12-2010

A PTAS for the Uncertain Capacity Knapsack Problem

Matthew Dabney
Clemson University, mdabney@clemson.edu

A PTAS FOR THE UNCERTAIN CAPACITY KNAPSACK PROBLEM

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Computer Science

---

by
Matthew H. Dabney
November 2010

Accepted by:
Brian C. Dean, Ph.D., Committee Chair
David Jacobs, Ph.D.
Jason Hallstrom, Ph.D.

Abstract

The standard NP-hard knapsack problem can be interpreted as a scheduling problem with n jobs with weights $w_1 \dots w_n$ and processing times $p_1 \dots p_n$, where our goal is to order the jobs on a single machine so as to maximize the weight of all jobs completing prior to a known common deadline d. In this paper, we study the *uncertain capacity knapsack problem* (UCKP), a generalization of this problem in which the deadline d is not known with certainty, but rather is provided as a probability distribution, and our goal is to order the jobs so as to maximize the expected weight of the set of jobs completing by the deadline. We develop a polynomial-time approximation scheme (PTAS) for this problem. We make no assumptions about probability distributions except that each job, scheduled by itself, completes by the deadline with some constant probability.

Table of Contents

List of Figures

Chapter 1

Introduction

In this paper, we present a polynomial-time approximation scheme (PTAS) for the *uncertain capacity knapsack problem* (UCKP), a natural stochastic generalization of the classical NP-hard knapsack problem. The UCKP is perhaps best motivated as a scheduling problem. In this context, the standard knapsack problem takes as input n jobs with weights $w_1 \dots w_n$ and processing times $p_1 \dots p_n$, and its goal is to compute an ordering of the jobs on one machine to maximize the weight of the jobs that complete by a common deadline d. In three-field scheduling notation (e.g., see [5]), we would write the problem as $1 \mid \overline{d}_j = d \mid \max \sum w_j(1-U_j)$, where U_j is an indicator variable taking the value 1 if job j completes after the deadline (i.e., if $C_j > d$). In the UCKP, the common deadline d shared by all jobs is no longer known with certainty, but rather described by an arbitrary probability distribution, provided in the form of an oracle function $f(t) = \mathbf{Pr}[d \geq t]$. Here, our goal is to compute an ordering of the jobs on a single machine so as to maximize the expected weight of the jobs completing by the deadline: $1 \mid \overline{d}_j = d, d \sim stoch \mid \max \mathbf{E}[\sum w_j(1-U_j)]$. The problem is "non-adaptive" in the sense that we cannot change the ordering algorithm later on, after realizing that the deadline has not yet elapsed by a certain point in time in the scheduling process. In fact, adaptivity is not useful in the case of UCKP because knowing that the deadline has not yet occurred gives no new useful information. In addition, preemption is not useful for UCKP because jobs only contribute their weight upon completion, so reordering the jobs in a preemptive schedule to remove preemption cannot decrease the expected value of the solution. The UCKP

has natural applications in scheduling, since we shall see in a moment that it exactly models the general problem of scheduling to maximize the sum of time-discounted rewards of jobs. If time is replaced with money, it can also serve as a good model for a resource allocation problem subject to an uncertain total budget.

Based on its completion time C_j, the probability that job j completes by the deadline d is $\mathbf{E}[1-U_j] = f(C_j)$. By linearity of expectation, we can therefore rephrase the UCKP as the scheduling problem $1 \mid \overline{d}_j = d, d \sim stoch \mid \max \sum w_j f(C_j)$, whose objective involves maximizing the sum of the "discounted weight" $w_j f(C_j)$ over all jobs j. That is, job j has its weight penalized by a factor $f(C_j)$ that decreases from one to zero over time so it contributes less and less the later it completes. In our solution, we make the assumption that for all jobs j, $f(p_j) \geq \delta$, where δ is a fixed constant independent of n. That is, the probability that any job, scheduled by itself, will complete by the deadline is greater than some constant value δ.

According to Rothkopf and Smith [6], an optimal schedule can be computed greedily if and only if f has the form $f(t) = -kt$ or $f(t) = e^{-kt}$. The problem becomes NP-hard even for f with rather simple structure; for example, in the special case of the standard knapsack problem with deadline d, we have $f(t \leq d) = 1$ and $f(t > d) = 0$.

Sahni [7] gave the first polynomial-time approximation scheme (PTAS) for the standard knapsack problem, delivering a solution of weight at least $(1-\varepsilon)OPT$ in time polynomial in n for any $\varepsilon = O(1)$. Ibarra and Kim [4] gave the first fully-polynomial approximation scheme (FPTAS), running in time polynomial in n and $1/\varepsilon$. In this paper, we give the first PTAS for the UCKP, which to the best of our knowledge has never yet been studied from an approximation point of view. Our approach combines new structural insight as well as a number of "standard" approximation techniques, although it seems to take a surprisingly complicated combination of such techniques to make headway in approximating the UCKP.

To explain our approximation algorithm for UCKP, we first describe a simplified version for approximating the standard knapsack problem, which has a common high-level structure with our algorithm for UCKP. We discuss this simplified algorithm in Chapter 3 after going through preliminary concepts in Chapter 2. Following this, we discuss the full algorithm for UCKP in

Chapter 4.

Chapter 2

Preliminaries

We use several techniques described in this chapter common to both our PTAS for the standard knapsack problem and our PTAS for UCKP. Many of these techniques will cause us to lose $(1+\varepsilon)$ factors in our approximation of an optimal schedule. However, as long as we only lose a constant number of these factors along the way, we can still achieve an expected solution objective of at least $(1-\varepsilon)$ by choosing an appropriate initial value of ε.

Our notation is for the most part standard. If S is a subset of jobs, then we let $p(S)$ and $w(S)$ respectively denote $\sum_{j\in S} p_j$ and $\sum_{j\in S} w_j$. We assume that the values of $p_1 \ldots p_n$, $w_1 \ldots w_n$, and $f(t)$ are b-bit integers, so our final running time will be polynomial in n and b.

2.1 Guessing OPT

We guess a value B such that $\frac{OPT}{2} \leq B \leq OPT$, where OPT denotes the optimal solution weight. In order to guess B in polynomial time, we observe that $OPT \in [L, nL]$, where $L = \max_j w_j f(p_j)$ denotes the maximum discounted weight obtained from scheduling only a single job (we call the quantity $w_j f(p_j)$ the *best-case weight* of job j, since it reflects the maximum amount we could receive from j in our objective function). We can then guess B by trying $B = L, 2L, 4L, \ldots, nL$; this requires at most $\log_2 n$ guesses. We run the remainder of our algorithm for each guess, taking the best answer we obtain.

2.2 Rounding

We delete all jobs j with weight $w_j \leq \frac{\varepsilon B}{n} \leq \frac{\varepsilon OPT}{n}$. The total discounted weight all such jobs contribute to the objective function of an optimal solution is at most εOPT, so their removal has a negligible impact. Observe that all weights w_j now satisfy $\frac{\varepsilon B}{n} \leq w_j \leq \frac{2B}{\delta}$ (since $w_j \delta \leq w_j f(p_j) \leq OPT \leq 2B$) and the ratio of the largest weight to the smallest weight is at most $\frac{2n}{\varepsilon\delta}$. We then round down weight w_j of each job so its weight drops to the next-lowest value of $\frac{2B}{\delta(1+\varepsilon)^i}$ (for $i = 0, 1, 2, \ldots$). This effectively partitions our jobs into M weight classes $W_1, \ldots, W_M$ with $M = \log_{1+\varepsilon} \frac{2n}{\varepsilon\delta} = O(\log n)$, where W_i contains all jobs j with original weights w_j in the range $\left(\frac{2B}{\delta(1+\varepsilon)^{i-1}}, \frac{2B}{\delta(1+\varepsilon)^i}\right]$. Since each job weight decreases by at most a $(1+\varepsilon)$ factor during this process, this deflates the weight of an optimal solution by at most a $(1+\varepsilon)$ relative factor. Henceforth, we consider all job weights to be rounded.

2.3 Cheap and Expensive Jobs

We define *expensive* jobs to be those that have weight $w_j \geq \varepsilon B$. *Cheap* jobs are those that have weight $w_j < \varepsilon B < \varepsilon OPT$. Note that after rounding there can only be $E = \log_{1+\varepsilon} \frac{2}{\varepsilon\delta} = O(1)$ expensive job classes before the weight of the class drops below εB. We note that without enforcing the assumption $f(p_j) \geq \delta$, we would have more than $O(1)$ expensive weight classes and the running time of our algorithm would be quasi-polynomial.

2.4 Integral Versus Fractional Schedules

We let $\sigma = (\sigma_E, \sigma_C)$ denote a schedule of our n jobs, with σ_E representing the subset of the schedule pertaining only to the expensive jobs, and σ_C representing the schedule pertaining only to the cheap jobs. An *integral* (or *non-preemptive*) schedule σ is simply an ordering of the n jobs in our input. A *fractional* (or *preemptive*) schedule allows jobs to be broken up and processed in multiple non-consecutive pieces. To represent a fractional (or integral) schedule, we use an indicator function $I_j(t)$ for each job j such that $I_j(t) = 1$ at all points in time t during which j is active. The

completion time of a job j is then $C_j = \sup\{t : I_j(t) = 1\}$.

The original objective function for integer solutions in which all jobs are integrally (non-preemptively) scheduled is represented by

$$V(\sigma) = \sum_j w_j f(C_j). \tag{2.1}$$

We can also evaluate schedules (both integral and fractional) using a relaxed, fractional objective based on the discounted weight over the entire extent of a job rather than just its completion time:

$$V'(\sigma) = \sum_j \frac{w_j}{p_j} \int_{t=0}^{\infty} I_j(t) f(t) dt. \tag{2.2}$$

We note that $V'(\sigma) \geq V(\sigma)$ due to the fact that $f(t)$ is a monotonically decreasing function. We will often deal with a fractional schedule $\sigma = (\sigma_E, \sigma_C)$ in which the integral objective V is used to evaluate the expensive jobs (typically integrally scheduled), while the fractional objective V' is used to evaluate the fractional jobs (typically fractionally scheduled). In this case, we have $V'(\sigma) \geq V(\sigma_E) + V'(\sigma_C) \geq V(\sigma)$. Also note that one can easily maximize $V'(\sigma)$ by greedily scheduling jobs in non-increasing order of w_j/p_j.

Chapter 3

An Alternative Knapsack PTAS

To help describe our PTAS for UCKP, we first discuss an alternative PTAS for the standard knapsack problem with the same high-level structure. Recall that we have already used the technique described in Chapter 2.1 to guess a value B such that $\frac{OPT}{2} \leq B \leq OPT$, rounded job weights to obtain a logarithmic number of weight classes as described in Section 2.2, and divided the jobs into *cheap* and *expensive* jobs, as seen in Section 2.3, with only a constant number of expensive job classes.

3.1 Special Solutions

Let us define a "special" knapsack solution σ as any ordering that begins with a subset S of expensive jobs (arbitrarily ordered), followed by all cheap jobs ordered greedily in order of non-increasing w_j/p_j, followed by the remaining expensive jobs.

We now make three key observations that explain the high-level structure of our knapsack PTAS:

1. There always exists a special solution $\sigma = (\sigma_E, \sigma_C)$ such that $V(\sigma_E) + V'(\sigma_C) \geq OPT$.

2. Any special solution $\sigma = (\sigma_E, \sigma_C)$ can be converted to a schedule $\overline{\sigma}$ for which $V(\overline{\sigma}) \geq (1 - \varepsilon)(V(\sigma_E) + V'(\sigma_C))$.

3. One can compute a special solution $\sigma = (\sigma_E, \sigma_C)$ maximizing $V(\sigma_E) + V'(\sigma_C)$ in polynomial

time.

We need only establish these three points to complete the description of our PTAS. The first observation is easy to see by starting with an optimal schedule and rearranging the jobs completing by time d so expensive jobs come first, followed by cheap jobs ordered greedily. The remaining jobs (those completing after the deadline d) can be reordered so they consist of the remaining cheap jobs followed by the remaining expensive jobs. By using the objective V' instead of V to account for the cheap jobs, we can only increase our total objective value, since now we can count the fractional contribution of the single cheap job that runs over the deadline d. Further, we can only increase our objective by reordering the cheap jobs greedily in non-increasing order by w_j/p_j.

The second observation is also easy to establish: by setting $\overline{\sigma} = \sigma$, we obtain a schedule for which $V(\overline{\sigma})$ counts all of the same job weights that were counted by $V(\sigma_E) + V'(\sigma_C)$, except for the single cheap job j partially overflowing the deadline. By removing this cheap job from consideration, we lose at most $w_j \leq \varepsilon B \leq \varepsilon OPT$.

For the third observation, we introduce the notion of a *prefix set*. Suppose the jobs in each of our E expensive weight classes are ordered in non-decreasing order of p_j. A set of expensive jobs is called a prefix set if it consists of a prefix of each of these orderings – containing only some number of the shortest jobs from each of the expensive weight classes. Since $E = O(1)$, there are at most $n^E = n^{O(1)}$ different prefix sets. We commonly represent a prefix set by a length-E vector s for which s_i gives the number of jobs from weight class i. We also denote by $p(s)$ and $w(s)$ the aggregate processing time and weight of all jobs in s.

Lemma 1. *For every instance of the UCKP, there exists an optimal schedule σ (after we round job weights as in Section 2.2) such that for any time $t \geq 0$, the set of expensive jobs with $C_j \leq t$ is a prefix set.*

Proof. Consider any optimal schedule σ not satisfying the property in the Lemma. Then σ must contain two jobs j and j' belonging to the same weight class W_i, with $C_j < C_{j'}$ but with j appearing later than j' in the sorted ordering of jobs within W_i. Hence, $p_j \geq p_{j'}$, so if we swap j and j', this cannot increase the completion time of any job in our schedule, and therefore it cannot increase

$V(\sigma)$. By repeatedly performing swaps of this nature, we can transform σ so it satisfies the Lemma without increasing $V(\sigma)$ during the process. □

Noting that the proof of Lemma 1 applies also to special solutions, we can now complete our discussion of the third observation above. To find an optimal special solution $\sigma = (\sigma_E, \sigma_C)$, we see that the subset of expensive jobs completing by time d can be assumed to be a prefix set, and there are only $n^{0(1)}$ such sets, permitting us to enumerate them all. For each such valid set s (with $p(s) \leq d$), we complete it with a greedy ordering of cheap jobs, and we take the best such schedule to be our optimal special solution.

Chapter 4

Generalizing the PTAS for UCKP

We now generalize the approach from the previous chapter to obtain a PTAS for the UCKP. Here, we redefine expensive jobs to be those with weight $w_j \geq \varepsilon^2 B$ (rather than εB as before). Cheap jobs are now those with weight less than $\varepsilon^2 B < \varepsilon^2 OPT$. Note that expensive jobs still comprise only $O(1)$ weight classes, so in particular there are still at most $n^{O(1)}$ different prefix sets of expensive jobs.

4.1 Discretizing $f(t)$

To deal with the arbitrary nature of $f(t)$ more easily, we round it to a piecewise constant function $\hat{f}(t)$ as follows. Letting $p_{min} = \min_j p_j$, we first set $\hat{f}(t \leq p_{min}) = f(p_{min})$, noting that this change cannot affect the objective value of any integral solution. We then binary search for the minimum value of t^* for which $f(t \geq t^*) \leq \varepsilon\delta/n$. This search requires only polynomial time in n and b. Recall that $L = \max_j w_j f(p_j)$ is the maximum discounted value we could get from scheduling any single job, and that $L \leq OPT$. Due to our assumption, we know that $f(p_{max}) \geq \delta$. Hence, for any job j, we have $L \geq w_j f(p_j) \geq w_j \delta$, so $w_j \leq \frac{L}{\delta}$. Using this inequality for w_j along with the inequality for $f(t \geq t^*) \leq \varepsilon\delta/n$, we see that for any j such that $C_j > t^*$, $w_j f(C_j) \leq \frac{L}{\delta} \cdot \frac{\varepsilon\delta}{n} \leq \frac{\varepsilon L}{n} \leq \frac{\varepsilon OPT}{n}$. Discarding n of these jobs would cause our objective to drop by at most εOPT, so we can safely set $\hat{f}(t \geq t^*) = 0$.

Between $t = p_{min}$ and $t = t^*$, we discretize f so that $f(t) \in [\hat{f}(t), (1+\varepsilon)\hat{f}(t)]$, meaning that if compute the objective $V(\sigma)$ using $\hat{f}$ instead of f, the discounted weight of every job j with $C_j \in [p_{min}, t^*)$ drops by at most a $(1+\varepsilon)$ relative factor, so it suffices to consider formulating a PTAS using $\hat{f}$ instead of f. Discretization is straightforward: letting $t_0 = p_{min}$, we binary search for the minimum value of t_1 such that $f(t \geq t_1) \leq f(t_0)/(1+\varepsilon)$. We then set $\hat{f}(t) = f(t_1)$ over $t \in [t_0, t_1)$, binary search for the minimum t_2 such that $f(t \geq t_2) \leq f(t_1)/(1+\varepsilon)$, and so on. We call the range of time $[t_{i-1}, t_i]$ the ith *region*, and note that we need at most $R = \log_{1+\varepsilon}(\frac{n}{\varepsilon\delta}) = O(\log n)$ regions before we reach time t^*. We henceforth use $\hat{f}$, a piecewise constant function with $O(\log n)$ pieces, for the remainder of our algorithm.

4.2 Optimal Solutions Involving Only Expensive Jobs

Suppose all jobs happen to be expensive. It will be useful to see how this special case can be solved in polynomial time, since this will end up being a subroutine in the more general algorithm yet to come. According to Lemma 1, we can describe the ordering of jobs in an optimal solution in terms of a succession of prefix sets $s(1) \ldots s(n)$, each one extending the preceding set by the addition of a single job. That is, $s(i) \geq s(i-1)$ and $||s(i) - s(i-1)||_1 = 1$. In general, we say prefix set s' *extends* prefix set s if $s \leq s'$ and $||s - s'||_1 = 1$. In this case, we let $j(s, s')$ denote the unique job in s' not belonging to s.

We now compute an optimal ordering of prefix sets by solving a longest path problem in a directed acyclic graph G whose vertices are prefix sets, with a directed edge (s, s') of length $w_{j(s,s')} f(p(s'))$ whenever s' extends s. A longest path through G from the empty prefix set to the prefix set containing all jobs will then give us a schedule σ maximizing $V(\sigma)$. Since G has only a polynomial number of vertices, this computation takes polynomial time.

4.3 Special UCKP Solutions

Let us say that a schedule σ is *right-justified* if, for every expensive job j, the entire region of time from C_j up to the next region boundary of $\hat{f}(t \geq C_j)$ is filled exclusively with expensive

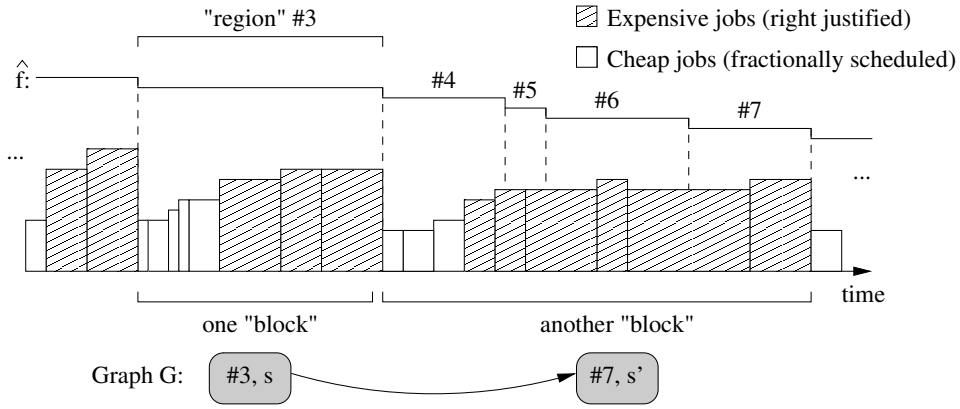

Figure 4.1: Diagram of a special solution, with expensive jobs shaded and cheap jobs not shaded; the height of a job indicates its weight class. Expensive jobs are right-justified and scheduled integrally, and cheap jobs are fractionally scheduled in the remaining gaps in a greedy fashion. Regions of the piecewise linear discount function $\hat{f}$ are numbered and shown separated by dashed lines. Blocks of regions are also indicated, each with a corresponding node in G labeled with the ending region of the block and the prefix set of expensive jobs up until the ending point of the block.

jobs. In such a schedule, each expensive job is pushed as far "to the right" (forward in time) as possible, subject to staying in order with its surrounding expensive jobs, and also to the constraint that its completion time must not cross a region boundary for $\hat{f}$. We now say that a "special" solution $\sigma = (\sigma_E, \sigma_C)$ for a UCKP instance is a schedule in which all expensive jobs are integrally scheduled in a right-justified fashion, with any remaining gaps filled in by fractionally scheduled cheap jobs, ordered greedily in non-increasing order by w_j/p_j. The structure of a special schedule is shown in Figure 4.1.

Again, our PTAS follows from three key observations:

1. There always exists a special solution $\sigma = (\sigma_E, \sigma_C)$ such that $V(\sigma_E) + V'(\sigma_C) \geq OPT$,

2. Any special solution $\sigma = (\sigma_E, \sigma_C)$ can be converted to a valid integral schedule $\overline{\sigma}$ for which $V(\overline{\sigma}) \geq (1-2\varepsilon)(V(\sigma_E) + V'(\sigma_C))$.

3. One can compute a special solution $\sigma = (\sigma_E, \sigma_C)$ maximizing $V(\sigma_E) + V'(\sigma_C)$ in polynomial time.

To prove the first observation, note that jobs can be re-ordered within each region of $\hat{f}$ without affecting the objective value of our solution. This allows us to start with an optimal schedule $\sigma = (\sigma_E, \sigma_C)$ and right-justify it by shifting the expensive jobs as far "to the right" within their respective regions of $\hat{f}$ as possible (so after shifting, each job completes in the same region as before and so contributes the same discounted weight). During this process, if we want to shift right an expensive job j followed by a cheap job j', we fractionally displace j' and re-introduce j' to the left of j to fill the gap left by moving j to the right. As a result, the cheap jobs now become fractionally scheduled, but since no expensive job has moved its completion time out of its original region of $\hat{f}$ and the fractional composition of each cheap job only moves earlier in time, the value of $V(\sigma_E) + V'(\sigma_C)$ can only increase. If we now remove all of the cheap job mass and fill in the resulting gaps by fractionally re-introducing the cheap jobs in non-increasing order of w_j/p_j, this can also cause the value of $V(\sigma_E) + V'(\sigma_C)$ only to increase. The result is a special schedule for which $V(\sigma_E) + V'(\sigma_C) \geq OPT$.

To achieve the second observation, we form $\overline{\sigma}$ from σ by deleting all cheap jobs straddling region boundaries of $\hat{f}$. Note that only these jobs can be fractionally scheduled, so the resulting schedule $\overline{\sigma}$ is integral. The total amount of weight lost during the formation of $\overline{\sigma}$ is at most the maximum weight of a cheap job ($\varepsilon^2 OPT$) times the discount factor $\hat{f}$ from each region r, which we upper bound by

$$\sum_{r=0}^{\infty} \frac{1}{(1+\varepsilon)^r} = \frac{1}{1-\frac{1}{1+\varepsilon}} = \frac{1}{\varepsilon} + 1.$$

Multiplying the two together, we have $\varepsilon^2 OPT(\frac{1}{\varepsilon} + 1) = \varepsilon OPT + \varepsilon^2 OPT \leq 2\varepsilon OPT$ for $\varepsilon \leq 1$.

4.4 Computing an Optimal Special Solution

To complete our PTAS by proving the third observation above, we construct an optimal special solution by finding a longest path in a DAG, in a somewhat similar fashion to the method used in Section 4.2 (although this method also will be used as a subroutine here). As shown in Figure 4.1, a special solution can be decomposed into *blocks*, where each block consists of a prefix of fractionally-scheduled cheap jobs, followed by a consecutive block of integrally-scheduled expen-

sive jobs, all right-aligned to the boundary of some region r of $\hat{f}$. Since the proof of Lemma 1 holds for special solutions as defined above, we know that the set of expensive jobs in each block can be characterized as the difference between two prefix solutions of expensive jobs — one containing all expensive jobs up to the end of the block, and the other containing all expensive jobs up to the end of the preceding block.

We now build an optimal special solution stepping one block at a time by computing the longest path through a directed acyclic graph G. Each vertex of G is of the form (r,s), where r indexes a region of $\hat{f}$ ending at time t_r, and s is a prefix set with $p(s) \leq t_r$. We have an edge from vertex (r,s) to vertex (r',s') if $r' > r$, $s' > s$, and $p(s') - p(s) \leq t_{r'} - t_r$. The interpretation of following such an edge is that we move from a block ending at region r (at time t_r) with prefix set s of expensive jobs to a block ending at region r' (ending at time $t_{r'}$) with larger prefix set s' of expensive jobs.

The somewhat challenging aspect of constructing G is the assignment of lengths to its edges. Consider an edge e from (r,s) to (r',s'). Its length $l(e)$ should reflect the discounted weight of all jobs, cheap and expensive, scheduled from time t_r through time $t_{r'}$. We compute $l(e) = l_E(e) + l_C(e)$, where $l_E(e)$ is the contribution due to expensive jobs and $l_C(e)$ is the contribution due to cheap jobs, as follows:

- **Expensive Jobs.** To compute $l_E(e)$, we note that we know exactly what expensive jobs must be scheduled in the block corresponding to edge e — exactly those jobs belonging to the prefix set s' but not s. Moreover, we know the exact region of time $[t_{r'} - (p(s') - p(s)), t_{r'}]$ over which these jobs must be scheduled. The only question remaining is how to order these jobs so that we obtain as much discounted weight from them as possible (remember that these jobs might span multiple regions in $\hat{f}$, so different orderings can have a large impact on our objective). However, we observe that this ordering problem can be cast as an instance of the special case of UCKP where all jobs are expensive, which we optimally solved in Section 4.2 by finding a longest path through a DAG. Hence, we can compute $l_E(e)$ in polynomial time.

- **Cheap Jobs.** Consider the set of all cheap jobs ordered so $w_1/p_1 \geq w_2/p_2 \geq \ldots$, and let $\phi(t)$

denote the value of w_j/p_j of the job j scheduled at time t if we schedule only the cheap jobs in this order. That is, $\phi(0 \leq t \leq p_1) = w_1/p_1$, $\phi(p_1 \leq t \leq p_1 + p_2) = w_2/p_2$, and so on. We can now compute $l_C(e)$ by observing that we know exactly which cheap jobs must be scheduled within the block corresponding to edge e. We know that $\tau = t_{r'} - t_r - (p(s') - p(s))$ units of time worth of cheap jobs must be present in the block, and that $t_r - p(s)$ units of cheap jobs have already been scheduled prior to the start of the block. Hence, the total value of the cheap jobs within the block is given by

$$l_C(e) = \int_{t=0}^{\tau} \phi(t_r - p(s) + t)\hat{f}(t_r + t)dt,$$

and this can be easily evaluated in polynomial time since both ϕ and $\hat{f}$ are piecewise constant with a polynomial number of pieces.

Just as in Section 4.2, a longest path through G now tells us the optimal set of blocks to step through while constructing an optimal special solution. Since G has a polynomial number of vertices, and each edge length takes only polynomial time to compute, we can perform this computation in only polynomial time, thereby concluding the description of our PTAS for the UCKP.

Chapter 5

Concluding Remarks

We have defined a polynomial-time approximation scheme for the uncertain capacity knapsack problem. An open problem remains to remove the assumption that each job j must satisfy $f(p_j) = \Omega(1)$. There are several related problems to the standard knapsack problem that admit polynomial-time approximation schemes in the deterministic case for which one could consider constructing similar algorithms for the stochastic variant. For example, versions of the multiple knapsack problem (MKP) [2] and multidimensional knapsack problem (d-KP) [1] with stochastic deadlines would be interesting to consider for finding polynomial-time approximation schemes. A covering variant in which one would seek to cover an uncertain duration while minimizing the cost of jobs scheduled within that duration is another interesting stochastic problem; this variant yields a FPTAS in the deterministic case [3].

Bibliography

[1] Alberto Caprara, Hans Kellerer, Ulrich Pferschy, and David Pisinger. Approximation algorithms for knapsack problems with cardinality constraints, January 25 1997.

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

[3] G. V. Gens and E. V. Levner. Fast approximation algorithm for job sequencing with deadlines. *Discrete Applied Mathematics*, 3:313–318, 1981.

[4] 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.

[5] Michael Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Prentice-Hall, Inc., second edition, 2002.

[6] Michael H. Rothkopf and Stephen A. Smith. There are no undiscovered priority index sequencing rules for minimizing total delay costs. *Operations Research*, 32(2):451–456, 1984.

[7] Sartaj Sahni. Approximate algorithms for the 0/1 knapsack problem. *Journal of the ACM*, 22(1):115–124, 1975.