributions. In such a discrete-time setting, we can use dynamic programming (DP) to compute optimal and approximate solutions to a variety of problems related to our fundamental stochastic scheduling problems.

The deterministic counterparts of many of our fundamental stochastic scheduling problems are variants of the 0/1 knapsack problem. Although integer knapsack problems are NP-hard, they are commonly solved in practice by assuming item sizes are small integers and using DP. We begin this chapter by showing how this approach also applies to $1 \mid p_j \sim stoch, d_j = d \mid \mathbf{E}[\sum w_j \overline{V}_j]$ and $1 \mid p_j \sim stoch, d_j = d \mid \mathbf{E}[\sum w_j \overline{U}_j]$ (the two problems on which we focus our attention for most of this chapter) in the case where processing times are small integers. However, there is an important fundamental limitation inherent in the use of DP applied to these stochastic scheduling problems: it only seems capable of computing an optimal *ordered* adaptive policy rather than general optimal adaptive policy. Recall from Section 1.1.9 that an ordered adaptive policy considers jobs in the order they appear in the input and for each one, either attempts to schedule it or irrevocably skips over it.

How good is an optimal ordered adaptive policy compared to a general optimal adaptive policy? If we start with the ordering of jobs suggested by any of the non-adaptive policies developed in earlier chapters, then an optimal ordered adaptive policy using this ordering can only potentially deliver more expected value, never less. Therefore, we might view the DP algorithms in this chapter as a heuristic means of squeezing more performance out of our existing non-adaptive policies. If we start with an arbitrary ordering of jobs, then an optimal adaptive ordered policy will give us at least as much expected value as we can obtain in the *fixed set* model. Recall (again from Section 1.1.9) that the fixed set model asks us to compute a single set of jobs, for which we only receive the value of the entire set if all the jobs in the set successfully start/complete prior to our deadline. We show in this chapter how to compute a 9.5-approximate (completion deadline model) or 8-approximate (start deadline model) solution in the fixed set model for the stochastic knapsack problem, which therefore also implies that an optimal adaptive ordered policy using *any* ordering must deliver a solution that falls within these performance guarantees.

The DP formulations for our stochastic scheduling problems are somewhat interesting in their own right. Like many other stochastic DP formulations, we need to compute discrete convolutions to solve each successive subproblem, so by using the Fast Fourier Transform (FFT) we can improve the running time of these DP algorithms. Furthermore, for problem variants in which we are allowed to schedule multiple "copies" of each job, we show how the traditional FFT no longer helps, but that a recent technique from signal processing called *zero delay convolution* can be used to improve our running times (and also the running times of many other stochastic problems with DP formulations — see [13]).

7.1 Dynamic Programming and Ordered Models

Consider the deterministic problems $1 \mid d_j = d \mid \mathbf{E}[\sum w_j \overline{V}_j]$ and $1 \mid d_j = d \mid \mathbf{E}[\sum w_j \overline{U}_j]$ (i.e., the 0/1 knapsack problem in the start deadline and completion deadline models) in which all processing times p_j are integers in the range $0 \ldots d$. It is well-known that these problem can be optimally solved in $O(nd)$ time using the following simple DP formulation: let $V[j,t]$ denote the optimal value one can obtain by scheduling only a subset of jobs $1 \ldots j$ with t units of time remaining until the deadline. Then for $t > 0$,

$$V[j,t] = \max \begin{cases} V[j-1,t] \\ V[j-1,t-p_j] + w_j \end{cases} \tag{7.1}$$

where $V[0,t] = 0$ and $V[j,t<0] = 0$ (in the start deadline model) or $V[j,t<0] = -\infty$ (in the completion deadline model) are our initial conditions. The optimal overall solution is given by $V[n,d]$. In the stochastic case, we generalize this formulation in the following natural way: let $V[j,t]$ denote the optimal expected value we can obtain by scheduling a subset of jobs $1 \ldots j$ (in reverse order, allowing adaptivity) with t units of time remaining until the deadline. Then for $t > 0$,

$$V[j,t] = \max \begin{cases} V[j-1,t] \\ V'[j,t] + w_j \mathbf{Pr}[p_j \leq t] \quad (*) \end{cases} \tag{7.2}$$

where

$$V'[j,t] = \sum_{\tau=0}^{t} V[j-1,t-\tau]\mathbf{Pr}[p_j = \tau]. \tag{7.3}$$

In the start deadline model, we remove the $\mathbf{Pr}[p_j \leq t]$ term from (*). This DP formulation computes an optimal ordered (actually, reverse-ordered) adaptive policy, since the recursive formula above computes the best solution for jobs $1 \ldots j$ by deciding first whether or not to schedule job j, and then by completing the scheduling using jobs $j-1 \ldots 1$. The entire ordered adaptive solution is encoded succinctly in terms of the "traceback" paths through the DP subproblems. That is, by looking at how $V[n,d]$ was computed (i.e., which term in (7.2) was selected as the maximum), we can determine whether or not we should start out by scheduling or skipping job n. After this decision (suppose we now have t units of time

left), we look at the way $V[n-1,t]$ was computed to determine whether to schedule job $n-1$, and so on.

For the deterministic case, the ordering of jobs over time in our solution is not of any consequence. However, in the stochastic case, our solution may be sensitive to the ordering of items in our input, and we may not end up with an overall optimal adaptive policy. It does not seem that there exists a DP formulation for computing a general (non-ordered) optimal adaptive policy whose state space is polynomial in n and d. The difficulty is that since jobs might need to be scheduled in almost any order, we need to remember in our state space the subset of jobs that have already been scheduled, and this requires exponential space.

As a final note, the DP formulations above can be generalized to accommodate individual deadlines for jobs, in both the start deadline and completion deadline models. For the remainder of this chapter, however, we assume common deadlines.

7.1.1 Efficient Computation Using the FFT

Using the deterministic formulation (7.1) in the previous section, we can solve the 0/1 knapsack problem in $\Theta(nd)$ time. One should note that this is only a pseudo-polynomial running time due to the polynomial dependence on d (although it is polynomial in the size of the input if we assume that each processing time p_j is represented by a length-$(d+1)$ vector $\mathbf{Pr}[p_j = 0] \ldots \mathbf{Pr}[p_j = d]$). A straightforward DP implementation based on the stochastic formulation (7.2)-(7.3) runs somewhat slower, in $\Theta(nd^2)$ time. In practice, where values of d in the hundreds or thousands are common, the d^2 term can be a significant liability. Fortunately, we can improve the running time to $\Theta(nd \log d)$ using the Fast Fourier Transform.

Let us focus on evaluating (7.3) in $\Theta(nd \log d)$ total time, since (7.2) only consumes $\Theta(nd)$ time in total. Fixing a particular value of j, one can regard (7.3) for all values of t as the computation of the discrete convolution between two sequences: $V[j-1, 0 \ldots d]$ and $\mathbf{Pr}[p_j = 0] \ldots \mathbf{Pr}[p_j = d]$. The FFT can convolve two length-d sequences in $\Theta(d \log d)$ time, giving us a total runtime $\Theta(nd \log d)$ to apply the FFT for each value $j = 1 \ldots n$. Note that the "row by row" computation order of our table of DP subproblems is essential for this approach to work.

7.1.2 Converting to a Discrete Distribution

If we do not have integer-valued processing times distributions in our instance to start with, we can create such distributions by paying only a small penalty in terms of feasibility. More precisely, for any $\varepsilon > 0$, we can discretize job processing times so they are positive integers in the range $1 \ldots D$ where $D = \frac{2n}{\varepsilon} = O(n)$. After discretization, we can compute an optimal ordered policy with DP (now in *strongly* polynomial time) that is at least as good as an optimal ordered policy for the original processing times, provided that we inflate the common deadline of our jobs by a $(1+\varepsilon)$ factor. If we apply such a discretization scheme

on top of one of our non-adaptive α-approximation algorithms, we therefore end up with an $(\alpha, 1+\varepsilon)$-pseudo-approximation algorithm. Note that this approach only applies when our jobs share a common deadline.

The transformation we use is as follows. First, scale all processing times $p_1 \ldots p_n$ by a factor of $\frac{2n}{\varepsilon d}$ so they have the right magnitude. We then discretize each p_j into a new integer-valued distribution p'_j as follows:

- $\mathbf{Pr}[p'_j = \infty] := \mathbf{Pr}[p_j > D]$.
- $\mathbf{Pr}[p'_j = D] := \mathbf{Pr}[p_j = D]$.
- For $1 < t < D$, $\mathbf{Pr}[p'_j = t] = \mathbf{Pr}[t \leq p_j < t+1]$.
- $\mathbf{Pr}[p'_j = 1] := \mathbf{Pr}[p_j < 2]$.
- $\mathbf{Pr}[p'_j = 0] := 0$.

We are essentially "rounding down" the mass in the probability distribution of p_j to the next lowest integer, except for the case where $p_j < 1$, in which we round up since we want to end up with strictly positive processing times (we can skip this case if processing times of zero are acceptable, as they are for the DP formulations in the preceding section).

A solution policy is now computed using the p'_j distributions. Since a job with *actual* processing time less than 1 (according to p_j) appears to our algorithm as a job of processing time 1, we choose to run our algorithm up to time $D+n$. At this point, we can be assured that the actual amount of processing time spent is at least D. Therefore, any algorithm applied to the discretized instance will have a deadline constraint that is no tighter than that of the original instance, so we can still hope to achieve an expected value of $ADAPT$. Furthermore, the actual processing time of a job may turn out to be up to one unit larger than it appears to our algorithm. Hence, when our algorithm reaches its deadline at time $D+n$, it may have actually spent up to $D+2n$ units of time. In terms of the original deadline, we are spending at most $(D+2n)\frac{\varepsilon d}{2n} = (\frac{2n}{\varepsilon}+2n)\frac{\varepsilon d}{2n} = (1+\varepsilon)d$ units of time.

As a remark, all of the "true' approximation algorithms (producing a feasible solution) for packing problems like the knapsack and multi-knapsack problems perform discretization on item values rather than item sizes. There is a good reason for this: by rounding sizes up, we might render the sole optimal solution infeasible if it just barely satisfies capacity constraints. By rounding sizes down, we run into trouble with the smallest size class that is rounded to size zero, since our algorithm now thinks it can pack an infinite quantity of such items (which may not be feasible for the original instance). It seems that any rounding in the space of sizes dooms us to a pseudo-approximation algorithm, but in our case this seems to be the best route since the simplification gained in terms of probability distribution structure is worth the small amount of potential infeasibility we incur.

7.2 The Fixed Set Model

In this section we describe an 8-approximation algorithm for $1 \mid d_j = 1 \mid \mathbf{E}[\sum w_j \overline{V}_j]$ and a 9.5-approximation algorithm for $1 \mid d_j = 1 \mid \mathbf{E}[\sum w_j \overline{U}_j]$ in the fixed set model. As a consequence, we obtain a constant-factor bound on the approximate performance of any optimal adaptive ordered policy, regardless of the initial ordering of jobs we use. Note that we assume for simplicity that $d = 1$ throughout this section (this is without loss of generality).

Theorem 20. *Consider the single-machine environment where all jobs share a common deadline. For* any *ordering of jobs, an optimal ordered adaptive policy must receive at least an expected value of $ADAPT/8$ (in the start deadline model) or $ADAPT/9.5$ (in the completion deadline model).*

The theorem follows from simple fact that an adaptive ordered policy can elect to schedule only the jobs belonging to the best possible fixed set. Our objective value in this case will be at least as good as in the fixed set model, and it can potentially be much better since in the ordered adaptive model we get "partial credit" even if a subset of these jobs are successfully scheduled.

Lemma 24. *In the start deadline model, index our jobs according to the WSEPT ordering ($\frac{w_1}{\mu_1} \geq \frac{w_2}{\mu_2} \geq \ldots \geq \frac{w_n}{\mu_n}$), and let J be the smallest prefix of this ordering for which $\mu(J) \geq 1/2$. Then the set J is an 8-approximate solution in the fixed set model.*

Proof. Let j be the "final" job in J (according to the WSEPT ordering). Since $\mu(J \backslash \{j\}) \leq 1/2$, Lemma 4 tells us that

$$\mathbf{Pr}\left[\sum_{i \in J \backslash \{j\}} p_i \leq 1\right] \geq 1/2.$$

In the start deadline model, all the jobs in J will therefore be successfully scheduled with probability at least $1/2$. Moreover, for the function $\Phi(\cdot)$ as defined in Chapter 3, we have $w(J) = \Phi(\mu(J)) \geq \Phi(1/2) \geq \Phi(1)/2$ and by Theorem 1 we have $ADAPT \leq \Phi(2)$, so $w(J) \geq ADAPT/4$. Our total expected value ($w(J)$ times the probability that all jobs in J are successfully scheduled) is therefore at least $ADAPT/8$. □

In the completion deadline model, our 9.5-approximation algorithm is only slightly more complicated. We take the better of two solutions, one of which the best single job, delivering an expected value of

$$SINGLE = \max_j w'_j,$$

and the other solution is the set of small ($\mu_j \leq \varepsilon$) jobs J maximizing the objective

$$MULTI = \max_{J \subseteq S} w'(J)(1 - \mu(J)).$$

Note that by Lemma 4, $MULTI$ gives a lower bound on the expected value we can hope to receive by using the set J. Also note that while it seems hard to compute $MULTI$ exactly, we can find a set J that yields an expected value of at least $(1-\varepsilon)MULTI$ by using the PTAS for knapsack. Suppose for simplicity that $\varepsilon \in [0, 1/2]$ (although any range $[0, c]$ with $c < 1$ suffices) define the cutoff between large and small jobs, so $\mu_j \leq \varepsilon \leq 1/2$ for all $j \in S$. Let $V = \max_{j \in S} w'_j(1-\mu_j) \geq \max_{j \in S} w'_j/2$, so we know that $MULTI \geq V$ and also $MULTI \leq 2nV$. Now round all job values down to the nearest multiple of $\varepsilon V/n$. The total value lost for any set is at most $\varepsilon V \leq \varepsilon MULTI$, and our values now effectively lie in the range $0 \ldots n/\varepsilon$. We then use dynamic programming, in $O(n^3/\varepsilon)$ total time, to compute an optimal set (whose value is therefore at least $(1-\varepsilon)$ times that of a truly optimal set prior to rounding). The DP subproblems are of the form $A[j, v]$, specifying the maximum possible value of $(1-\mu(J))$ (equivalently, the minimum value of $\mu(J)$) over all sets $J \subseteq \{1, 2, \ldots j\} \subseteq S$ with value precisely equal to v.

Lemma 25. *Let $B = \max(SINGLE, MULTI)$. Then for $\varepsilon = \sqrt{5} - 2$, we have $B \geq ADAPT/9.5$, where $ADAPT$ denotes the expected value obtained by an optimal adaptive policy for $1 \mid d_j = 1 \mid \mathbf{E}[\sum w_j \overline{U}_j]$.*

Proof. Fix an optimal adaptive policy P for $1 \mid p_j \sim stoch, d_j = 1 \mid \mathbf{E}[\sum w_j \overline{U}_j]$. Consider jobs as being small or large based on some threshold ε, and let J_S and J_L denote the (random) sets of small and large jobs that P attempts to schedule. Lemma 9 tells us that $\mathbf{E}[w'(J_L)] \leq \mathbf{E}[\mu(J_L)]\ SINGLE/\varepsilon$. We now argue that for any set T of small jobs,

$$w'(T) \leq \left(1 + \frac{4\mu(T)}{1-\varepsilon^2}\right) MULTI.$$

If $\mu(T) > (1+\varepsilon)/2$, then we use induction on $|T|$: let T' be a proper subset of T such that $\frac{1-\varepsilon}{2} \leq \mu(T') \leq \frac{1+\varepsilon}{2}$. Then by induction,

$$w'(T \backslash T') \leq \left(1 + \frac{4(\mu(T) - \mu(T'))}{1-\varepsilon^2}\right) MULTI.$$

Furthermore, since $w'(T')(1-\mu(T')) \leq MULTI$, we have

$$w'(T')\frac{1-\varepsilon^2}{4} \leq w'(T')\mu(T')(1-\mu(T')) \leq \mu(T')MULTI,$$

so

$$w'(T) = w'(T') + w'(T \backslash T') \leq \left(1 + \frac{4\mu(T)}{1-\varepsilon^2}\right) MULTI.$$

On the other hand, if $\mu(T) \leq (1+\varepsilon)/2$, then

$$w'(T) \leq \frac{1}{1-\mu(T)} MULTI \leq \left(1 + \frac{4\mu(T)}{1-\varepsilon^2}\right) MULTI.$$

To conclude our proof, we have

$$\begin{aligned} ADAPT &\leq \mathbf{E}[w'(J_S)] + \mathbf{E}[w'(J_L)] \\ &\leq \left(1 + \frac{4\mathbf{E}[\mu(J_S)]}{1-\varepsilon^2}\right) MULTI + \frac{\mathbf{E}[\mu(J_L)]}{\varepsilon} SINGLE \\ &\leq \left(1 + \frac{4\mathbf{E}[\mu(J_S)]}{1-\varepsilon^2} + \frac{\mathbf{E}[\mu(J_L)]}{\varepsilon}\right) B \\ &\leq \left(1 + \max\left(\frac{4}{1-\varepsilon^2}, \frac{1}{\varepsilon}\right) \mathbf{E}[\mu(J_S \cup J_L)]\right) B \\ &\leq \left(1 + 2\max\left(\frac{4}{1-\varepsilon^2}, \frac{1}{\varepsilon}\right)\right) B \end{aligned}$$

and for $\varepsilon = \sqrt{5} - 2 \approx 0.236$ this gives us $ADAPT \leq 9.48B$. Recall that we do not know how to compute $MULTI$ exactly in polynomial time, although we can approximate this quantity to within an arbitrary constant factor. Taking this factor to be small enough, we obtain a 9.5-approximation algorithm that runs in polynomial time. □

In order to improve the performance guarantees for our fixed set approximation algorithms, one approach to consider in the future might be to compare against a less aggressive adversary than an optimal adaptive policy. For example, if we compare against the best possible expected value for a fixed set, we might be able to improve our performance guarantees substantially.

7.3 High-Multiplicity Scheduling Problems

Another common type of integer knapsack problem arises when we think in terms of "job types" (multiple copies of which can be included in a solution) rather than single jobs. For example, we can assign a multiplicity c_j to each job j, which specifies the number of copies of job j available to schedule. In the extreme case, we can allow unlimited copies of each type of job. This case is particularly nice from a DP standpoint, since it not only simplifies our state space, but also allows us to compute an optimal (general, not ordered) adaptive solution. Let us now abandon the $d = 1$ assumption from the preceding section and return to the case where d and the p_j's are integer-valued. For a deterministic problem, our DP subproblems are now as follows: let $V[t]$ denote the value of an optimal schedule if our deadline is t units ahead. If we include a "dummy" job with $p_j = 1$ and $w_j = 0$ (a simple way to avoid treating idle time as a special case), then

$$V[t] = \max_j V[t - p_j] + w_j \tag{7.4}$$

where $V[j = 0] = 0$ and $V[j < 0] = 0$ (start deadline model) or $V[j < 0] = -\infty$ (completion deadline model) are base cases. In the stochastic case, $V[t]$ is an expected value, and for

$t > 0$ we have

$$V[t] = \max_j \left(V_j[t] + w_j \mathbf{Pr}[p_j \leq t]\right) \tag{7.5}$$

where

$$V_j[t] = \sum_{\tau=1}^{t} V[t-\tau]\mathbf{Pr}[p_j = \tau]. \tag{7.6}$$

Again, in the start deadline model we need to remove the $\mathbf{Pr}[p_j \leq t]$ term from (7.5). Note also that as opposed to the 0/1 case, we assume here for simplicity that processing times are strictly positive. Furthermore, note that our discretization techniques from the preceding section no longer apply in this case, since those techniques assume that an optimal solution contains at most n jobs — therefore, for any "high multiplicity" problem instance we assume that our processing time distributions start out being integer-valued.

7.3.1 Zero-Delay Convolution

A straightforward DP algorithm based on (7.5)-(7.6) runs in $\Theta(nd^2)$ time. In the 0/1 case, we managed to improve this exact running time to $\Theta(nd \log d)$ using the FFT. Here, however, we find that this approach no longer seems to work. The FFT is designed to convolve two *complete* length-d sequences in $\Theta(d \log d)$ time, but the DP approach in (7.5)-(7.6) requires the computation of the convolution of two sequences where one of the sequences is being generated in an on-line fashion and then "fed back" into the convolution. It turns out, however, that we can still speed up this sort of DP (so the running time drops to $\Theta(nd \log^2 d)$) using a more sophisticated technique from signal processing known *zero delay convolution*, developed by Gardner in 1995 [21]. In fact, zero-delay convolution can be seen as a general-purpose technique for speeding up certain types of stochastic dynamic programming algorithms [13].

We illustrate the technique of zero-delay convolution applied to DP using a simple example. Let $h[1 \ldots n]$ be the probability distribution of an integer-valued random variable X, so $h[i] = \mathbf{Pr}[X = i]$. What is the expected number of independent samples of X one can draw until their sum reaches or exceeds some threshold $T \geq n$? We can answer this question using a simple dynamic program. Letting $A[j]$ be the expected number of independent samples required to reach a sum of j, we have

$$A[j] = 1 + \sum_{i=1}^{n} A[j-i]h[i], \tag{7.7}$$

where $A[j \leq 0] = 0$ as a base case. Straightforward computation of $A[1 \ldots T]$ by direct application of (7.7) requires $\Theta(nT)$ time. To improve this, we think of (7.7) as a special type of convolution between two sequences A and h. In a standard convolution problem we are given as input two pre-specified sequences to convolve. However, in this case the sequence A is actually generated in an on-line fashion, with each successive element of A

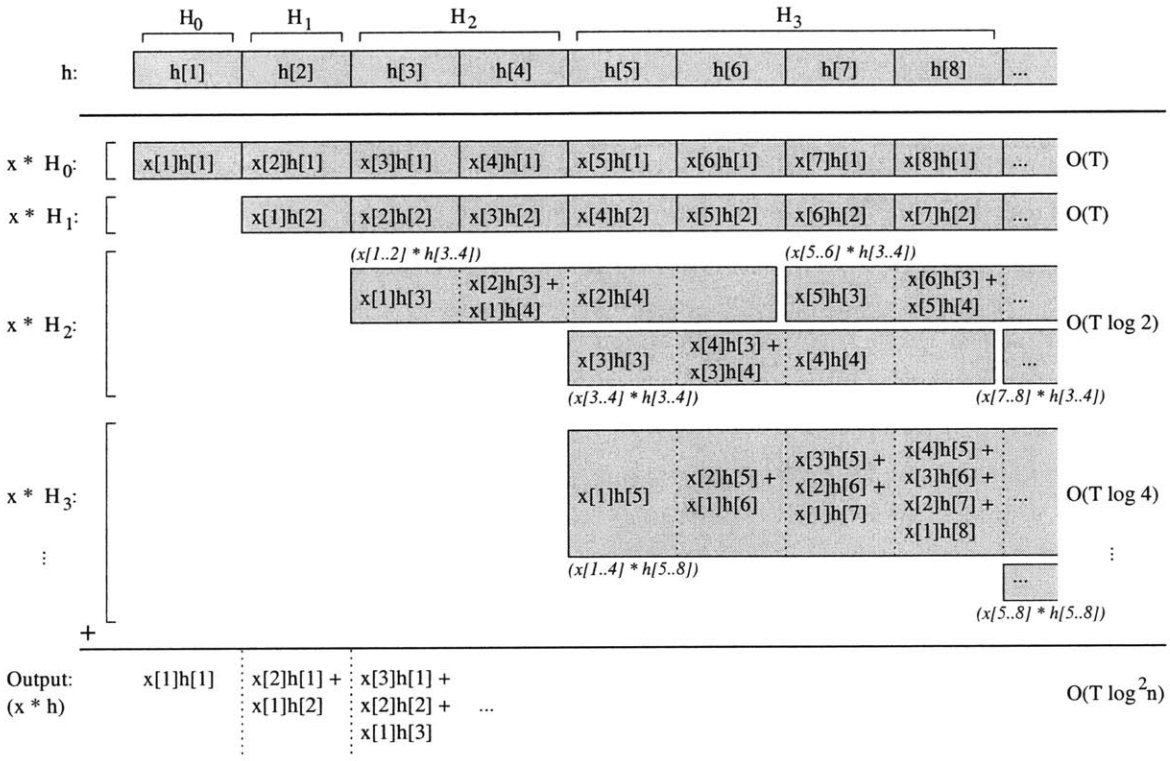

Figure 7-1: Illustration of the zero-delay convolution algorithm. We divide h into blocks $H_0, H_1, H_2, \ldots$ of exponentially increasing size (except for the first two, both of size 1). Each of these blocks is then convolved, in parallel, against x, using buffering and block convolution. The output, $x * h$ (we denote by $*$ the convolution operation), is obtained by summing these partial results. Running times are specified in the rightmost column, summing to $\Theta(T \log^2 n)$. Applied to our example of computing $A[1 \ldots T]$, we would take each output element, increment it, and feed it back in as the next input element.

depending on the results of the convolution so far. From a signal processing standpoint, we can picture this as a discrete-time system with impulse response h (so any signal fed through the system is convolved with h) where each output element is immediately incremented and fed back as the next element of the input signal.

A fundamental computation performed by discrete-time signal processing devices is the convolution of a long input sequence $x[1 \ldots T]$ with the impulse response $h[1 \ldots n]$ of the system. Since the Fast Fourier Transform (FFT) can convolve two length-n signals in $\Theta(n \log n)$ time, the usual approach is to buffer x into T/n length-n blocks and to convolve each of these in sequence against h using the FFT. This requires only $\Theta(\log n)$ time per output element, but it has the unpleasant side effect of introducing some input/output delay due to the buffering of x; that is, we must wait for n elements of x to arrive at the system

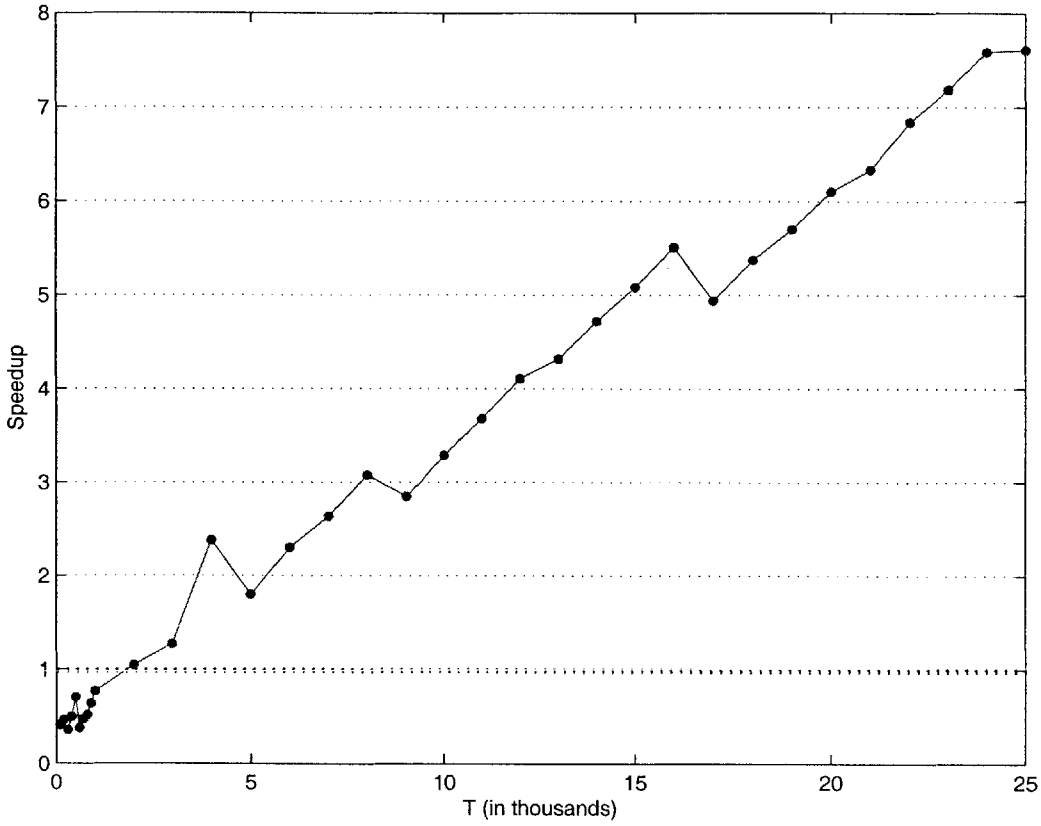

Figure 7-2: Speedup obtained by using zero-delay convolution. The small drops in performance for zero-delay convolution occur right after we cross a power of 2 in T, since this causes the zero-delay algorithm to perform quite a few more FFTs on larger subarrays.

before we can produce any output. Input/output delay is undesirable for obvious reasons in many signal processing applications, and it is absolutely unacceptable for our problem since we cannot advance the input signal until the complete result of the convolution thus far is determined.

Zero-delay convolution eliminates input/output delay at the expense of only a small running time penalty, producing each output element in $\Theta(\log^2 n)$ amortized time (this is incorrectly analyzed as $O(\log n)$ time in Gardner's paper [21]). Assume for simplicity and without loss of generality that n is a power of 2, and break the impulse response h into blocks of exponentially increasing size (except the first two, which are both of size 1), as shown in Figure 7-1. We then launch separate convolutions that move forward in parallel between each block and the entire input sequence. By adding together the results of these sub-convolutions, we obtain the final convolution of x with h. We convolve each block of h against x using the standard buffering approach: for a block of size B we buffer x into T/B

blocks of size B and convolve them each in sequence using the FFT, which requires a total of $\Theta(\frac{T}{B} \times B \log B) = \Theta(T \log B)$ time. Note that the block decomposition of h is designed so that buffering does not contribute to input/output delay. For example, in Figure 7-1 when we convolve x with H_3 we initially buffer the values $x[1 \ldots 4]$, but the result of convolving these buffered elements against H_3 is not used in computing elements $1 \ldots 4$ of the output. The total running time spent convolving x with all of the blocks of h is

$$\sum_{i=1}^{\log n} \Theta(T \log \frac{n}{2^i}) = \Theta(T \log^2 n),$$

which amortizes to $\Theta(\log^2 n)$ time per output element. Applying this technique to our original problem of computing of $A[1 \ldots T]$, we obtain an $\Theta(T \log^2 n)$ algorithm.

In order to develop a sense of how well the zero-delay convolution algorithm performs in practice, Figure 7-2 shows the results of a simple computational experiment. Zero-delay convolution was implemented in C using the well-known FFTW3.0.1 Fast Fourier Transform library of Frigo and Johnson, and compared against the "naïve" convolution approach for the sample formulation (7.7) above, setting $n = T$ (so the theoretical running times are $\Theta(T \log^2 T)$ for zero-delay convolution and $\Theta(T^2)$ for the naïve approach). The zero-delay approach seems to become superior for $T > 2000$ and improve from there. For problems that are finely-discretized in the time dimension, a value of T in the mid thousands is entirely conceivable in practice.

Let us now consider the use of zero-delay convolution to solve our original DP problem based on (7.5)-(7.6). Here, we use a separate zero-delay convolution process (running in parallel) for each job $j = 1 \ldots n$ to compute the sequence $V_j[\cdot]$ according to (7.6), by convolving $V[t]$ against the sequence $\mathbf{Pr}[p_j = 1] \ldots \mathbf{Pr}[p_j = d]$ as the entries of $V[t]$ are generated one by one. For each value of time t in sequence, we execute one "step" of each of our n parallel zero-delay convolution processes to obtain $V_1[t] \ldots V_n[t]$, which are then fed into (7.5) to obtain $V[t]$. The value of $V[t]$ is then fed back as the next element in the input sequence of each zero-delay convolution process so that they may then compute $V_1[t+1] \ldots V_n[t+1]$, and so on. The total time required for all n of the zero-delay convolutions is $\Theta(nd \log^2 d)$, and this dominates all of the other work done by the DP algorithm.

7.3.2 Other High-Multiplicity Problem Variants

Many other stochastic DP problems fit the same general structure as our "multiple copies allowed" variant of the stochastic knapsack problem, and hence are amenable to the zero-delay convolution technique.

The Stochastic Knapsack Cover Problem. From a combinatorial optimization perspective, all of our deadline-constrained scheduling problems are "packing" problems. Many of these problems also have "covering" analogs that also have reasonable interpretations in practice. For example, the stochastic knapsack cover problem assigns costs to items (jobs) rather than values, and asks for a minimum-cost collection of items whose total size (pro-

cessing time) covers the entire knapsack capacity (exceeds our common deadline). In this case, we typically focus exclusively on the "multiple copies allowed" variants, since otherwise there might be some chance the we include all n items but still fail to cover the knapsack due to bad luck. Derman et al. [16] describe an excellent motivation for this problem in which we have a machine (e.g. an automobile) that must be kept running for T units of time and that depends on some critical component (e.g. a battery) to run. We have n types of components from which to choose, each with a deterministic cost and each having a lifetime described by a random variable of known distribution. Our task is to devise an optimal adaptive policy that selects a sequence of components over time that keeps the machine running at minimum expected cost. The DP structure for this problem is nearly identical to that of the packing variant from the preceding section, and we can use zero-delay convolution to speed its running time from $\Theta(nT^2)$ to $\Theta(nT \log^2 T)$.

The Stochastic and Dynamic Knapsack Problem. We can extend the packing DP formulations above to solve a problem variant known as the stochastic and dynamic knapsack problem [44, 36], in which jobs arrive in an on-line fashion. For example, suppose $A_j[t]$ denotes the probability that a copy of job type j arrives when precisely t units of time remain before our deadline. At this point in time if we are currently not processing any job we may choose to accept this or any other arriving job and begin processing it. Otherwise, if we decline the task it is cannot be recalled later (although another task of the same type might arrive later). Papastavrou et al. [44] describe numerous applications of stochastic and dynamic knapsack problems. Note that only packing problems make sense in this framework, and that an optimal adaptive policy might involve "idle" time. If we proceed until a deadline at time d, zero-delay convolution provides an $\Theta(nd \log^2 d)$ algorithm that optimally solves this problem, improving the $\Theta(nd^2)$ running time one would obtain without zero-delay convolution.

Time-Varying Job Values. For either the packing or covering variants of the stochastic knapsack problem (and also the stochastic and dynamic knapsack problem), our DP formulations allow us to make job values/costs vary with time without introducing any additional algorithmic complexity.

The Stochastic Shortest Path Problem. Consider a variant of the stochastic shortest path (SSP) problem whose input consists of a graph $G = (N, A)$ with $n = |N|$ nodes and $m = |A|$ directed arcs, where the length of each edge $(i, j) \in A$ is a discrete positive-integer-valued random variable l_{ij} whose distribution we denote $L_{ij}[\cdot]$, so $L_{ij}[t] = \mathbf{Pr}[l_{ij} = t]$. We are interested in finding a path from a specific source node to a specific destination node d that gives us a maximum probability of arriving by some specified deadline d. In an adaptive setting (where we choose an edge to follow, observe the instantiated length of the edge, choose another edge accordingly, etc.) we can use zero-delay convolution to improve the worst-case DP running time for this problem from $\Theta(md^2)$ to $\Theta(md \log^2 d)$. Further details can be found in [13].

8. Concluding Remarks

One of the nice (or perhaps overwhelming, depending on your point of view) aspects of the domain of scheduling is that for any new result, there is a nearly limitless supply of even more sophisticated problems one could consider for future work. In this chapter, we mention a few of the potential areas for further study along the lines of the research in this dissertation. The most obvious direction for future work is to further improve the existing results in the dissertation; for example, one could try to improve performance guarantees, simplify or generalize some of the analysis, or conduct studies on the performance of various techniques when implemented in practice.

All of the deadline-constrained problems we have considered have maximization objectives. One can also consider the mirror image of each of these problems where the goal is to minimize the cost of unscheduled jobs rather than maximize the value of scheduled jobs. Although these two types of problems are equivalent from the perspective of computing an optimal solution, the can behave very differently in terms of computing approximate solutions. There is even a significant difference between deterministic and stochastic problems in this setting. For example, the problem $1 \mid p_j \sim stoch, d_j = d \mid \mathbf{E}[\sum w_j U_j]$ asks us to devise a policy that minimizes the expected cost of jobs that we fail to schedule by the deadline. The deterministic analog of this problem is called the knapsack cover problem, and it asks us to compute a minimum-cost set of jobs whose combined processing time "covers" a space of time of size $\sum p_j - d$. In the stochastic case, we now have two distinct problem variants: for a true "covering" problem, an adaptive policy would proceed by selecting jobs one by one until their combined processing time is large enough; however, in our scheduling problems what actually happens is that the adaptive policy choose jobs one by one that do *not* belong to the cover (i.e., that are actually scheduled prior to the deadline), and stops when the deadline is exceeded.

There are quite a few different common scheduling models and constraints we could investigate in conjunction with our stochastic deadline-constrained problems. For example, we could consider shop scheduling models where jobs need to be processed on a collection of different machines (although these problems tend to be quite difficult even in a deterministic setting). In terms of constraints, an obvious type of scheduling constraint we have not considered is the precedence constraint. However, precedence constraints combined with deadlines seem to be a very problematic combination. In the deterministic case, the prob-

lem 1 | $prec, d_j = d$ | $\mathbf{E}[\sum w_j \overline{U}_j]$ is known as either the *precedence constrained knapsack problem* or the *precedence ordered knapsack problem* [37], and very little is known about its approximability in general (the problem is NP-hard since it generalizes the knapsack problem).

We have investigated quite a few interesting models in this dissertation (e.g., adaptive, non-adaptive, ordered adaptive, fixed set, etc.) but there are still more models that are perhaps reasonable to consider. One example that comes to mind is the "blackjack" model, which is similar to the adaptive model but allows us to stop scheduling at any time and keep our "earnings", or risk losing everything if we reach the deadline. Another interesting model involves deterministic processing times but a random deadline (the distribution of which is known in advance). In this case, since adaptivity is useless we would seek a non-adaptive policy that maximizes our expected value (in either the start deadline or completion deadline models). Finally, we can reconsider our assumptions that forbid cancellation and preemption of jobs. It may be possible to achieve reasonable approximation results if jobs can be terminated early or preempted, but remember that in this case our algorithms *must* take advantage of this extra flexibility or else they cannot hope to achieve approximation guarantees better than $\Omega(n)$ in the worst case.

Another interesting direction might be to consider deadline-constrained stochastic scheduling problems in which "hard" deadline constraints are replaced by "soft" penalties to our objective function. In deterministic scheduling, a common example of this type of objective is minimizing $\sum w_j(1 - e^{rC_j})$. It turns out that the optimal ordering in this case is to use decreasing order in terms of

$$\frac{w_j e^{rp_j}}{1 - e^{rp_j}},$$

as one can prove using a simple interchange argument. More generally, one can consider the objective $\sum f_j(C_j)$ for non-decreasing functions f_j, although this case is NP-hard as it generalizes the objective $\sum w_j U_j$. In a stochastic setting, Weber, Varaiya and Walrand [61] show that if the f_j's are convex and processing times are stochastically ordered as $p_1 \leq_{st} \ldots \leq_{st} p_n$ (by $p_i \leq_{st} p_j$ we mean that p_i is stochastically smaller than p_j: $\mathbf{Pr}[p_i \geq x] \leq \mathbf{Pr}[p_j \geq x]$ for all values of x), then list scheduling in this order (i.e., SEPT) is an optimal policy for minimizing $\mathbf{E}[\sum_j f_j(C_j)]$ on identical parallel machines without preemption. It might be interesting to consider generalizing these results (either by removing restrictions on the distributions of the p_j's or by considering a more general class of f_j's) with a goal of computing good approximation algorithms.

Bibliography

[1] Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph Naor, and Baruch Schieber. A unified approach to approximating resource allocation and scheduling. *Journal of the ACM*, 48(5):1069–1090, 2001.

[2] Michael A. Bender, S. Muthukrishnan, and Rajmohan Rajaraman. Improved algorithms for stretch scheduling. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 762–771, 2002.

[3] John R. Birge and François V. Louveaux. *Introduction to Stochastic Programming.* Springer Verlag, New York, 1997.

[4] Allan Borodin, Morten N. Nielsen, and Charles Rackoff. (incremental) priority algorithms. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 752–761. Society for Industrial and Applied Mathematics, 2002.

[5] John Bruno, Peter Downey, and Greg N. Frederickson. Sequencing tasks with exponential service times to minimize the expected flow time or makespan. *Journal of the ACM*, 28(1):100–113, 1981.

[6] James M. Calvin and Joseph Y-T. Leung. Average-case analysis of a greedy algorithm for the 0/1 knapsack problem. *Operations Research Letters*, 31(3):202–210, 2003.

[7] Robert L. Carraway, Robert L. Schmidt, and Lawrence R. Weatherford. An algorithm for maximizing target achievement in the stochastic knapsack problem with normal returns. *Naval Research Logistics*, 40:161–173, 1993.

[8] Cheng-Shang Chang, XiuLi Chao, Michael Pinedo, and Richard R. Weber. On the optimality of LEPT and $c\mu$ rules for machines in parallel. *Journal of Applied Probability*, 29:667–681, 1992.

[9] Chandra Chekuri and Sanjeev Khanna. A PTAS for the multiple knapsack problem. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 213–222, 2000.

[10] Herman Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, 23(4):493–507, 1952.

[11] Ed G. Coffmann and Edgar N. Gilbert. Expected relative performance of list scheduling rules. *Operations Research*, 33(3):548–561, 1985.

[12] Sashka Davis and Russell Impagliazzo. Models of greedy algorithms for graph problems. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 381–390, 2004.

[13] Brian C. Dean. Speeding up stochastic dynamic programming with zero-delay convolution, 2004. Submitted for publication.

[14] Brian C. Dean, Michel X. Goemans, and Jan Vondrák. Approximating the stochastic knapsack problem: the benefit of adaptivity. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 208–217, 2004.

[15] Brian C. Dean, Michel X. Goemans, and Jan Vondrák. Adaptivity and approximation for stochastic packing problems. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2005.

[16] Cyrus Derman, Gerald. J. Lieberman, and Sheldon M. Ross. A renewal decision problem. *Management Science*, 24(5):554–561, 1978.

[17] Joseph L. Doob. *Stochastic Processes.* John Wiley, NY, 1965.

[18] Hamilton Emmons and Michael Pinedo. Scheduling stochastic jobs with due dates on parallel machines. *European Journal of Operations Research*, 47:49–55, 1990.

[19] Uriel Feige. On the sum of nonnegative independent random variables with unbounded variance, 2003.

[20] Zvi Galil and Kunsoo Park. Dynamic programming with convexity, concavity and sparsity. *Theoretical Computer Science*, 92:49–76, 1992.

[21] William G. Gardner. Efficient convolution without input-output delay. *Journal of the Audio Engineering Society*, 43(3):127–136, 1995.

[22] Kevin D. Glazebrook. Scheduling tasks with exponential service times on parallel processors. *Journal of Applied Probability*, 16:685–689, 1979.

[23] Ashish Goel and Piotr Indyk. Stochastic load balancing and related problems. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 579–586, 1999.

[24] A. V. Goldberg and A. Marchetti-Spaccamela. On finding the exact solution of a zero-one knapsack problem. In *Proceedings of the 16th annual ACM Symposium on Theory of Computing (STOC)*, pages 359–368, 1984.

[25] Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Math*, 17:416–426, 1969.

[26] Ronald L. Graham, Eugene L. Lawler, Jan Karel Lenstra, and Alexander H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[27] Jane Hagstrom. Computational complexity of PERT problems. *Networks*, 18:139–147, 1988.

[28] Leslie Hall. Approximation algorithms for scheduling. In Dorit Hochbaum, editor, *Approximation Algorithms for NP-hard Problems*, chapter 1, pages 1–45. PWS Publishing, 1996.

[29] Mordechai Henig. Risk criteria in a stochastic knapsack problem. *Operations Research*, 38(5):820–825, 1990.

[30] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.

[31] Oscar H. Ibarra and Chul E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM*, 22(4):463–468, 1975.

[32] Nicole Immorlica, David Karger, Maria Minkoff, and Vahab Mirrokni. On the costs and benefits of procrastination: Approximation algorithms for stochastic combinatorial optimization problems. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 184–693, 2004.

[33] Thomas Kampke. On the optimality of static priority policies in stochastic scheduling on parallel machines. *Journal of Applied Probability*, 24:430–448, 1987.

[34] David Karger, Cliff Stein, and Joel Wein. Scheduling algorithms. In M.J. Atallah, editor, *CRC Handbook on Algorithms and Theory of Computation*, chapter 35. CRC Press, 1998.

[35] Jon Kleinberg, Yuval Rabani, and Éva Tardos. Allocating bandwidth for bursty connections. *SIAM Journal on Computing*, 30(1):191–217, 2000.

[36] Anton Kleywegt and Jason D. Papastavrou. The dynamic and stochastic knapsack problem with random sized items. *Operations Research*, 49(1):26–41, 2001.

[37] Stavros Kolliopoulos and George Steiner. Partially-ordered knapsack and applications to scheduling. In *Proceedings of the European Symposium on Algorithms (ESA)*, pages 612–624, 2002.

[38] Elias Koutsoupias and Christos H. Papadimitriou. Beyond competitive analysis. *SIAM Journal on Computing*, 30(1):300–317, 2000.

[39] George S. Leuker. On the average difference between the solutions to the linear and integer knapsack problems. In R.L. Disney and T.J. Offa, editors, *Applied Probability - Computer Science: The Interface, Vol I*, pages 489–504. Birkhauser, Boston, 1982.

[40] Nicole Megow, Marc Uetz, and Tjark Vredeveld. Stochastic online scheduling on parallel machines. To appear in Proceedings of the Second Workshop on Approximation and Online Algorithms (WAOA), 2004.

[41] Rolf H. Möhring, Andreas S. Schulz, and Marc Uetz. Approximation in stochastic scheduling: the power of LP-based priority policies. *Journal of the ACM*, 46(6):924–942, 1999.

[42] J. Michael Moore. An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science*, 15(1):102–109, 1968.

[43] Christos H. Papadimitriou. Games against nature. *Journal of Computer and System Sciences*, 31:288–301, 1985.

[44] Jason D. Papastavrou, S. Rajagopalan, and Anton Kleywegt. The dynamic and stochastic knapsack problem with deadlines. *Management Science*, 42(12):1706–1718, 1996.

[45] Michael Pinedo. Stochastic scheduling with release dates and due dates. *Operations Reesearch*, 31:559–572, 1983.

[46] Michael Pinedo. *Scheduling: Theory, Algorithms, and Systems.* Prentice Hall, 2002.

[47] R. Ravi and Amitabh Sinha. Hedging uncertainty: Approximation algorithms for stochastic optimization problems. In *Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 101–115, 2004.

[48] Michael Rothkopf. Scheduling with random service times. *Management Science*, 12(9):707–713, 1966.

[49] Mark Scharbrodt, Thomas Schickinger, and Angelika Steger. A new average case analysis for completion time scheduling. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 170–178. ACM Press, 2002.

[50] David B. Shmoys and Éva Tardos. Scheduling unrelated machines with costs. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, pages 448–454, 1993.

[51] David B. Shmoys and Chaitanya Swamy. Stochastic optimization is (almost) as easy as deterministic optimization. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 228–237, 2004.

[52] Martin Skutella and Marc Uetz. Stochastic machine scheduling with precedence constraints. To appear in the SIAM Journal on Computing.

[53] Wayne E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3:59–66, 1956.

[54] Moshe Sniedovich. Preference order stochastic knapsack problems: methodological issues. *The Journal of the Operational Research Society*, 31(11):1025–1032, 1980.

[55] E. Steinberg and M.S. Parks. A preference order dynamic program for a knapsack problem with stochastic rewards. *The Journal of the Operational Research Society*, 30(2):141–147, 1979.

[56] Marc Uetz. *Algorithms for deterministic and stochastic scheduling.* PhD thesis, Institut für Mathematik, Technische Universität Berlin, 2001.

[57] Marjan van den Akker and Han Hoogeveen. Minimizing the number of late jobs in case of stochastic processing times with minimum success probabilities. Technical report, Institute of Information and Computation Sciences, Utrecht University, 2004.

[58] Jan Vondrák. *Probabilistic methods in combinatorial and stochastic optimization.* PhD thesis, Massachusetts Institute of Technology, 2005.

[59] Richard Weber. Stochastic scheduling bibliography. `http://www.statslab.cam.ac.uk/~rrw1/stoc_sched/index.html`.

[60] Richard Weber. *Optimal organization of multi-server systems.* PhD thesis, Cambridge University, 1980.

[61] Richard Weber, Pravin Varaiya, and Jean Walrand. Scheduling jobs with stochastically ordered processing times on parallel machines to minimize expected flowtime. *Journal of Applied Probability*, 23:841–847, 1986.

[62] Gideon Weiss and Michael Pinedo. Scheduling tasks with exponential service times on non-identical processors to minimize various cost functions. *Journal of Applied Probability*, 17:187–202, 1980.