LP-BASED APPROXIMATION ALGORITHMS FOR SCHEDULING AND INVENTORY MANAGEMENT PROBLEMS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Maurice Yuk Leung Cheung

May 2012

LP-BASED APPROXIMATION ALGORITHMS FOR SCHEDULING AND INVENTORY MANAGEMENT PROBLEMS

Maurice Yuk Leung Cheung, Ph.D.

Cornell University 2012

There are two fundamental approaches for using linear programming in designing approximation algorithms: LP-rounding and the primal-dual method. In this thesis, we develop LP-based approximation algorithms for several scheduling and inventory management problems, using both LP-rounding and the primal-dual method.

In machine scheduling, we consider a general class of single-machine scheduling problem of minimizing the total cost summing over all jobs, and the only requirement on the cost function of each job is that it is non-negative and non-decreasing. Using the primal-dual method, we give a simple algorithm for this problem that is guaranteed to return a solution that costs at most twice the optimal. To obtain this result, we add an exponential number of valid inequalities to strengthen the natural LP-relaxation, then design a primal-dual method that works on this exponential-sized LP. We then show how to modify our algorithm for scheduling problems with machine breakdown.

In inventory management, we consider several generalization of the classical Joint Replenishment Problem (JRP): the tree JRP and the cardinality JRP. Using the LP-rounding technique, we give novel algorithms for the tree JRP and cardinality JRP that are guaranteed to generate a solution with cost within a factor of three and five, respectively, of the optimal.

BIOGRAPHICAL SKETCH

Maurice Cheung was born on May 25, 1983 in Hong Kong, and moved to Toronto at the age of twelve. He received a B. Math in Combinatorics and Optimization with Minor in Computer Science from University of Waterloo in 2007. In May 2012, he is expected to receive his Ph.D. in Operations Research from Cornell University. After completing his Ph.D. he will begin working at Innovative Scheduling.

To my parents

ACKNOWLEDGEMENTS

First and for most, I would like to thank my advisor David Shmoys, for all the guidance and advice he has given me throughout my graduate studies, for always being patient and having confidence in me even through the tough times, and for always looking out for my best interest. It is a pleasure working with you!

I would also like to thank the faculty members in the School of ORIE at Cornell, for their excellence in teaching , which reaffirm my interest in the area of optimization and algorithms. In particular, thanks to Adrian Lewis and Bobby Kleinberg for serving on my thesis committee, and David Williamson for many interesting and helpful discussions on approximation algorithms in general and implementation of primal-dual algorithms.

Thank you to the faculties and students at MIT Sloan School of Management, where I spent the academic year 2010-2011 as a visiting student. I had a great time in Boston, and many of the results in this thesis is completed while visiting MIT. In particular, the results in chapter 3 is joint work with Adam Elmachtoub and Retsef Levi from MIT.

Going back further in time, I would like to thank my undergraduate advisors at University of Waterloo: Chaitanya Swamy, Jochen Konemann, Romy Shioda and Levent Tuncel, who introduce me to the area of optimization and simulated my interest in doing research.

Life at Cornell would have been less interesting without all my friends. Thanks to all the fellow students in the OR department for their support and making my time at Cornell a wonderful experience. Special thanks to my roommates and officemates throughout the years: Jiawei Qian, Sophia Liu, Chao Ding, Tim Carnes, Gwen Spencer, Yuemeng Sun, Jialie Chen, Eric Guo, Yi Shen

and Kunlaya Soiaporn.

Finally, I thank my family for their unconditional love, support and constant encouragement throughout the years. I am grateful to my parents for instilling the love of learning and knowledge in me, and for encouraging me to pursue my interest, and my sister Denise for being a constant source of support and friendship. I definitely would not be where I am today without the support of my family.

This work is supported by NSF grants CCF-0832782, CCF-1017688 and NSERC award PGS-D3 358528 .

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Approximation Algorithms

Ever since the theory of NP-completeness was developed, it has been shown that most interesting optimization problems, including many of those arising in scheduling and inventory management, are NP-hard. Under the widely believed conjecture that $P \neq NP$, there is no efficient algorithms that find optimal solutions for such NP-hard problems, where we follow the convention that an efficient algorithm is one that runs in time bounded by a polynomial in its input size. In other words, given an NP-hard problem, it is impossible to have algorithms that simultaneously (1) find an optimal solution, (2) in polynomial time, and (3) for any instance. At least one of these requirements must be relaxed.

One approach is to relax the requirement that the algorithm needs to be able to solve any instance of the problem, and focus on solving special cases of the given problem. For example, although the Independent Set problem is an NP-hard problem, the problem becomes polynomially solvable if the input graph is a tree, and a simple greedy algorithm solves the problem (Ch 10, [35]). However, this approach is useful only if the instances one wishes to solve fall into one of these special cases.

Another approach is to relax the requirement of the algorithm having polynomial running time, and to find optimal solutions by a clever exploration of some set which contains an optimal solution. One example of this approach is the branch-and-bound algorithm for solving Integer Programs. Although the

branch-and-bound algorithm finds an optimal solution for any integer program in theory, one is never certain that the algorithm will terminate in any reasonable amount of time.

In this thesis, we consider the approach of using approximation algorithms, which relax the requirement of finding an optimal solution, and instead are guaranteed to find a good solution quickly for all instances of the given problem. This is by far the most common approach in dealing with NP-hard problems. We assume that there is some objective function that maps each possible solution to some nonnegative value, and an *optimal solution* to a given optimization problem is one that either minimizes or maximizes the value of this objective function. The value of an optimal solution is referred to as the *optimal value*. We define an α-approximation algorithm of a given problem as a polynomial-time algorithm that produces a solution whose value is within a factor of α of the optimal value for all instances of the given problem. We say α is the *performance guarantee* of the algorithm, which is also referred to as the *approximation ratio* of the algorithm. The problems we considered in this thesis are minimization problems and hence $\alpha > 1$.

In 1966, Graham [26] analyzed a simple algorithm for the problem of scheduling jobs on identical parallel machine with the objective of minimizing the makespan (the time it takes for all jobs to finish processing), which is one of the first papers published on the analysis of an approximation algorithm. Other early work in approximation algorithms includes Vizing's algorithm for the edge coloring problem [50] and Erdős's algorithm on the maximum cut problem [19]. The term "approximation algorithm" was coined by Johnson [31] in 1974. Since then, tremendous progress has been made in the field of approx-

imation algorithm. The recent book by Williamson and Shmoys [52] provides a very good overview of the subject area as it stands today and summarizes many common techniques used in the design of approximation algorithms, including the LP-based methods used in this thesis. The books by Vazirani[48] and Hochbaum [30] are also good references on approximation algorithms.

In designing approximation algorithms, a key issue is how to show a solution is close to optimal when the optimal value itself is unknown. Hence one need to obtain good lower bounds (for minimization problems) or upper bounds (for maximization problems) for the optimal value. For minimization problems, if one can show that an algorithm runs in polynomial time and always produces a solution whose value is within a factor of α of a lower bound, this would imply this is an α-approximation algorithm.

One approach for producing such lower bounds is to make arguments about the optimal value using the structure of the given problem. To illustrate this, consider the problem of scheduling jobs in parallel machines with the objective of minimizing makespan. Suppose that there are n jobs to processed, and m identical machines available. Each job $j = 1, ..., n$ must be processed on one of these machines for p_j units of time without interruption. Each machine can process at most one job at a time. The goal is to complete all the jobs as soon as possible. Graham [26] considered the following algorithm, known as *list scheduling* (LS): given a list of jobs in some arbitrary order, schedule the next job on the list whenever a machine becomes available. Graham showed that this is a $(2-1/m)$-approximation algorithm for minimizing the makespan.

Let us denote the completion time of job j to be C_j in list scheduling. Let $C^{LS}_{max} = \max_{j=1,...,n} C_j$ denote the makespan of list scheduling, and C^*_{max} denote

the makespan of an optimal schedule. Focus on the last job that completes processing in the list scheduling algorithm; call it job k. Now we obtain two lower bounds on C^*_{max} as follows. First, since job k must be processed on some machine, we have:

$$C^*_{max} \geq p_k$$

Second, since the total amount of processing is $\sum_{j=1}^{n} p_j$ and there are m machines available, the machine with the maximum load is at least $(1/m)\sum_{j=1}^{n} p_j$, and because the completion time of the last job on any machine equals to the load of that machine, we have:

$$C^*_{max} \geq (1/m)\sum_{j=1}^{n} p_j$$

To analyze the performance guarantee of the list scheduling algorithm, we express C^{LS}_{max} in terms of the two lower bounds derived above. By the choice of k, $C^{LS}_{max} = C_k$. Let S_k denotes the starting time of job k. Note that $C_k = S_k + p_k$. Observe that by properties of the list scheduling, no machine is idle before S_k, since otherwise job k would have started sooner. On the other hand S_k is upper-bounded by $(1/m)\sum_{j\neq k} p_j$ since at least one of the machine must be available at that time. Putting it all together, we have:

$$\begin{aligned} C^{LS}_{max} = S_k + p_k &\leq (1/m)\sum_{j\neq k} p_j + p_k \\ &= (1/m)\sum_{j=1}^{n} p_j + (1-1/m)p_k \\ &\leq C^*_{max} + (1-1/m)C^*_{max} = (2-1/m)C^*_{max} \end{aligned}$$

We see that the key to the analysis above is to use structural properties of feasible schedule to establish the two lower bounds on the makespan, which holds true for the any schedule. Often, bounds of this type are not specific to

the problem instance in hand, but provides a worst-case guarantee that holds true for all instances. Hence this would not provide a better performance guarantee even if we achieve a solution that happens to be much closer to optimality for a given instance . On the other hand, LP-based approximation algorithms produce problem-specific lower bounds that can be much better than the worst-case performance guarantee.

1.2 LP Based Approximation Algorithms

Linear programming provides a unified way of generating bounds for the optimal value of optimization problems. Most discrete optimization problems can be modeled as an integer program (IP). Hence to find an optimal solution, one can solve the corresponding integer program which has the same optimal value as the given problem. Although there are commercial packages for solving integer programs that are able to solve large integer programs that arise in some application areas in a reasonable amount of time, integer programs cannot be generally solved in polynomial time. Instead, we relax the integrality constraints on the variables, and allow them to take on fractional values, thereby expanding the set of feasible solutions. This is called the *LP-Relaxation* of the IP. There are several polynomial-time algorithms for solving linear programs such as the ellipsoid method or interior point methods [12]. Moreover, the optimal value of the LP-relaxation will be a lower bound (for minimization problems) to the optimal value of the IP, since any solution of the IP is a feasible solution to the LP-relaxation. Hence we have the following general approach for producing an approximation algorithm for a given minimization problem: formulate an IP and its corresponding LP-relaxation, and produce an integer solution with

value no more than a factor of α of the optimal value of the LP-relaxation.

In this thesis, we consider two types of LP based approximation algorithms: LP-rounding algorithms and primal-dual algorithms. LP rounding algorithms explicitly solve the LP relaxation and use the optimal fractional solution to obtain an integer solution by a rounding procedure that ensures that the cost does not increase much. The performance guarantee of the algorithm is established by comparing the cost of the integral solution and optimal LP solution. On the other hand, primal-dual algorithms exploit the theory of LP duality and uses the LP-relaxation implicitly as a guide for constructing an integer solution and a feasible solution to the dual linear program of the LP relaxation. The performance guarantee is established by comparing the integral solution and the solution of the dual linear program, which provides a lower bound to the optimal value.

One advantage of an LP-based approximation algorithm is that we establish the approximation ratio for each specific instance of the problem by comparing the value of the integer solution to optimal value of the LP relaxation (or the value of solution to the dual LP for primal-dual algorithm), which can be much better than the worst-case performance guarantee of the approximation algorithm. However, there is also an inherent limitation to an LP-based approximation algorithm based on how close the optimal value of the LP relaxation is to the optimal value of the IP. Let $OPT(I)$ denote the optimal value of instances I and $OPT_f(I)$ be the optimal value of the the LP relaxation. We define the *integrality gap of an LP relaxation* as $\sup_I OPT(I)/OPT_f(I)$. In words, it is the worst-case ratio between the optimal value of the IP and the optimal value of the LP relaxation, over all instances of the problem. An LP-based algorithm that

compares the cost of the integral solution constructed with an optimal fractional solution (or feasible solution of the dual LP) cannot achieve a performance guarantee better than the integrality gap. Hence in designing LP-based approximation algorithm, it is important to work with an LP relaxation that has a small integrality gap. One approach in dealing with an LP relaxation that has a large integrality gap is to strengthen it by adding *valid inequalities* that reduce the set of feasible solutions to the LP relaxation, but does not change the feasible region of the IP.

1.2.1 LP-Rounding Algorithms

LP-rounding is perhaps the most natural approach for LP-based approximation algorithm. Starting from the optimal solution of the LP relaxation, construct an integer solution by rounding while ensuring that the cost of the integer solution is close to the LP optimum. There are two varieties of LP-rounding algorithms: deterministic rounding and randomize rounding, where the value of a variable is set according to some probability distribution.

Early deterministic LP-rounding algorithms include the set cover algorithm due to Hochbaum [29] and the bin-packing algorithm due to Karmarkar and Karp [34]. Raghavan and Thompson were the first to introduce the idea of randomized rounding of LP relaxation [43]. For many fundamental problems in combinatorial optimization such as the Uncapacitated Facility Location problem, Maximum Satisfiability problem and the Prize-Collecting Steiner Tree problem, there are several LP-rounding algorithms using both techniques, see [52] and the references within.

Finally, we mention some previous work for various scheduling problems that uses LP rounding. There is a long line of work for the objective of minimizing the weighted sum of completion time of jobs with release dates and/or precedence constraint in a variety of machine environments. For problems where jobs have a release date, the seminal paper by Hall, Schulz, Shmoys and Wein [28] gives a 3-approximation algorithm based on deterministic LP-rounding for minimizing the weighted completion time in a single machine (expressed as $1|r_j|\sum w_jC_j$ using the conventional notation for classifying scheduling problems). The same paper also give a $(4-1/m)$-approximation algorithm for the parallel machine setting ($P|r_j|\sum w_jC_j$) and a $16/3$-approximation algorithm for unrelated machines ($R|r_j|\sum w_jC_j$). These algorithms are deterministic LP-rounding algorithms which use an LP relaxation with variables C_j to indicate the completion time of job j. Given an optimal LP solution, it schedule jobs in nondecreasing order of C_j^*, the completion time of jobs in the optimal LP solution. Subsequently, algorithms with improved performance guarantee have been obtain using randomized rounding. In particular, it uses a time-indexed formulation and order jobs by their α_j point, where $\alpha_j \in [0,1]$ is chosen from an appropriately chosen probability distribution. Using this idea, Goemans gives a 1.6853-approximation algorithm for $1|r_j|\sum w_jC_j$ [23], and Schulz and Skutella give a 2-approximation for both $P|r_j|\sum w_jC_j$ and $R|r_j|\sum w_jC_j$. For the setting of unrelated machines ($R|r_j|\sum w_jC_j$), this is still the best approximation ratio. On the other hand, a polynomial time approximation scheme (PTAS), which can produce a $(1+\epsilon)$ approximate solution for every fixed $\epsilon > 0$, exists for the single machine and parallel machine setting [1].

For the single-machine scheduling problem with precedence constraint ($1|prec|\sum w_jC_j$), scheduling jobs in the same order as C_j^* in optimal LP solu-

tion gives a 2-approximation algorithm [28]. Assuming a variant of the Unique Games conjecture, this is the best possible result [8]. LP-rounding techniques are also used in approximation algorithms for minimizing the makespan in the unrelated machine setting [45], and many stochastic scheduling problems [40].

1.2.2 The Primal-Dual Method for Approximation Algorithm

One weakness of LP-rounding is that they require solving an LP. Although LPs can be solved in polynomial time, it typically takes longer than running a purely combinatorial algorithm. On the other hand, primal-dual algorithms obviate the need to actually solve an LP; instead it simultaneously constructs a feasible integer solution and a feasible solution to the dual LP, which, by LP duality theory, provides a lower bound to the optimal value.

Let's briefly review the theory of LP duality. For concreteness, consider the following primal linear program:

$$\begin{aligned} \text{minimize } & \sum_{j=1}^{n} c_j x_j & & (1.1) \\ \text{subject to } & \sum_{j=1}^{n} a_{ij} x_j \geq b_i, & i = 1, ..., m; & \\ & x_j \geq 0, & j = 1, ..., n. & \end{aligned}$$

For every linear program, there is a associated dual linear program. The dual

linear program of (1.1) is as follows:

$$\begin{aligned} \text{maximize } & \sum_{i=1}^{m} b_i y_i & & (1.2) \\ \text{subject to } & \sum_{i=1}^{m} a_{ij} y_i \leq c_j, & j = 1, ..., n; \\ & y_i \geq 0 & i = 1, ..., m. \end{aligned}$$

The weak duality theorem states that the dual objective value for every feasible solution y of (1.2) is a lower bound to the optimal value of (1.1).

Theorem 1 (Weak duality)**.** *If $x = (x_1, ..., x_n)$ is a feasible solution to (1.1) and $y = (y_1, ..., y_m)$ is a feasible solution to (1.2), then $\sum_{j=1}^{n} c_j x_j \geq \sum_{i=1}^{m} b_i y_i$.*

The strong duality theorem is the central result of the theory of linear programming duality

Theorem 2 (Strong duality)**.** *If the primal linear program (1.1) and the dual linear program (1.2) both have feasible solutions, then (1.1) has a optimal solution $x^* = (x_1^*, ..., x_n^*)$ and (1.2) has a optimal solution $y^* = (y_1^*, ..., y_m^*)$, and $\sum_{j=1}^{n} c_j x_j^* = \sum_{i=1}^{m} b_i y_i^*$.*

Let $x^* = (x_1^*, ..., x_n^*)$ be an optimal solution of (1.1) and $y^* = (y_1^*, ..., y_m^*)$ be an optimal solution of (1.2). Since y^* is dual feasible and x_j^*'s are nonnegative:

$$\sum_{j=1}^{n} c_j x_j^* \geq \sum_{j=1}^{n} (\sum_{i=1}^{m} a_{ij} y_i^*) x_j^* \tag{1.3}$$

Similarly, since x^* is primal feasible and y_i^*'s are nonnegative:

$$\sum_{i=1}^{m} (\sum_{j=1}^{n} a_{ij} x_j^*) y_i^* \geq \sum_{i=1}^{m} b_i y_i^*. \tag{1.4}$$

On the other hand, by strong duality $\sum_{j=1}^{n} c_j x_j^* = \sum_{i=1}^{m} b_i y_i^*$, which holds if and only if both (1.3) and (1.4) are satisfied at equality. This, in turn, is equivalent to following conditions:

Primal complementary slackness conditions:

$$\sum_{i=1}^{m} a_{ij} y_i^* = c_j \text{ or } x_j^* = 0, j = 1, ..., n \tag{1.5}$$

Dual complementary slackness conditions:

$$\sum_{j=1}^{n} a_{ij} x_j^* = b_i \text{ or } y_i^* = 0, i = 1, ..., m \tag{1.6}$$

Hence by strong duality, we have the following theorem

Theorem 3 (Complementary slackness)**.** *Let x and y be feasible solutions to (1.1) and (1.2) respectively. Then x and y are optimal solutions to their respective linear programs if and only if they obey the complementary slackness conditions (1.5) and (1.6).*

The primal-dual method for exact algorithms is driven by complementary slackness conditions above. Many optimization algorithms for classical problems in combinatorial optimization that are polynomial-time solvable can be cast in terms of the primal-dual methods, including Dijkstra's shortest path algorithm, Kuhn's Hungarian Method for the assignment problem and Ford-Fulkerson's algorithm for maximum flow. To use the primal-dual method for approximation algorithms, we work with an LP-relaxation of an IP of the given problem and the dual LP of the LP-relaxation. The goal is to find a near-optimal integer solution, but unless there is an optimal solution of the LP-relaxation that is integral, we have no hope of finding a feasible integer primal solution that satisfies the complementary slackness conditions. It turns out that an approximation algorithm can be driven by a suitable relaxation of the complementary slackness conditions.

Relaxed Primal complementary slackness conditions:

$$\text{Let } \alpha \geq 1: \text{ either } c_j/\alpha \leq \sum_{i=1}^{m} a_{ij} y_i^* \leq c_j \text{ or } x_j^* = 0, j = 1, ..., n \tag{1.7}$$

Relaxed Dual complementary slackness conditions:

$$\text{Let } \beta \geq 1: \text{ either } b_i \leq \sum_{j=1}^{n} a_{ij} x_j^* \leq \beta \cdot b_i \text{ or } y_i^* = 0, i = 1, ..., m \tag{1.8}$$

If we are able to satisfy the relaxed complementary slackness conditions above, then the solution we produced is within a factor of $\alpha \cdot \beta$ of the optimal:

Lemma 1 (Ch. 15, [48]). *If x and y are feasible solution to the primal and dual LPs satisfying conditions (1.7) and (1.8) then $\sum_{j=1}^{n} c_j x_j^* \leq \alpha\beta \cdot \sum_{i=1}^{m} b_i y_i^*$*

A typical approach of using the primal-dual method starts with a dual feasible solution (usually $y = 0$ works) and a primal integer solution that satisfies the primal complementary slackness conditions, but is infeasible. We then improve the dual solution until more dual constraints becomes tight, and we augment the primal solution correspondingly. We iterate until the primal solution become feasible.

The first use of the primal-dual method for approximation algorithms were developed by Bar-Yeuda and Even for the weighted vertex cover [10] and Chvátal for set cover [16]. Subsequently, this approach has been applied to many other settings, such as the classic result for Agrawal, Klein & Ravi [2] and Goemans & Wiliamson [24] on network design problems, Levi, Roundy & Shmoys on inventory management problems. The primal-dual method by Carnes & Shmoys [13] on capacitated covering problem serves as a starting point to our result in Chapter 2.

1.3 Summary of Results

In Chapter 2, we consider the following single-machine scheduling problem, which is often denoted $1||\sum f_j$; we are given n jobs, where each job j has an integral processing time p_j, and there is a nonnegative cost function $f_j(C_j)$ that specifies the cost of finishing j at time C_j. In addition to being nonnegative, the only restriction on the cost function $f_j(C_j)$ is that it is a nondecreasing function of C_j; each job is to be scheduled on a single machine, and the objective is to minimize $\sum_{j=1}^{n} f_j(C_j)$. In a recent paper, Bansal & Pruhs [9] gave the first constant approximation algorithm for this problem; more precisely, they gave a 16-approximation algorithm. We improve on this result: we give a primal-dual pseudo-polynomial-time algorithm that finds a solution of cost at most twice the optimal cost, and then show how this can be extended to yield, for any $\epsilon > 0$, a $(2+\epsilon)$-approximation algorithm for this problem. The framework that we provide for this algorithm is sufficiently flexible to give several related results with little extra effort. For example, one model that has received recent attention is the problem of scheduling a single machine that has non-availability periods (where a job may be resumed without penalty if its processing is interrupted by such a period). We give, for any $\epsilon > 0$, a $(2+\epsilon)$-approximation algorithm for this model, or even more generally, for the setting in which the machine's speed can vary over time arbitrarily – no constant-factor approximation algorithm was known previously for this model.

In Chapter 3, we consider the submodular Joint Replenishment Problem (JRP). This is a problem in inventory management that generalizes the classical JRP. The input of the problem is as follows: there are N items that are needed over a planning horizon of T periods, and deterministic demand $d_{it} \geq 0$ for

each item $i = 1, ..., N$ and time period $t = 1, ..., T$. Each demand point d_{it} with positive demand needs to be serve by an order of item i in period s with $s \leq t$. There are two components to the total cost incurred, the ordering cost and holding cost.

To model the ordering cost, let $f(S)$ denote the cost of ordering the set of item types S in any given period. The only requirement is that f is non-negative monotone submodular function, meaning for every set $S_1, S_2 \subseteq \{1, \ldots, N\}, f(S_1) + f(S_2) \geq f(S_1 \cup S_2) + f(S_1 \cap S_2)$. Hence the ordering cost is capable of capturing the full effects of economies of scales. In constrast, the ordering cost of the classical JRP is rather restrictive: there is a joint setup cost if any item is order, in addition to individual oredering cost associated with each item.

The holding cost from period s to t for the demand of item i in period t is denoted $H^i_{st} := d_{it} \cdot h^i_{st}$, where h^i_{st} is the per-unit holding cost of item i from period s to t. We assume h^i_{st} is nonnegative, and for each (i, t) is a non-increasing function of s.

We first study a linear programming relaxation for the submodular JRP and prove some structural properties about an optimal solution of the linear program, which is useful in designing an LP-based approximation algorithm for this problem. We then give approximation algorithm to several special cases of the submodular JRP: the JRP with Tree Ordering Cost (Tree JRP), and the JRP with Concave Ordering Cost (Cardinality JRP).

In the Tree JRP, the ordering cost is specified by a rooted tree. Specifically, there is a cost K_j associated for each node in the tree, and each item type i is a

leaf in the tree. Let j be a node in the tree and $path(j)$ denote the unique path from node j to the root of the tree. The cost of ordering a set of item type S is defined to be $\sum_{j \ni path(i) | i \in S} K_j$, i.e., the cost of all nodes that are in the path from some item i to the root, where item i is in set S. Note that this contains the classical JRP as a special case since the holding cost of the classical JRP can viewed as a tree with a root node with cost K_0, the joint ordering cost, connected to N leaves, one for each item i with cost K_i, the cost of ordering item i. We extend the ideas in [38] to give a 3-approximation algorithm for this problem via LP-Rounding.

In the cardinality JRP, the ordering cost is given by a concave function based on the cardinality of set of item type being ordered. Formally, let $g(k)$ be a concave function which denote the cost of ordering k item types in any given period. Then the cost of ordering the item type S in any given period is $g(|S|)$. We give a 5-approximation algorithm for this problem via a LP-Rounding algorithm that uses a novel charging scheme to bound the ordering cost.

CHAPTER 2

PRIMAL-DUAL APPROXIMATION ALGORITHM FOR MIN-SUM SINGLE-MACHINE SCHEDULING

2.1 Introduction

We consider the following general scheduling problem, which is denoted as $1||\sum f_j$ in the notation of scheduling problems formulated by Graham, Lawler, Lenstra, & Rinnooy Kan [25]: we are given n jobs to schedule on a single machine, where each job $j = 1, \ldots, n$ has a positive integral processing time p_j, and there is a nonnegative cost function $f_j(C_j)$ that specifies the cost of finishing j at time C_j. The only restriction on the cost function $f_j(C_j)$ is that it is a nondecreasing function of C_j; the objective is to minimize $\sum_{j=1}^{n} f_j(C_j)$. In a recent paper, Bansal & Pruhs [9] gave the first constant approximation algorithm for this problem; more precisely, they present a 16-approximation algorithm, that is, a polynomial-time algorithm guaranteed to be within a factor of 16 of the optimum. We improve on this result: we give a primal-dual pseudopolynomial-time algorithm that finds a solution of cost at most twice the optimal cost, and then show how this can be extended to yield, for any $\epsilon > 0$, a $(2 + \epsilon)$-approximation algorithm for this problem. This problem is strongly NP-hard (simply by considering the case of the weighted total tardiness, where $f_j(C_j) = w_j \max_{j=1,\ldots,n}\{0, C_j - d_j\}$ and d_j is a specified due date of job j, $j = 1, \ldots, n$). However, no hardness results are known other than this, and so it is still conceivable (and perhaps likely) that there exists a polynomial approximation scheme for this problem (though by the classic result of Garey & Johnson [22], no fully polynomial approximation scheme exists unless P=NP).

No such scheme is known even for the special case of weighted total tardiness.

Our results are based on the linear programming relaxation of a time-indexed integer programming formulation in which the 0-1 decision variables x_{jt} indicate whether a given job $j = 1, \ldots, n$, completes at time $t = 1, \ldots, T$, where $T = \sum_{j=1}^{n} p_j$; note that, since the cost functions are nondecreasing with time, we can assume, without loss of generality, that the machine is active only throughout the interval $[0, T]$, without any idle periods. For convenience, we will set $f_j(t) = f_j(p_j)$ for each $t = 1, \ldots, p_j$; this is clearly without loss of generality, since in any feasible schedule each job j cannot finish before time p_j, and we will see later that our algorithm ensures that x_{jt} will be set to 0 when $t = 1, \ldots, p_j - 1$. With these time-indexed variables, it is trivial to ensure that each job is scheduled; the only difficulty is to ensure that the machine is not required to process more than one job at a time. To do this, we observe that, for each time $t = 1, \ldots, n$, the jobs completing at time t or later have total processing time at least $T - t + 1$ (by the assumption that the processing times p_j are positive integers); let $D(t) = T - t + 1$. This gives the following integer program:

$$\text{minimize} \quad \sum_{j=1}^{n} \sum_{t=1}^{T} f_j(t) x_{jt} \tag{IP}$$

$$\text{subject to} \quad \sum_{j=1}^{n} \sum_{s=t}^{T} p_j x_{js} \geq D(t), \qquad \text{for each } t = 1, \ldots, T; \tag{2.1}$$

$$\sum_{t=1}^{T} x_{jt} = 1, \qquad \text{for each } j = 1, \ldots, n; \tag{2.2}$$

$$x_{jt} \in \{0, 1\}, \qquad \text{for each } j = 1, \ldots, n,\ t = 1, \ldots, T.$$

We first argue that this a valid formulation of the problem. Clearly, each feasible schedule corresponds to a feasible solution to (IP) of equal objective function value. Conversely, consider any feasible solution, and for each job

$j = 1, \ldots, n$, assign it the due date $d_j = t$ corresponding to $x_{jt} = 1$. If we schedule the jobs in Earliest Due Date (EDD) order, then we claim that each job $j = 1, \ldots, n$, completes by its due date d_j. If we consider the constraint (2.1) in (IP) corresponding to $t = d_j + 1$, then since each job is assigned once, we know that

$$\sum_{j=1}^{n} \sum_{t=1}^{d_j} p_j x_{jt} \leq d_j;$$

in words, the set of jobs with due date at most d_j have total processing time at most d_j. Since each job completes by its due date, and the cost functions $f_j(\cdot)$ are nondecreasing, we have a schedule of cost no more than that of the original feasible solution to (IP).

The formulation (IP) has an unbounded integrality gap: the ratio of the optimal value of (IP) to the optimal value of its linear programming relaxation can be arbitrarily large. We strengthen this formulation by introducing a class of valid inequalities called *knapsack-cover inequalities*. To understand the starting point for our work, consider the special case of this scheduling problem in which all n jobs have a common due date D, and for each job $j = 1, \ldots, n$, the cost function is 0 if the job completes by time D, and is w_j, otherwise. In this case, we select a set of jobs of total size at most D, so as to minimize the total weight of the complementary set (of late jobs). This is equivalent to the minimum-cost (covering) knapsack problem, in which we wish to select a subset of items of total size at least a given threshold, of minimum total cost. Carr, Fleischer, Leung, and Phillips [14] introduced knapsack-cover inequalities for this problem (as a variant of flow-cover inequalities introduced by Padberg, Van Roy, and Wolsey [41]) and gave an LP-rounding 2-approximation algorithm based on this formulation. Our results instead generalize a primal-dual 2-approximation algorithm based on the same formulation, which was given by

Carnes and Shmoys [13]. Knapsack-cover inequalities have subsequently been used to derive approximation algorithms in a variety of other settings, including the work of Bansal & Pruhs [9], Bansal, Buchbinder, & Naor [5, 6], Gupta, Krishnaswamy, Kumar, & Segev [27], Bansal, Gupta, & Krishnaswamy [7], and Pritchard [42].

The idea behind the knapsack-cover inequalities is quite simple. Fix a subset of jobs $A \subseteq \{1, .., n\}$ that contribute towards satisfy the demand $D(t)$ for time t or later; then there is a *residual demand* from the remaining jobs of $D(t, A) := \max\{D(t) - \sum_{j \in A} p_j, 0\}$. Thus, each job $j = 1, \ldots, n$ can make an effective contribution to this residual demand of $p_j(t, A) := \min\{p_j, D(t, A)\}$; that is, given the inclusion of the set A, the effective contribution of job j towards satisfying the residual demand can be at most the residual demand itself. Thus, we have the constraint:

$$\sum_{j \notin A} \sum_{s=t}^{T} p_j(t, A) x_{js} \geq D(t, A)$$

for each $t = 1, ..., T$, and each $A \subseteq \{1, .., n\}$.

Our primal-dual algorithm has two phases: a growing phase and a pruning phase. Throughout the algorithm, we maintain a set A_t for each time $t = 1, \ldots, T$. In each iteration of the growing phase, we choose a dual variable to increase, corresponding to the demand $D(t, A_t)$ that is largest, and increase that dual variable as much as possible. This causes a dual constraint corresponding to some job j to become tight for some time t', and so that we set $x_{jt'} = 1$, and add j to each set A_s with $s \leq t'$. Note that this may result in jobs being assigned to complete at multiple times t; then in the pruning phase we do a "reverse delete" that both ensures that each job is uniquely assigned, and also that the solution is minimal, in the sense that each job passes the test that if it were

deleted, then some demand constraint (3.8) in (IP) would be violated. It will be straightforward to show that the algorithm runs in time polynomial in n and T, which is a pseudopolynomial bound. To convert this algorithm to be polynomial, we adopt an interval-indexed formulation, where we bound the change of cost of any job to be within a factor of $(1 + \epsilon)$ within any interval. This is sufficient to ensure a (weakly) polynomial number of intervals, while degrading the performance guarantee by a factor of $(1 + \epsilon)$, and this yields the desired result. One surprising consequence of this interval-indexed approach, combined with the generality of the objective function structure, is that the same algorithm can be used to obtain the same performance guarantee in the more general setting in which the machine has a time-dependent speed $s(t)$ (where we assume that we can compute the processing capacity of the machine for any time interval $(t_1, t_2]$ in polynomial time); this greatly generalizes and improves upon the randomized e-approximation algorithm of Epstein, Levin, Marchetti-Spaccamela, Megow, Mestre, Skutella, and Stougie [18] for this setting in the special case where the objective function is to minimize $\sum_j w_j C_j$ (though the results in [18] have the advantage of not needing to know the speed function in advance).

The main question left open by this work is whether similar techniques can yield analogous results for the analogous problem $1|r_j, pmtn| \sum f_j$. Bansal and Pruhs gave a $O(\log \log(nT))$-approximation algorithm for this problem, and yet there is no evidence to suggest that there does not exist an approximation algorithm with a constant performance guarantee (or potentially even a polynomial approximation scheme). (Since there is no advantage to preemption if all release dates are equal to 0, it follows that, as for $1|| \sum f_j$, the problem is strongly NP-hard, and hence no fully polynomial approximation scheme exists unless P=NP). One interesting point of contrast is the "bottleneck" or "min-max" ana-

logue of these two problems: $1||f_{\max}$ and $1|r_j, pmtn|f_{\max}$. Both problems are solvable in polynomial time, by variants of the least cost last rule, by results of Lawler [36] and of Baker, Lawler, Lenstra, and Rinnooy Kan [4], respectively.

2.2 A Pseudopolynomial Algorithm for $1||\sum f_j$

We give a primal-dual algorithm that runs in pseudopolynomial time and has a performance guarantee of 2 and is based on the following LP relaxation:

$$\begin{aligned}
&\text{minimize} \quad \sum_{j=1}^{n}\sum_{t=1}^{T} f_j(t)x_{jt} && \text{(P)}\\
&\text{subject to} \quad \sum_{j\notin A}\sum_{s=t}^{T} p_j(t,A)x_{js} \geq D(t,A), \quad \text{for each } t=1,...,T,\ A\subseteq\{1,..,n\}; && \text{(2.3)}\\
&\qquad x_{jt}\geq 0, \qquad\qquad \text{for each } j=1,\ldots,n,\ ,t=1,\ldots,T.
\end{aligned}$$

Notice that the assignment constraints (2.2) are not included in (P). In fact, the following lemma shows that they are redundant, given the knapsack-cover inequalities. This leaves a much more tractable formulation on which to base the design of our primal-dual algorithm, and is one of the reasons that we are able to obtain an improved guarantee.

Lemma 2. *Let x be an integer feasible solution to the linear programming relaxation (P). Then there is an integer feasible solution $\bar{x}$ of no greater cost that also satisfies the assignment constraints (2.2).*

Proof. We first show that each job $k = 1,\ldots,n$ is assigned at least once by x.

To see this, consider the constraints (2.3) with the set $A = \{1, \ldots, n\} - \{k\}$ and $t = 1$. In this case, $D(1, A) = \max\{T - \sum_{j \neq k} p_j, 0\} = p_k$ and hence $p_k(1, A) = p_k$. Hence, constraint (3) yields that $p_k \sum_{s=1}^{T} x_{ks} \geq p_k$, and so $\sum_{s=1}^{T} x_{ks} \geq 1$.

We next show that each job is assigned at most once. We may assume without loss of generality that x is an integer feasible solution for (P) in which $\sum_{j=1}^{n} \sum_{s=1}^{T} x_{js}$ is minimum. Suppose, for a contradiction, that job k is assigned at least twice, at let $t_1 \leq t_2$ be the latest two assignments of job j. (Consider the case $x_{ks} > 1$ as a degenerate case as if job k is assigned multiple times to time s.) Consider the solution in which we decrease x_{kt_1} by 1. We claim that this modified solution $\bar{x}$ is also an integer feasible solution to (P); if we show this, that would complete the proof, since that would contradict the choice of x. Suppose $\bar{x}$ is not feasible for (P), and let (t, A) specify a violated inequality (3). Since the corresponding inequality is satisfied by x, we know that the left-hand side must decrease by resetting x_{kt_1}, and hence $t \leq t_1$. Furthermore, $k \notin A$. We know that $p_k(t, A) < D(t, A)$, since otherwise, the fact that $x_{kt_2} \geq 1$ would imply that this constraint (3) remains satisfied by $\bar{x}$. Hence $p_k(t, A) = p_k$ and so we decrease the left-hand side by p_k as a result of our modification of x. Similarly, if we consider the constraint (3) for (t, A') where $A' = A \cup \{k\}$, we can conclude that

$$\sum_{s=t}^{T} \sum_{j \notin A'} p_j(t, A') x_{js} \leq -p_k + \sum_{s=t}^{T} \sum_{j \notin A} p_j(t, A) \bar{x}_{js} < -p_k + D(t, A);$$

this uses the fact that x and $\bar{x}$ differ only for job k, and that $p_j(t, A') \leq p_j(t, A)$ for each job $j = 1, \ldots, n$ and each time $t = 1, \ldots, T$. But since we already observed that $p_k < D(t, A)$, we know that $D(t, A') = D(t, A) - p_k$. But then the constraint (t, A') is violated by x, which is a contradiction.

Finally, since $\bar{x} \leq x$ component-wise and the objective $f_j(t)$ is non-negative, $\bar{x}$ is a solution of no greater cost than x. $\square$

Observe that essentially the same proof works with fractional solutions; we need only decrease the components of the overassigned job k so that the assignment constraint is satisfied, retaining exactly one total unit of assignment. We restrict attention to integer solutions only for notational convenience.

Taking the dual of (P) gives:

$$\begin{aligned}
&\text{maximize} \quad \sum_{t=1}^{T} \sum_{A} D(t,A) y(t,A) && \text{(D)}\\
&\text{subject to} \quad \sum_{t=1}^{s} \sum_{A: j \notin A} p_j(t,A) y(t,A) \leq f_j(s); \qquad \text{for each } \; j = 1,..,n, \; s = 1,..,T; && \text{(2.4)}\\
&\qquad\qquad y(t,A) \geq 0 \qquad\qquad \text{for each } t = 1,...,T, \; A \subseteq \{1,..,n\}.
\end{aligned}$$

We now give the primal-dual algorithm for the scheduling problem $1||\sum f_j$. The algorithm consists of two phases: a growing phase and a pruning phase.

The growing phase constructs a feasible solution x to (P) over a series of iteration. For each $t = 1, \ldots, T$, we let A_t denote the set of jobs that are set to finish at time t or later by the algorithm, and thus contribute towards satisfying the demand $D(t)$. In each iteration, we set a variable x_{jt} to 1 and add j to A_s for all $s \leq t$. We continue until all demands $D(t)$ are satisfied. Specifically, in the k^{th} iteration, the algorithm select $t^k := \text{argmax}_t D(t, A_t)$, which is the time index that has the largest residual demand with respect to the current partial solution. If there are ties, we choose the *largest* such time index to be t^k (this is not essential to the correctness of the algorithm – only for consistency and efficiency). If $D(t^k, A_{t^k}) = 0$, then we must have $\sum_{j \in A_t} p_j \geq D(t)$ for each $t = 1, \ldots, T$; all demands have been satisfied and the growing phase terminates. Otherwise, we

increase the dual variable $y(t^k, A_{t^k})$ until some dual constraint (2.4) with right-hand side $f_j(t)$ becomes tight. We set $x_{jt} = 1$ and add j to A_s for *all* $s \leq t$ (if j is not yet in A_s). If multiple constraints become tight at the same time, we pick the one with the *largest* time index (and if there are still ties, just pick one of these jobs arbitrarily). However, at the end of the growing phase, we might have too many variables set to 1, thus we proceed to the pruning phase.

The pruning phase is a "reverse delete" procedure that checks each variable x_{jt} that is set to 1, in decreasing order of the iteration k in which that variable was set in the growing phase. We attempt to set x_{jt} back to 0 and correspondingly delete jobs from A_t, provided this does not violate the feasibility of the solution. Specifically, for each variable $x_{jt} = 1$, if j is also in A_{t+1} then we set $x_{jt} = 0$. It is safe to do so, since in this case, there must exist $t' > t$ where $x_{jt'} = 1$, and as we argued in Lemma 2, it is redundant to have x_{jt} also set to 1. Otherwise, if $j \notin A_{t+1}$, we check if $\sum_{j' \in A_s \setminus \{j\}} p_{j'} \geq D(s)$ for each time index s where j has been added to A_s in the same iteration of the growing phase. If so, then j is not needed to satisfy the demand at time s. Hence, we remove j from all such A_s and set $x_{jt} = 0$. We will show that at the end of the pruning phase, each job j has exactly one x_{jt} set to 1. Hence, we set this time t as the *due date* of job j.

Finally, the algorithm outputs a schedule by sequencing the jobs in Earliest Due Date (EDD) order. We give pseudo-code for the algorithm below.

Analysis Throughout the algorithm's execution, we maintain both a solution x along with the sets A_t, for each $t = 1, \ldots, T$; an easy inductive argument shows that the algorithm maintains the following invariant.

Lemma 3. *Throughout the algorithm, $j \in A_s$ if and only if there exists $t \geq s$ such that*

Algorithm 1: Primal-Dual Algorithm for $1||\sum f_j$

```
// Initialization
```
$x, y, k \leftarrow 0;\ A_t = \emptyset,\ (t = 1, \ldots, T)$;
$t^0 := \text{argmax}_t D(t, A_t)$ (break ties by choosing largest time index) ;
```
// Growing phase
```
while $D(t^k, A_{t^k}) > 0$ **do**
  Increase $y_{t^k, A_{t^k}}$ until a dual constraint (2.4) with right hand side $f_j(t)$ becomes tight (break ties by choosing the largest time index);
  $x_{jt} \leftarrow 1$;
  $A_s \leftarrow A_s \cup \{j\}$ for each $s \leq t$;
  $k \leftarrow k + 1$;
  $t^k := \text{argmax}_t D(t, A_t)$ (break ties by choosing largest time index);

```
// Pruning phase
```
Consider $\{(j,t) : x_{jt} = 1\}$ in reverse order in which they are set to 1 ;
if $j \in A_{t+1}$ **then**
  $x_{jt} \leftarrow 0$
else if $\sum_{j' \in A_s \setminus \{j\}} p_{j'} \geq D(s)$ *for all* $s \leq t$ *where* j *is added to* A_s *in the same iteration of growing phase* **then**
  $x_{j,t} \leftarrow 0$;
  $A_s \leftarrow A_s \setminus \{j\}$ for all such s ;

```
// Output schedule
```
for $j \leftarrow 1$ **to** n **do**
  Set due date d_j of job j to time t if $x_{jt} = 1$;
Schedule jobs using EDD rule ;

$x_{jt} = 1$.

Note that this lemma also implies that the sets A_t are nested; i.e., for any two time indices $s < t$, it follows that $A_s \supseteq A_t$. Using the above lemma, we will show that algorithm produces a feasible solution to (P) and (D).

Lemma 4. *The algorithm produces a feasible solution x to (P) that is integral and satisfies the assignment constraints (2.2), as well as a feasible solution y to (D).*

Proof. First note that, by construction, the solution x is integral. The algorithm

starts with the all-zero solution to both (P) and (D), which is feasible for (D) but infeasible for (P). We will now show that at termination, the algorithm obtains a feasible solution for (P) while maintaining dual feasibility.

When the growing phase terminates, all residual demands $D(t, A_t)$ are zero, and hence, $\sum_{j \in A_t} p_j \geq D(t)$ for each $t = 1, \ldots, T$. During the pruning phase, we reset x_{jt} to 0 only if $A_s \setminus \{j\}$ satisfies the demand $D(s)$, for each time index $s \leq t$ in which job j is added to A_s in the same iteration of the growing phase as the variable x_{jt} was set to 1. Hence, we still have $\sum_{j \in A_t} p_j \geq D(t)$ for each $t = 1, \ldots, T$ when the algorithm terminates.

Next, we argue that for each job j there is exactly one t where $x_{jt} = 1$ when the algorithm terminates. Notice that $D(1)$ (the demand at time 1) is T, which is also the sum of processing time of all jobs; hence we must have A_1 be the set of all jobs at the end of the growing phase in order to satisfy $D(1)$. This implies that at this point in the execution of the algorithm, each job has at least some time t for which $x_{jt} = 1$. From the pruning step (in particular, the first *if* statement in the pseudocode), each job j has x_{jt} set to 1 for at most one value t. However, since no job can be deleted from A_1, by Lemma 2, we see that, for each job j, there is still at least one x_{jt} set to 1 at the end of the pruning phase. Combining the two inequalities, we see that each job j has one value t for which $x_{jt} = 1$.

By invoking Lemma 2 for the final solution x, we have that $\sum_{s=t}^{T} \sum_{j=1}^{n} p_j x_{js} \geq D(t)$. Furthermore, x also satisfies the constraint $\sum_{t=1}^{T} x_{jt} = 1$, as argued above. Hence, x is feasible for (IP), which implies the feasibility for (P).

Next we argue that dual feasibility is maintained throughout the algorithm. In iteration k of the growing phase, only the dual variable $y(t^k, A_{t^k})$ is increased.

The increase is stopped as soon as some dual constraint (2.4) with right-hand side $f_j(t)$ becomes tight. At this point, all constraints are still satisfied. Next, we add j to A_s for each $s \leq t$. Notice that by doing so, the left-hand side of all those constraints must remain constant from this point on. Hence, dual feasibility is maintained in the growing phase. Since the dual remains unchanged in the reverse delete step, the algorithm computes a feasible solution to (D). □

Next we argue that given a feasible solution x to (P) that satisfies the assignment constraints (2.2), the EDD schedule is a feasible solution that costs no more than the objective value for x.

Lemma 5. *Given a feasible integral solution to (P) that satisfies the assignment constraint (2), the EDD schedule is a feasible schedule with cost no more than the value of the given primal solution*

Proof. Since each job $j = 1, \ldots, n$ has exactly one x_{jt} set to 1, it follows that $\sum_{j=1}^{n} \sum_{s=1}^{T} p_j x_{js} = T$. Now, taking $A = \emptyset$ from constraints (2.3), we have that $\sum_{j=1}^{n} \sum_{s=t}^{T} p_j x_{js} \geq D(t) = T - t + 1$. Hence, $\sum_{j=1}^{n} \sum_{s=1}^{t-1} p_j x_{js} \leq t - 1$.

This ensures that the sum of processing assigned to finish before time t is no greater than the machine's capacity for job processing up to this time (which is $t - 1$). Hence, we obtain a feasible schedule by the EDD rule applied to the instance in which, for each job $j = 1, \ldots, n$, we set its due date $d_j = t$, where t is the unique time such that $x_{jt} = 1$. As a corollary, this also shows $x_{jt} = 0$, for $t = 1, \ldots, p_j$. Finally, this schedule costs no more than the optimal value of (P), since each job $j = 1, \ldots, n$ finishes by d_j, and each function $f_j(t)$ is nondecreasing in t. □

Next we analyze the cost of the schedule returned by the algorithm. Given the above lemma, it suffices to show that the cost of the primal solution is no more than twice the cost of the dual solution; the weak duality theorem of linear programming then implies that our algorithm has a performance guarantee of 2.

We first introduce some notation used in the analysis. Given the final solution $\bar{x}$ returned by the algorithm, define $\bar{J}_t := \{j : \bar{x}_{jt} = 1\}$, and $\bar{A}_t := \{j : \exists \bar{x}_{jt'} = 1, t' \geq t\}$. In words, $\bar{A}_t$ is the set of jobs that contribute towards satisfying the demand at time t in the final solution; hence, we say that j *covers* t if $j \in \bar{A}_t$. Let x^k be the partial solution of (P) at the beginning of the k^{th} iteration of the growing phase. We define J_t^k and A_t^k analogously with respect to x^k. Next we prove the key lemma in our analysis.

Lemma 6. $\sum_{s=t}^{T} \sum_{j \in \bar{J}_s \setminus A} p_j(s, A) < 2D(t, A)$, *for each* (t, A) *where* $y(t, A) > 0$.

Proof. Recall that the algorithm increases only one dual variable in each iteration of the growing phase. Suppose that $y(t, A)$ is the variable chosen in iteration k, i.e., $t = t^k$. Using the notation introduced above, we can write $y(t, A)$ as $y(t^k, A_{t_k}^k)$. However, for notational convenience, we shall denote the set $A_{t_k}^k$ as A^k. Then the lemma can be restated as $\sum_{j \in \bar{A}_{t^k} \setminus A^k} p_j(t^k, A^k) < 2D(t^k, A^k)$. We can interpret the set on the left-hand side as the jobs that cover the demand of t^k that are added to the solution after the start of iteration k.

First, suppose time t^k is *critical* with respect to $\bar{x}$ and A^k, meaning there exists some job ℓ in $\bar{A}_{t^k} \setminus A^k$ such that ℓ cannot be deleted from $\bar{A}_{t^k}$ for $\bar{x}$ to remain feasible. This can be expressed as $\sum_{j \in \bar{A}_{t^k} \setminus (A^k \cup \ell)} p_j < D(t^k, A^k)$. Notice that each of these jobs in the above summation must have processing time less than

$D(t^k, A^k)$, and thus by definition, $p_j = p_j(t^k, A^k)$. Hence,

$$\sum_{j \in \bar{A}_{t^k} \setminus (A^k \cup \ell)} p_j(t^k, A^k) < D(t^k, A^k).$$

Also, $p_\ell(t^k, A^k_t) \leq D(t^k, A^k)$, by definition. Adding these two together gives

$$\sum_{j \in \bar{A}_{t^k} \setminus A^k} p_j(t^k, A^k) < 2D(t^k, A^k).$$

Hence, if t^k is critical we have the desired result. Now we will argue by contradiction that this must be the case. Suppose otherwise; then for every job ℓ in $\bar{A}_{t^k} \setminus A^k$, we have that

$$\sum_{j \in \bar{A}_{t^k} \setminus (A^k \cup \ell)} p_j \geq D(t^k, A^k).$$

We first argue that there must exist some time t_ℓ that is critical with respect to $\bar{x}$ and $A^k_{t_\ell}$ because of job ℓ; i.e.,

$$\sum_{j \in \bar{A}_{t_\ell} \setminus (A^k_{t_\ell} \cup \ell)} p_j < D(t_\ell, A^k_{t_\ell}). \tag{2.5}$$

Suppose not; from the definition of $D(t_\ell, A^k_{t_\ell})$, we must have that $\sum_{j \in \bar{A}_{t_\ell} \setminus (A^k_{t_\ell} \cup \ell)} p_j + \sum_{j \in A^k_{t_\ell}} p_j \geq D(t_\ell)$ for each time t_ℓ that job ℓ covers. This, combined with the fact that ℓ is considered in the pruning phase before any of the jobs in $A^k_{t_\ell}$ (since ℓ is added after the start of iteration k), implies that ℓ should have been deleted in the pruning phase, which is a contradiction.

Hence, such a time t_ℓ must exist for each job ℓ in $\bar{A}_{t^k} \setminus A^k$. If there are multiple time indices such that (2.5) holds for job ℓ, let t_ℓ be the earliest such time. Consider the following two cases:

Case 1 Suppose there exists some job ℓ with $t_\ell < t^k$. Notice that each job that covers t^k also covers t_ℓ. Hence, the set of jobs in the final solution that covers t_ℓ

added after the start of iteration k of the growing phase is a superset of those that covers t^k and added after the start of iteration k, i.e., $\bar{A}_{t_\ell} \setminus (A^k_{t_\ell} \cup \ell) \supseteq \bar{A}_{t^k} \setminus (A^k \cup \ell)$ and we have that

$$\begin{aligned} \sum_{j \in \bar{A}_{t_\ell} \setminus (A^k_{t_\ell} \cup \ell)} p_j &\geq \sum_{j \in \bar{A}_{t^k} \setminus (A^k \cup \ell)} p_j \\ &\geq D(t^k, A^k) \\ &\geq D(t_\ell, A^k_{t_\ell}), \end{aligned}$$

where the second inequality follows from our assumption that t^k is not critical, and the last inequality follows from the definition of t^k having the largest residual demand. This gives a contradiction to (2.5); thus Case 1 is impossible.

Case 2 Otherwise, all $t_\ell > t^k$. Pick ℓ so that t_ℓ is the earliest among all jobs in $\bar{A}_{t^k} \setminus A^k$. Notice that each job that covers t_ℓ must cover t^k as well. However, by the assumption that t^k is not critical and the choice of t_ℓ, the set of jobs that covers t_ℓ added after the start of iteration k of the growing phase is the same as those that cover t^k and are added after the start of iteration k, i.e., $\bar{A}_t \setminus (A^k_t \cup \ell) = \bar{A}_{t^k} \setminus (A^k \cup \ell)$. Then we can derive a contradiction similar to Case 1:

$$\begin{aligned} \sum_{j \in \bar{A}_{t_\ell} \setminus (A^k_{t_\ell} \cup \ell)} p_j &= \sum_{j \in \bar{A}_{t^k} \setminus (A^k \cup \ell)} p_j \\ &\geq D(t^k, A^k) \\ &> D(t_\ell, A^k_{t_\ell}). \end{aligned}$$

This gives a contradiction to (2.5); thus Case 2 is also impossible. Combining the two cases, we see that t^k must be critical with respect to $\bar{x}$, giving us the desired result. □

Now we can show our main theorem.

Theorem 4. *The primal-dual algorithm produces a schedule for* $1||\sum f_j$ *with cost at most twice the optimum.*

Proof. It suffices to show that the cost of the schedule is no more than twice the dual objective value. The cost of our solution is denoted by $\sum_{t=1}^{T}\sum_{j\in\bar{J}_t} f_j(t)$. We have that

$$\begin{aligned}\sum_{t=1}^{T}\sum_{j\in\bar{J}_t} f_j(t) &= \sum_{t=1}^{T}\sum_{j\in\bar{J}_t}\sum_{s=1}^{t}\sum_{A:j\notin A} p_j(s,A)y_{sA} \\ &= \sum_{s=1}^{T}\sum_{A} y_{sA}(\sum_{t=s}^{T}\sum_{j\in\bar{J}_t\setminus A} p_j(s,A))\end{aligned}$$

The first line is true because we set $x_{jt} = 1$ only if the dual constraint is tight, and the second line is by interchanging the order of summations and using the relation $s \leq t$. Now, from Lemma 6 we know that $\sum_{t=s}^{T}\sum_{j\in\bar{J}_t\setminus A} p_j(s,A) < 2D(s,A)$. Hence it follows that

$$\sum_{s=1}^{T}\sum_{A} y_{sA}(\sum_{t=s}^{T}\sum_{j\in\bar{J}_t\setminus A} p_j(s,A)) \;<\; \sum_{s=1}^{T}\sum_{A} 2D(s,A)y_{sA},$$

where the right-hand side is twice the dual objective. The result now follows, since the dual objective gives a lower bound of the value of the optimal schedule. □

2.3 A $(2+\epsilon)$-Approximation Algorithm

We now give a polynomial-time $(2+\epsilon)$-approximation algorithm for $1||\sum f_j$. This is achieved by simplifying the input via rounding in a fairly standard fashion, and then running the primal-dual algorithm on the LP relaxation of the

simplified input, which only has a polynomial number of interval-indexed variables. A similar approach was employed in the work of Bansal & Pruhs [9].

Fix a constant $\epsilon > 0$. We start by constructing n partitions of the time indices $\{1, \ldots, T\}$, one partition for each job, according to its cost function. Focus on some job j. First, the set of time indices $I_j^0 = \{t : f_j(t) = 0\}$ are those of *class* 0; class 1 is the set of indices $I_j^1 = \{t : 0 < f_j(t) \leq 1\}$; finally, class $k = 2, 3, \ldots$ is the set of indices $I_j^k = \{t : (1+\epsilon)^{k-2} < f_j(t) \leq (1+\epsilon)^{k-1}\}$. (We can bound the number of classes for job j by $1 + \log_{1+\epsilon} f_j(T)$.) Let ℓ_j^k denote the minimum element in I_j^k (if the set is non-empty), and let $\mathcal{T}_j$ be the set of all left endpoints ℓ_j^k. Finally, let $\mathcal{T} = \cup_{j=1}^n \mathcal{T}_j$. Notice that the time $t = 1$ is in $\mathcal{T}$. Index the elements such that $\mathcal{T} := \{t_1, ..., t_\tau\}$ where $1 = t_1 < t_2 < ... < t_\tau$. We then compute a master partition of the time horizon T into the intervals $\mathcal{I} = \{[t_1, t_2 - 1], [t_2, t_3 - 1], ..., [t_{\tau-1}, t_\tau - 1], [t_\tau, T]\}$. There are two key properties of this partition: the cost of any job changes by at most a factor of $1+\epsilon$ as its completion time varies within an interval, and the number of intervals is a polynomial in n, $\log P$ and $\log W$. Here P denotes the length of the longest job and $W = \max_{j,t}(f_j(t) - f_j(t-1))$, the maximum increase in cost function $f_j(t)$ in one time step over all jobs j and times t.

Next we define a modified cost function $f'_j(t)$ for each time $t \in \mathcal{T}$; in essence, the modified cost is an upper bound on the cost of job j completing in the interval for which t is the left endpoint. More precisely, for $t_i \in \mathcal{T}$, let $f'_j(t_i) := f_j(t_{i+1} - 1)$. Notice that, by construction, we have that $f_j(t) \leq f'_j(t) \leq (1+\epsilon) f_j(t)$ for each $t \in \mathcal{T}$. Consider the following integer programming formulation with variables x'_{jt} for each job j and each time $t \in \mathcal{T}$; we set the variable x'_{jt_i} to 1 to indicate that job j completes within the interval $[t_i, t_{i+1} - 1]$. The demand $D(t)$

is defined the same way as before.

$$\text{minimize} \quad \sum_{j=1}^{n} \sum_{t \in \mathcal{T}} f'_j(t) x'_{jt} \tag{IP'}$$

$$\text{subject to} \quad \sum_{j=1}^{n} \sum_{s \in \mathcal{T}: s \geq t} p_j x'_{js} \geq D(t), \qquad \text{for each } t \in \mathcal{T}; \tag{2.6}$$

$$\sum_{t \in \mathcal{T}} x'_{jt} = 1, \qquad \text{for each } j = 1, \ldots, n; \tag{2.7}$$

$$x'_{jt} \in \{0, 1\}, \qquad \text{for each } j = 1, \ldots, n,\ t \in \mathcal{T}.$$

The next two lemmas relate (IP′) to (IP).

Lemma 7. *If there is a feasible solution x to (IP) with objective value v, then there is a feasible solution x' to (IP′) with objective value at most $(1 + \epsilon)v$.*

Proof. Suppose $x_{jt} = 1$ where t lies in the interval $[t_i, t_{i+1} - 1]$ as defined by the time indices in $\mathcal{T}$, then we construct a solution to (IP′) by setting $x'_{jt_i} = 1$. It is straightforward to check x' is feasible for (IP′), and by construction $f'_j(t_i) = f_j(t_{i+1} - 1) \leq (1 + \epsilon) f_j(t)$. □

Lemma 8. *If there is a feasible solution x' to (IP′) with objective value v', then there is a feasible solution x to (IP) with objective value v'.*

Proof. Suppose $x'_{jt} = 1$, where $t = t_i$; then we construct a solution to (IP) by setting $x_{j,t_{i+1}-1} = 1$. Notice that the time $t_{i+1} - 1$ is the right endpoint to the interval $[t_i, t_{i+1} - 1]$. By construction, $f_j(t_{i+1} - 1) = f'_j(t_i)$; hence, the cost of solution x is also v'. To check its feasibility, it suffices to see that the constraint corresponding to $D(t_i)$ is satisfied. This uses the fact that within the interval

$[t_i, t_{i+1} - 1]$, $D(t)$ is largest at t_i and that the constraint corresponding to $D(t)$ contains all variables x_{js} with a time index s such that $s \geq t$. □

Using the two lemmas above, we see that running the primal-dual algorithm using the LP relaxation of (IP′) strengthened by the knapsack cover inequalities gives us a $2(1+\epsilon)$-approximation algorithm for the scheduling problem $1||\sum f_j$. Hence we have the following result:

Theorem 5. *For each $\epsilon > 0$, there is a $(2+\epsilon)$-approximation algorithm for the scheduling problem $1||\sum f_j$.*

The combination of the interval-indexed formulation along with the generality of the objective function allows us to capture even more general problems within the same framework. For example, in *scheduling problems with machine unavailability*, the machine might not be available at all times due to scheduled maintenance. We can modify our algorithm to give a $(2+\epsilon)$-approximation algorithm for general min-sum objective scheduling problems with machine unavailability, provided that the unavailable intervals are known in advance.

The setting of the problem is as follows: there are S periods in which the machine is available, denoted by $[B_i, F_i]$ where $i = 1, ..., S$. We assume all time B_i and F_i are integral. We also assume that the machine is *resumable*, meaning if the processing of a job is interrupted because of machine down-time, it can be continued without any penalty once the machine becomes available again. Without loss of generality, we assume that the total available time equals to the sum of processing time of all jobs, that is $\sum_{i=1}^{S}(F_i - B_i) = \sum_{j=1}^{n} p_j$. Hence the processing of all jobs would be completed at time F_S. As before, we are given cost function $f_j(t)$ for each job j which is non-decreasing in t. Furthermore,

the function $f_j(t)$ is only defined on time t where the machine is available in the time interval $[t-1,t]$. The objective is to minimize $\sum_{j=1}^{n} f_j(C_j)$, where C_j denotes the completion time of job j.

We first describe the modification of the primal-dual algorithm that yields a 2-approximation for this scheduling problem that runs in pseudopolynomial time. Let $m(t)$ denote the aggregated amount of processing time available on the machine up to time t. Notice that at any time t, at least $\sum_{j=1}^{n} p_j - m(t) + 1$ units of processing needs to be completed at or after time t. Hence, we modify the definition of the demand at time t as $D(t) := \sum_{j=1}^{n} p_j - m(t) + 1$. For each subset of jobs $A \subseteq 1, ..., n$, we then define the residual demand at time t with respect to A as $D(t, A) := \max\{D(t) - \sum_{j \in A} p_j, 0\}$. Using this modified definition of residual demand, we formulate the LP relaxation (P) with knapsack-cover inequalities as before, but only for the time indices t for which the machine is available during $[t-1,t]$. Running the primal-dual algorithm on (P) and its corresponding dual (D) produces a solution of cost at most twice optimal. This can be converted into a polynomial time $(2+\epsilon)$-approximation algorithm by applying the rounding procedure in the previous section. In fact, the same techniques can be further generalized to the setting in which the machine runs at a time varying speed $s(t)$, so that within the interval $(t, t']$ the machine has the capacity to process $\int_t^{t'} s(t)dt$ units of processing.

2.4 Computational Results

In this section we analyze the performance of our primal-dual scheduling algorithm in computational experiments. This is by no means a comprehensive

computational study, but rather a proof of concept that it is possible to apply our primal-dual algorithm to various single-machine scheduling problems to provide solutions with reasonable quality. We test our algorithm for two specific objectives: (1) the weighted sum of tardiness, and (2) the weighted sum of tardiness square. Given a set of n jobs where job j has an associated due date d_j, the tardiness of a job is defined to be $T_j = \max\{C_j - d_j, 0\}$. The reason why we conduct our computational experiments on objectives related to weighted tardiness is that it is one of the fundamental objectives in machine scheduling. There exist many heuristics in the literature that work well in practice and had been tested extensively, yet until recently there is no constant factor approximation algorithms for the weighted tardiness objective [9]. To the best of our knowledge, there is no previous work done on the weighted tardiness squared objective.

2.4.1 Weighted Tardiness

We conduct our test on problem instances where the number of jobs, n, range from 10 to 100. For $n = 40, 50$ and 100, we use the instances that are publicly available in OR-Library , as they are the standard benchmark instances for comparing heuristics for the weighted tardiness problem [11]. For $n = 10$ and 20, there are no available instances in the OR-library, but we adopt the same generation scheme for the instances used there, which we now describe. For each job j ($j = 1, ..., n$), an integer processing time p_j is generated from the uniform distribution $[1, 100]$ and integer weight w_j is generated from the uniform distribution $[1, 10]$. There are two parameters that control how the due date d_j is generated: the average tardiness factor (TF) and the relative range of due dates

(RDD). Given these two parameters, an integer due date d_j is randomly generated from the uniform distribution $[P(1-TF-RDD/2), P(1-TF+RDD/2)]$, where $P = \sum_{j=1}^{n} p_j$. For this experiment, we use five values for TF and RDD: 0.2,0.4,0.6,0.8 and 1, giving us 25 pairs of values for TF and RDD. We generate five test cases for each set of parameters, giving us 125 test cases for each fixed number of jobs.

We make no attempt to optimize the run time of our implementation of the primal-dual algorithm, as we use the pseudo-polynomial version where for each job, there is a decision variable x_{jt} for each time index for the length of the schedule. The average run time is about 0.1 seconds for $n = 10$, but grows to about 685 seconds for $n = 100$. It is an interesting open question to improve the efficiency of the primal-dual scheduling algorithm.

We compare our solution with a well known heuristic for minimizing weighted tardiness, known as Apparent Urgency (AU) rule. The AU is a constructive heuristic that compare favorably with other heuristics, and it is used to produce initial solution to a metaheuristic known as dynasearch, which produces the best known solution to the OR-library test instances [17]. The AU rule is a dynamic list scheduling rule that selects an unscheduled job j with the highest AU_j value to the next available position of the sequence, where AU_j is defined by

$$AU_j = \frac{w_j}{p_j} \exp(-\frac{n \cdot \max\{0, d_j - p_j - t\}}{k \cdot P}).$$

In this expression, t is the sum of the processing times of jobs already scheduled, P is the total processing time of all jobs, and k is the lookahead parameter that is set to $k = 0.5$ for $TF = 0.2$, $k = 0.9$ for $TF = 0.4$ and $k = 2.0$ for $TF > 0.4$.

We first report the result for our primal-dual algorithm. We report the error

of our solution when compared to the best known solution in the literature in terms of mean, median and the maximum error in percentage. For $n = 10, 40$ and 50, these are verified to be the actual optimal solution. For $n = 20$, we generated the corresponding IP formulation, use the better solution generated from the primal-dual algorithm and the AU rule as an initial solution for CPLEX solver with a time limit of 600 seconds to generate a solution for comparison. For $n = 100$, we use the best known solution reported in the literature. We also report the duality gap, which is the difference between the value of the solution, and the dual feasible solution. This serves as an upper bound to the error to the optimal value, when the optimal value is not known. Finally the Num. Opt column reports the number of instances (out of a total of 125) where the optimal solution is found. We summarize the result below. The units used for all columns related to error and duality gap is in percentage (%).

N	Avg. Error	Median Error	Max Error	Avg. D. Gap	Num. Opt
10	1.537	0	31.481	11.516	66
20	1.023	0.117	16.520	9.762	56
40	0.669	0.188	12.980	6.965	36
50	0.742	0.264	10.453	8.737	29
100	0.403	0.131	11.421	4.189	30

We see that in terms of average performance, the quality of the solution generated is closer to the optimal as the number of jobs increase. One of the reason might be that the duality gap is also decreasing as the number of jobs increase. On the other hand, there are also more instances in which optimal solutions are found for smaller value of n, which indicates there is more variability to the performance of the algorithm for small n. It is also worth noting that the majority

of the duality gap is due to the gap between the value of the dual feasible solution and the optimal value, as the error is much smaller compared to the duality gap. Next we report the results of the AU algorithm.

N	Avg. Error	Median Error	Max Error	Num. Opt
10	13.176	2.551	457.1429	37
20	36.358	3.782	2200	38
40	240.958	1.612	13700	19
50	236.06	2.011	18800	18
100	39.230	1.142	2225	18

We see that the primal-dual algorithm outperforms the AU rule in every statistic. One significant difference worth pointing out is the Max. Error, where there are a few test cases that are extreme outliers. There are instances where the error is well over 10000 percent, which also affects the reported average error significantly, whereas the median error is not affected as much by these outliers. This highlight one of the main advantage of an approximation algorithm: there is a performance guarantee for the solutions being generated. On the other hand, it is worth pointing out the run time of AU rule is extremely low, in the order of fraction of a second. This is because its run time is essentially equivalent to sorting n numbers, which takes $O(n \ln n)$ time.

Finally, we compare the performance by counting the number of test cases where each algorithm provides the better schedule (i.e., has a smaller cost).

N	Primal-Dual	AU	Tie
10	68	30	27
20	81	22	22
40	83	26	16
50	80	29	16
100	77	30	18

We see that the primal-dual algorithm performs better than the AU rule in about 62 percent of test cases, follow by 22 percent where the two algorithms provide the same solution. Note that a significant number of these ties are for easy instances where the optimal value is 0, i.e., every job finishes before its due date. On the other hand, despite its simplicity, there are also about 16 percent of the test cases where AU rule provides the better solution.

2.4.2 Weighted Tardiness Squared

The generation of test cases for the weighted tardiness squared objective follows the same scheme for weighted tardiness, except the processing times have a smaller range, and follows the uniform distribution $[1, 20]$. In our test, we let the number of jobs, n, vary from 10 to 100 and generate 125 test cases for each n for various values of parameter TF and RDD, as described in the previous section. In contrast to the previous test where the optimal value for most test cases is known, it is very difficult to determine the optimal solution here, as CPLEX was unable to solve the integer program, or even get a good solution within a reasonable time limit. Instead we report the gap between the cost of our solution and the dual solution generated. We now summarizes our results.

In addition, we also find the average LP Gap for test cases from n=10 to 60. The LP Gap of an instance is define as the ratio of the difference between the value of the integer solution and the optimal LP value for the natural LP relaxation (without knapsack cover inequalities added), and the optimal LP value. Both the duality gap and the LP gap is an upper bound to the error of our solution, as both the dual solution and the optimal solution to the natural LP relaxation is a lower bound to the optimal value.

The average duality gap and LP gap is reported in percentage (%).

N	Duality Gap	LP Gap
10	10.430	119.78
20	9.941	71.66
30	8.687	59.88
40	7.491	42.36
50	7.042	41.05
60	6.199	30.84
70	6.046	n/a
80	5.246	n/a
90	4.624	n/a
100	4.404	n/a

We see the performance of the primal dual algorithm is similar to the results reported for weighted tardiness, indicating the robustness and versatility of our problem. The results also follows the same trend as the previous section, for example the dual gap is decreasing as the number of jobs increase. It is also interesting to note that the LP gap is much larger compared to the duality gap,

which indicate the value of adding knapsack cover inequalities to the LP relaxations.

CHAPTER 3

SUBMODULAR JOINT REPLENISHMENT PROBLEM

3.1 Introduction

Inventory models with deterministic but non-stationary demand have a long and rich history in supply chain management, beginning with the seminal paper of Wagner and Whitin[51]. One of the main features of these models is to capture the tradeoff between holding costs and fixed setup costs (also referred to as ordering cost). Setup costs typically represent, among others, the use of machines, trucks, and/or laborers. When multiple item types are ordered in the same period, some of the fixed costs are typically shared among the different item types. In most inventory management models, the economies of scale are traditionally captured by a *joint setup cost* structure. The most traditional model uses the *additive* joint setup cost structure. In this model, there is a joint setup cost if *any* item type is ordered, in addition to an individual item setup cost for each item type ordered. This problem is known as a *joint replenishment problem (JRP)*. The additive joint setup cost structure is clearly very limited in some cases, yet there is a surprising lack of results for models with more complex setup cost structures and non-stationary demand. This in stark contrast to the work of Federgruen and Zheng [21], and Teo and Bertsimas [47] which gave near-optimal algorithms for inventory models with a very general setup cost structure but with *constant* demand. In this work, we consider several generalizations of the deterministic, *non-stationary* joint replenishment problem beyond the additive cost structure; these capture many interesting situations in practice. For each of these variants, we provide an efficient algorithm that finds a feasible

solution with strong worst-case performance guarantees. Since joint replenishment problems are typically NP-hard, it is computationally intractable to find optimal solutions quickly. However, our algorithms can efficiently find solutions that are provably close to optimal, and in the worst case are guaranteed to be within a fixed constant factor of the optimal cost.

In the models we study, there are multiple item types each with a sequence of demands over a discrete time horizon of finitely many periods. That is, there is a specific demand quantity for each item type due in each time period. Each demand must be satisfied by an order at or prior to its due date period, which means neither backlogging nor lost sales are allowed. The cost of satisfying the demand is composed of setup costs and holding costs. The joint setup cost for ordering a set of items in any time period is a function of the specific set of item types ordered. The setup cost function satisfies two natural properties known as *monotonicity* and *submodularity*. The monotonicity requirement simply means that as more item types are ordered, the total setup cost does not decrease. The submodularity requirement characterizes the economies of scale in ordering more item types, i.e., the marginal (additional) cost of ordering any specific item type decreases as more item types are ordered. The holding cost for each demand point depends on the item type and the length of time the inventory was held. We assume no capacity constraints and stationary per unit variable costs. One can easily show that zero-inventory ordering policies are optimal in this model, and thus every demand of a given item type is satisfied by the latest order of that item prior to its due date period. The goal is to satisfy all the demands by a sequence of orders that minimizes the total setup costs plus holding costs. We refer to this problem as the Submodular Joint Replenishment Problem.

In this chapter, we study two special cases of the submodular joint replenishment problem with non-stationary demand that capture a rich variety of applications. These cases are called the Tree JRP, and the Cardinality JRP. In the *Tree JRP* case we are given a rooted tree where every node represents a process that incurs a setup cost, and the leaves of the tree represent the item types. The joint setup cost of ordering any subset of item types is the cost of all the nodes on the paths from the root to the leaves corresponding to the item types being ordered. One application of the Tree JRP is in maintenance scheduling problems for aircraft engines studied by Levi, Magnanti and Zarybnisky ([37]). Each module of the engine corresponds to a node in the tree, and to get to a certain engine part requires removing all necessary modules. It is worth noting that the Tree JRP is a generalization of the additive JRP. The *Cardinality JRP* is the case where the joint setup cost function has the property that the marginal cost of any item type depends on the cardinality of the set of item types already being ordered. A natural application of this model is when all of the item types are very similar, but vary in only one aspect such as color or size. We give novel algorithms for the Tree JRP and Cardinality JRP that are within a factor of three and five, respectively, of the optimal offline cost.

Literature Review. Joint replenishment problems are infamous for being intractable, and thus have been typically studied via the notion of *approximation algorithms*. When demand is assumed to be stationary and continuous for the additive JRP, Roundy [44] showed that "power-of-two" policies have approximation ratios of 1.06 and 1.02, depending on whether the base planning period is fixed or not. Federgruen and Zheng [21], generalized these results for the submodular JRP with constant demand. However, the literature for the submodular JRP with non-stationary demand has focused primarily on the additive

joint setup cost structure, which was shown to be NP-hard by Arkin, Joneja and Roundy [3]. Several heuristics for the non-stationary additive JRP have been proposed with varying degrees of theoretical performance guarantees in work by Veinott [49], Zangwill [53], Kao [33], Joneja [32], Federgruen and Tzur [20], Levi, Roundy and Shmoys [39], and Stauffer, Massonnet, Rapine and Gayon [46]. The current best approximation algorithm for the additive JRP with non-stationary demand is due to Levi, Roundy, Shmoys and Sviridenko [38], which has an approximation ratio of 1.80. Chan, Muriel, Shen, Simchi-Levi and Teo [15] show that zero-inventory policies are near-optimal for joint replenishment problems with piecewise linear costs, however their conditions do not imply submodularity. Since the non-stationary submodular JRP is quite general, we instead consider two special cases called the Tree and Cardinality JRP. These cases are fundamentally different than the traditional JRP which has been the focal point in the literature. In each case, we provide an efficient algorithm with a constant factor approximation ratio.

Contributions. The Tree JRP model captures situations where each item type requires a chain of processes to be produced, and several of those processes are shared by other item types. The tree with only a root and leaves is identical to the additive joint setup cost structure, and thus this problem is also NP-hard. We provide an approximation algorithm that is no more than three times the optimal offline cost. This algorithm is similar in spirit to the one proposed by Levi, Roundy, Shmoys and Sviridenko [38] which only considered the additive JRP. Specifically, the algorithm is based on solving a linear program, and then successively rounding the variables corresponding to the nodes in the tree in a particular fashion. Specifically, we start by opening orders containing the root process, and work our way down the tree using a breadth-first search. For every

node, we round the corresponding variables to a time where the parent node has been ordered, thus ensuring that all necessary processes for an item type are accounted for when it is ordered.

The Cardinality JRP is another rich problem that captures the additive JRP model and can be specialized to the case where the joint setup cost function simply depends on the number of item types being ordered. This implies that the setup cost is concave in the cardinality of the set of item types being ordered. Although the Cardinality JRP is NP-hard, we provide an efficient algorithm with an approximation ratio of five. This algorithm is based on an iterative rounding procedure that uses the variables from a linear relaxation of a novel integer programming formulation of the Cardinality JRP. Our algorithm carefully builds up orders based on their size, while ensuring that the cost of any particular order can be paid for using the primal objective costs. The holding costs are accounted for using a property of the respective dual linear program.

3.2 Formulation

In this section we give a precise mathematical formulation of the submodular JRP, then give a integer programming formulation and the corresponding LP relaxation. Finally we explore some properties of our LP relaxation.

The input of submodular JRP is as follows: there are N items that are needed over a planning horizon of T periods, and a deterministic demand $d_{it} \geq 0$ for each item $i = 1, ..., N$ and time period $t = 1, ..., T$. Each demand point d_{it} with positive demand needs to be served by an order of item i in period s with $s \leq t$. There are two components to the total cost incurred, the ordering cost and

holding cost.

To model the ordering cost, let $f(S)$ denote the cost of ordering the set of items S in any given period. As noted before, the only requirement is that f is a non-negative monotone submodular function, meaning that for every set $S_1, S_2 \subseteq \{1, \ldots, N\} f(S_1) + f(S_2) \geq f(S_1 \cup S_2) + f(S_1 \cap S_2)$. This definition can be shown to be equivalent to the following: for every set $S_1 \subseteq S_2$ and any item $i \notin S_2$,

$$f(S_1 \cup \{i\}) - f(S_1) \geq f(S_2 \cup \{i\}) - f(S_2),$$

which captures the effects of economy of scale. In contrast, in the classical Joint Replenishment Problem, ordering cost consists of two components: a joint ordering cost K_0 which is independent of the subset of items that are included in the order, and item ordering cost K_i for item i. The total ordering cost for a set S is $f(S) := K_0 + \sum_{i \in S} K_i$. It is easy to see that this satisfies the monotonicity and submodularity property, hence the classical JRP is a special case of our problem.

The holding cost from period s to t for the demand of item i in period t is denoted $H^i_{st} := d_{it}.h^i_{st}$, where h^i_{st} is the per unit holding cost of item i from period s to t. We assume h^i_{st} is nonnegative, and for each (i, t) is a non-increasing function of s.

The following is an integer programming formulation of the Submodular JRP. We use the 0-1 variable y^S_s to indicate if the set S is ordered in period s, and the 0-1 variable x^i_{st} to indicate whether the demand d_{it} was satisfied using an order from period s.

$$\text{minimize} \quad \sum_{S} \sum_{s=1}^{T} f(S) y_s^S + \sum_{i=1}^{n} \sum_{t=1}^{n} \sum_{s=1}^{t} H_{st}^i x_{st}^i \tag{IP}$$

$$\text{subject to} \quad \sum_{s=1}^{t} x_{st}^i = 1, \qquad i = 1, ..., N, t = 1, ..., T \tag{3.1}$$

$$x_{st}^i \leq \sum_{S: i \in S} y_s^S, \qquad i = 1, ..., N, t = 1, ..., T, s = 1, ...t \tag{3.2}$$

$$x_{st}^i, y_s^S \in \{0, 1\}$$

We first argue that this is indeed a valid formulation for the submodular JRP. Given an optimal solution to the submodular JRP, every demand (i, t) is served by some order S in time period $\{s \leq t\}$ that contains the item i. Also, once the orders are fixed, each demand (i, t) is served entirely from the order in time $s : s \leq t$ containing item i that is closest to period t to minimize the holding cost. Hence constraints (3.1) and (3.2) are satisfied, and the cost is given by the objective. Conversely, given a feasible solution to (IP), it must be a feasible solution to submodular JRP since by constraint (3.1) each demand point (i, t) is served and by constraint (3.2), the demand (i, t) is served by period s only if there contains an order S in period s that contains the item i.

Next we get the natural LP relaxation of IP1 by relaxing the $\{0, 1\}$ constraints on the variables:

$$\text{minimize} \quad \sum_{S} \sum_{s=1}^{T} f(S) y_s^S + \sum_{i=1}^{n} \sum_{t=1}^{n} \sum_{s=1}^{t} H_{st}^i x_{st}^i \tag{P}$$

$$\text{subject to} \quad \sum_{s=1}^{t} x_{st}^i = 1, \qquad i = 1, ..., N, t = 1, ..., T \tag{3.3}$$

$$x_{st}^i \leq \sum_{S: i \in S} y_s^S, \qquad i = 1, ..., N, t = 1, ..., T, s = 1, ...t \tag{3.4}$$

$$x_{st}^i, y_s^S \geq 0$$

We first prove that there exists an optimal solution to (P) where all the sets that are ordered in a given time period t are nested.

Lemma 9. *There exists an optimal solution (x, y) to (P) such that for any given period t and any two sets R and S, if $y_t^R > 0$ and $y_t^S > 0$ then they are nested, i.e., either $R \subset S$ or $S \subset R$.*

Proof. Given an arbitrary optimal solution $(\hat{x}, \hat{y})$ to (P), we will construct (x, y) with the desired property. Fixed a time period t, let $z_t^i := \sum_{S \ni i} \hat{y}_t^S$ be the fractional number of sets which contains the item i. Now consider the following auxiliary optimization problem which has a variable r^S for every set S (we drop the time index t for convenience):

$$\text{minimize} \quad \sum_{S} f(S) r^S \tag{AUX-t}$$

$$\text{subject to} \quad \sum_{S \ni i} r^S \geq z_t^i, \qquad i = 1, .., N \tag{3.5}$$

$$r_{\geq}^{S} 0$$

Recall $f(S)$ is the cost of ordering cost S and (AUX-t) exactly captures the cheapest way for ordering items so that each item is ordered at least z_t^i. However, $f(S)$ is a submodular function, hence (AUX-t) is the dual of polymatroid, where the greedy algorithm finds an optimal solution. Specifically, assume without loss of generality that the items are indexed such that $z_t^1 \geq z_t^2 \geq \ldots \geq z_t^n$ and let $S(i) := \{1, ..., i\}$. Then an optimal solution to (AUX-t) is given by $r^{S(n)} = z_t^n$, and $r^{S(i)} = z_t^i - z_t^{i+1}$ for $i = 1, ..., n-1$ and every other $r^S = 0$ for all other S. Notice that by construction $S(i)$ has the desired nested property. Finally, setting $y_t^S = r^S$ where r^S is the optimal solution to (AUX-t) and $x = \hat{x}$ gives a feasible solution to (P) that cost no more than that of $(\hat{x}, \hat{y})$, hence an optimal solution to (P) with the nested property as desired. □

Now we take the dual of (P). Let b_t^i and l_{st}^i be the dual variable corresponding to first and second set of constraints in (P) respectively. The dual of the (P) is:

$$\text{maximize} \quad \sum_{i=1}^{N} \sum_{t=1}^{T} b_t^i \tag{D}$$

$$\text{subject to} \quad b_t^i \leq H_{st}^i + l_{st}^i, \qquad i = 1, .., N, t = 1, .., T, s = 1, .., t \tag{3.6}$$

$$\sum_{i \in S} \sum_{t=s}^{T} l_{st}^i \leq c(S), \qquad s = 1, ..., T, S \subseteq \{1, ..., N\} \tag{3.7}$$

$$l_{st}^i \geq 0$$

Note that (P) contains an exponential number of variables and (D) contains an exponential number of constraints. Although the separation oracle can be implemented in polynomial time by using submodular function minimization, this doesn't give a efficient algorithm for solving (P) and (D) in practice. In the next few sections we consider polynomial-size LP relaxations for various spe-

cial cases of the submodular JRP, and derive LP-based approximation algorithm using these LP relaxations.

3.3 Joint Replenishment Problem with Tree Ordering Cost

In this section, we consider the Tree JRP where the ordering cost is specified by a rooted tree. Specifically, there is a cost K_j associated for each node in the tree, and each item i is a leaf in the tree. Let j be a node in the tree and $path(j)$ denote the unique path from node j to the root of the tree. The cost of ordering a set of item S is defined to be $\sum_{j \ni path(i) | i \in S} K_j$, i.e., the cost of all nodes that are in the path from some item i to the root, where item i is in set S. The structure of the holding cost is the same as the classical JRP.

Note that this contains the classical JRP as a special case since the ordering cost of the classical JRP can viewed as a tree with a root node with cost K_0, the joint ordering cost, connected to N leaves, one for each item i with cost K_i, the cost of ordering item i. We extend the ideas in the work of Levi, Roundy, Shmoys and Sviridenko [38] to give a 3 approximation algorithm for this problem via LP Rounding.

3.3.1 A Linear Program

The following is the natural LP relaxation of an IP formulation for the tree JRP. We use the 0-1 variable y_s^j to indicate if the node j is ordered in period s, and 0-1 variable x_{st}^i to indicate whether the demand d_{it} was provided from period s.

$$\text{minimize} \quad \sum_{s=1}^{T} \sum_{j} K_j y_s^j + \sum_{i=1}^{n} \sum_{t=1}^{n} \sum_{s=1}^{t} H_{st}^i x_{st}^i \tag{P-T}$$

$$\text{subject to} \sum_{s=1}^{t} x_{st}^i = 1, \qquad i = 1, ..., N, t = 1, ..., T \tag{3.8}$$

$$x_{st}^i \leq y_s^j, \qquad i = 1, ..., N, t = 1, ..., T, s = 1, ...t, j \in path(i) \tag{3.9}$$

$$x_{st}^i, y_s^j \geq 0$$

The correctness of (P-T), and in particular for constraint (3.9) follows from the fact that in order to place an order for item i, one has to pay $\sum_{j \in path(i)} K_j$.

Next we argue that (P-T) is as least as strong as (P), the LP relaxation given for the general Submodular JRP. Let Z_{PT} and Z_P be the optimal value of (P-T) and (P) respectively.

Lemma 10. *(P-T) is equivalent to (P) for the JRP with tree ordering cost, i.e.* $Z_{PT} = Z_P$

Proof. First we show that we can convert an optimal solution (x, y) of (P-T) to a feasible solution $(\hat{x}, \hat{y})$ of (P). We start by observing some properties about the optimal solution of (P-T).

Consider any nodes j and j' such that $j' \in path(j)$, then $y_s^j \leq y_s^{j'}$ for each time period $s = 1, ..., T$. This *monotonicity property* follows from constraint (3.9) in (P-T) and the fact that the set of items i where j is in $path(i)$ is a subset of those where j' is in $path(i)$.

Furthermore, for any time period s and any node i that is a leaf in the tree (i.e., i corresponds to an actual item), $y_s^i = \max_t x_{st}^i$ since otherwise we can de-

crease the value of y_s^i (and therefore its objective value) without affecting its feasibility.

Finally, for each node j in the tree that is not a leaf, let $C(j)$ be the set of children of j. Then for each time period s, $y_s^j = \max_{k \in C(j)} y_s^k$; otherwise we can decrease y_s^j without affecting feasibility.

Given the properties above, we can determine the set of orders that are induced by the optimal solution (x, y). In a particular time period s, let $s(.)$ be an ordering of the items $\{1, \ldots, N\}$ such that $y_s^{s(1)} \geq y_s^{s(2)} \geq \ldots \geq y_s^{s(N)}$. Then the sets that are ordered in period s are of the form $S(i), i = 1, \ldots, N$, where $S(i)$ includes the items $\{s(1), s(2), \ldots, s(i)\}$. The cost of set $S(i)$ is the sum of K_j for all nodes j in the minimal connected subtree that contains the leaves $s(1), \ldots, s(i)$. Notice that as in Lemma 9, the sets ordered here are nested, i.e. $S(1) \supset S(2) \supset \ldots \supset S(n)$.

Now we are ready to describe the conversion of an optimal solution (x, y) of (P-T) to a feasible solution $(\hat{x}, \hat{y})$ of (P). Let $\hat{x} = x$ and $\hat{y}_s^S = 0$ except for $\hat{y}_s^{S(n)} = y_s^{s(n)}$,$\hat{y}_s^{S(i)} = y_s^{s(i)} - y_s^{s(i+1)}$ for $i = 1, \ldots, n-1$. Notice by construction the item $s(i)$ is in set $S(i), \ldots, S(n)$ and using a telescoping sum on $\hat{y}_s^{S(i)}, \ldots, \hat{y}_s^{S(n)}$, it is easy to verify that $\sum_{S: s(i) \in S} \hat{y}_s^S = y_s^{s(i)}$ Hence the second set of constraint in (P) is satisfied from constraint (3.9) of (P-T). The proof is completed by noticing that the cost of $(\hat{x}, \hat{y})$ in (P) is the same as (x, y) in (P-T) by construction.

For the converse, we describe how to convert an optimal solution $(\hat{x}, \hat{y})$ of (P) to a feasible solution (x, y) of (P-T). Fix a time period s, and by Lemma 9, the sets S where $\hat{y}_s^S > 0$ are nested. Without loss of generality, we indexed the item so that all sets ordered are of the form $S(i) := \{1, ..., i\}$ where $i = 1, ..., N$. To

construct a feasible solution of (P-T), we first let $x = \hat{x}$. Next we describe how to set the value for variable y. For each node that is a leaf (which corresponds to an actual item), we let $y_s^i = \sum_{k=i}^{N} \hat{y}_s^{S(k)}$, which are all the sets ordered in (P) which included item i. Now, for each node j that is not a leaf, let $C(j)$ be the set of children of j. Then set $y_s^j = \max_{k \in C(j)} y_s^k$. It is easy to check that this solution is feasible for (P-T), and using telescoping sum, we can verify that the cost of (x, y) constructed for (P-T) is the same as the given solution $(\hat{x}, \hat{y})$ for (P), completing the proof. □

Next we consider the dual of (P-T), which will be used in the analysis of our LP rounding algorithm. Let b_t^i and l_{st}^{ij} be the dual variable corresponding to constraints (3.8) and (3.9) in (P-T) respectively. The dual of the (P-T) is:

$$\text{maximize} \quad \sum_{i=1}^{N} \sum_{t=1}^{T} b_t^i \tag{D-T}$$

$$\text{subject to} \quad b_t^i \leq H_{st}^i + \sum_{j \in path(i)} l_{st}^{ij}, \qquad i = 1, .., N, t = 1, .., T, s = 1, .., t \tag{3.10}$$

$$\sum_{i: j \in path(i)} \sum_{t=s}^{T} l_{st}^{ij} \leq K_j, \qquad s = 1, ..., T, j \tag{3.11}$$

$$l_{st}^{ij} \geq 0$$

3.3.2 The LP Rounding Algorithm

We will show how to round the optimal solution to (P-T), denoted by (x, y) to a feasible solution to the tree JRP problem with cost at most 3 times the optimal value of (P-T), thus giving a 3 approximation algorithm.

Our rounding procedure consider each node in the rooted tree one at a time, starting at the root node. The nodes can be process in any order, as long as any node j is processed after all the nodes in $path(j)$ (i.e. the nodes in the path from j to the root). Hence one can use Breadth First Search (BFS) or Depth First Search (DFS) starting from the root node.

We first describe the processing of the root node where we decide in which time period to place orders. The rounding procedure is based on the value of $y_1^0, ..., y_T^0$, the variables corresponding to fractional orders of the root node in (P-T). Consider the interval $(0, \sum_{r=1}^{T} y_r^0]$, which is the total weight of fractional orders of the root node. In this interval, focus on the points $1, 2, ..., \lfloor \sum_{r=1}^{T} y_r^0 \rfloor$, which we call *service points* for the root node. For each period $m = 1, ..., T$, define the interval $Y_m^0 = (\sum_{r=1}^{m-1} y_r^0, \sum_{r=1}^{m} y_r^0]$, which is of length y_m^0. We place an order for the root node in period m if there is a service point within the interval Y_m^0. Let $T^0 := \{r_1^0 < ... < r_M^0\}$ be the set of time periods in which orders are placed for the root node.

Next we describe the processing of other nodes in the tree. Let node j be the current node being processed. Recall that by construction, all other nodes in $path(j)$ has already been processed at that time. Let j' be the parent of j, and $T^{j'} := \{r_1^{j'} < ... < r_M^{j'}\}$ be the set of time periods in which orders are placed for node j'. Consider the interval $(0, \sum_{r=1}^{T} y_r^j]$. Similar to above, we call the points $1, 2, ..., \lfloor \sum_{r=1}^{T} y_r^j \rfloor$ *service points* for node j. These service points are only used to determine the *tentative orders* for node j, since the orders for the nodes in $path(j)$ needs to be synchronized. Specifically, we can only open an order for j in time periods in which j' has an order. For each period $s = 1, ..., T$, we say there is a tentative order for node j in period s if there is a service point within the interval

$(\sum_{r=1}^{s-1} y_r^j, \sum_{r=1}^{s} y_r^j]$.

The tentative orders are then used to determine the *permanent orders* via a *two-sided push* procedure as follows. Because we only place orders for j in time periods s where an order is open for j' (i.e. $s \in T^{j'}$), if s is a temporary order for node j where $s \notin T^{j'}$, we which to *push* this tentative order to periods in which orders for j' has been placed. In particular, for each tentative order, we place up to two permanent orders. One order is placed in period s_l, the latest period in $T^{j'}$ prior to period s, and a second order is placed in s_r, the earliest period in $T^{j'}$ after period s. In other words, we place permanent orders for node j in $\max\{r \in T^{j'} : r \leq s\}$ and $\min\{r \in T^{j'} : r \geq s\}$. Notice that by construction $T^j \subseteq T^{j'}$,i.e. the orders are synchronized. The pseudo-code for the algorithm is given below:

Algorithm 2: LP Rounding Algorithm for JRP with tree ordering cost

```
Solve (P-T)
Generate order points for the root node
// Process the rest of the nodes in BFS or DFS order
for each node j do
    Generate temporary order points for node j
    for each time period s that contains a temporary order point do
        if ∃ an order for parent(j) in time s then
            open an order for node j in time s
        else
            Place order in period s_l and s_r, the two order of parent(j)
            closest to time s
Serve each demand point (i, t) from the nearest order including item i
```

3.3.3 Analysis

We first prove a key lemma in the analysis of the correctness of the algorithm and its cost.

Lemma 11. *For any node j and time interval I where $\sum_{t\in I} \hat{y}_t^j \geq 1$, there is an order for node j in I.*

Proof. The proof is by induction on the nodes of the tree, starting from the root. The induction argument uses the *monotonicity property* of optimal solution (x, y) mentioned before. In particular, any nodes j and j' such that $j' \in path(j)$, then $y_s^j \leq y_s^{j'}$ for all time period $s = 1, ..., T$.

Base case: By construction of the algorithm, for any time interval I where $\sum_{t\in I} y_t^0 \geq 1$, there exist a service point of the root node. Since an order is open for every time period that contains a service point of the root node the claim follows.

Inductive case: Consider node j and any time interval I where $\sum_{t\in I} y_t^j \geq 1$. Let j' be the parent of j. By the monotonicity property of y, we know $y_t^{j'} \leq y_t^j$ hence $\sum_{t\in I} y_t^{j'} \geq 1$. Using the induction hypothesis, we know there is an order for node j' in I. Also, there is a service point for node j in some time period s in I. We open an order for j via a two way push procedure in time periods closest to s (both to the left and to the right of the time horizon) where an order exists for j'. Since an order for j' exist in I, at least one order of j opened for the service point s is also in I and the claim follows. □

The correctness of the algorithm follows from the lemma above. Specifically, for any demand point (i, t), it follows from constraint (3.8) and (3.9) in (P-T) that

$\sum_{s=1}^{t} y_s^i \geq 1$. Let I be the time interval $[1, t]$ and from the lemma above we know there exist at least one order for item i in I. Hence for every demand point (i, t) there is an order of item i no later then time t that can serve the demand and the solution is feasible.

Next we analyze the cost of the solution returned by the algorithm. We start by considering the ordering cost. Since the number of orders made is at most $\lfloor \sum_{r=1}^{T} y_r^0 \rfloor$ for the root node and $2\lfloor \sum_{r=1}^{T} y_r^j \rfloor$ for all other nodes j (we make up to two orders for every service point of j) this gives as the following:

Lemma 12. *The total ordering cost for the solution by the algorithm is at most* $2 \sum_{s=1}^{T} \sum_j K_j y_s^j$.

Finally we analyze the holding cost incurred by the solution by the algorithm. We will show that the total holding cost incurred is at most $\sum_{i=1}^{N} \sum_{t=1}^{T} b_t^i$, the optimal value of the (D-T).

Lemma 13. *The total holding cost for the solution by the algorithm is at most* $\sum_{i=1}^{N} \sum_{t=1}^{T} b_t^i$

Proof. For any demand point (i, t), consider the set of orders s that serve (i, t) fractionally in the optimal solution for (P-T), i.e., $x_{st}^i > 0$. Let s_1 be the earliest of such orders and we define $[s_1, t]$ as the *active interval* for demand (i, t). Since $x_{s_1 t}^i > 0$, by the complementary slackness conditions, the corresponding dual constraint must be tight, i.e., $b_t^i = H_{s_1 t}^i + \sum_{j \in path(i)} l_{s_1 t}^{ij}$. This, combined with the nonnegativity constraints on $l_{s_1 t}^{ij}$, implies that $b_t^i \geq H_{s_1 t}^i$. However, we also assume that the holding cost is non-increasing in s. It follows that for any time s in the active interval, we have that $b_t^i \geq H_{st}^i$. Hence, it suffices to show that there exists an order for i in the active interval for (i, t). However, by the definition of

active interval and constraints (3.8) and (3.9) in (P-T), we have that $\sum_{s=s_1}^{t} \hat{y}_s^i \geq 1$. Let I be the time interval $[s_1, t]$ (i.e. the active interval) and using Lemma 11 shows that there exist an order of item i in the active interval of (i, t), as desired. □

By Lemmas 12 and 13, and the fact that the optimal value of (P-T) and (D-T) are both lower bounds to the value of the optimal solution to the Joint Replenishment Problem with tree ordering cost gives us the following result:

Theorem 6. *The LP Rounding algorithm is a 3-approximation algorithm for the Joint Replenishment Problem with tree ordering cost.*

3.4 Cardinality Joint Replenishment Problem

In this section we consider the case where the ordering cost is given by a concave function based on the cardinality of set of items being ordered. Formally, let $g(k)$ be a concave function which denote the cost of ordering k items in any given period. Then the cost of ordering the items S in any given period is $g(|S|)$.

Recall the integer programming formulation (IP) given for the submodular JRP in Section 2, we let the variable x_{st}^i indicate whether or not demand (i, t) was served from period s. We let the variable y_s^S indicate whether or not we ordered the set of items S in period s.

$$\text{minimize} \sum_{s=1}^{T} \sum_{S \subseteq N} f(S) y_s^S + \sum_{i=1}^{n} \sum_{t=1}^{T} \sum_{s=1}^{t} H_{st}^i x_{st}^i \tag{IP}$$

$$\text{subject to} \sum_{s=1}^{t} x_{st}^i = 1 \qquad i = 1, \ldots, n, t = 1, \ldots, T$$

$$x_{st}^i \leq \sum_{S | i \in S \subseteq N} y_s^S \qquad i = 1, \ldots, n, s = 1, \ldots, T, t = s, \ldots, T$$

$$x_{st}^i \in \{0, 1\} \qquad i = 1, \ldots, n, t = 1, \ldots, T, s = 1, \ldots, t$$

$$y_s^S \in \{0, 1\} \qquad S \subseteq N, s = 1, \ldots, T$$

As noted before, this formulation has an exponential number of variables. In the following, we will give a polynomial-size formulation for the cardinality JRP.

3.4.1 A Linear Program

In the following formulation, we let x_{st}^i have the same interpretation as before, and let z_s^i indicate whether an order with item i is placed in period s. Finally we let q_s^k indicate whether there was an order in S of at least size k in time s.

$$\text{minimize} \sum_{s=1}^{T}\sum_{k=1}^{n}(g(k)-g(k-1))q_s^k + \sum_{i=1}^{n}\sum_{t=1}^{T}\sum_{s=1}^{t} H_{st}^i x_{st}^i \qquad \text{(IP-C)}$$

$$\text{subject to} \sum_{s=1}^{t} x_{st}^i = 1 \qquad i=1,\ldots,n, t=1,\ldots,T$$

$$x_{s,t}^i \le z_s^i \qquad i=1,\ldots,n, s=1,\ldots,T, t=s,\ldots,T$$

$$q_s^{k+1} \le q_s^k \qquad k=1,\ldots,n-1, s=1,\ldots,T$$

$$\sum_{k=1}^{i} z_s^k \le \sum_{k=1}^{i} q_s^k \qquad i=1,\ldots,n, s=1,\ldots,T$$

$$x_{st}^i, z_s^i, q_s^k \in \{0,1\}$$

Lemma 14. *(IP-C) is a proper integer programming formulation for the cardinality JRP.*

Proof. First, we show that there is a one-to-one correspondence between solutions of (IP-C) and the cardinality JRP. Given a solution to (IP-C), we simply order and serve demand according to the variables x_{st}^i. Conversely, given a solution to the cardinality JRP, we let x_{st}^i be defined as it is served in the solution. For every $s = 1,\ldots,T$, if there are k items ordered in s, we let $z_s^i = 1$ for those k items and otherwise set $z_s^i = 0$. Also we set $q_s^1 = \ldots = q_s^k = 1$ and $q_s^{k+1} = \ldots = q_s^n = 0$. Next we check that all constraints are satisfied. First, since every demand is served by some order, the first set of constraints are satisfied. Next, a demand cannot be served without an order being placed, so the second set of constraints are satisfied. By construction of q_s^k above, the third set of constraints is satisfied. Now we need to check only the forth set of constraints. Again, let k be the number of items ordered at time s. Fix an item type $i = 1,\ldots,n$. We know that for each i, $z_s^i = 1$ if and only if i was ordered at s. If $i > k$, then the forth

constraint holds since the RHS is k and the LHS is at most k. If $i \leq k$, then the RHS is exactly i and the LHS is at most i since we are only checking if the first i items were ordered at s. Finally, one can also see that the objective in (IP-C) models the cost of a solution of the cardinality JRP correctly: The holding cost is modeled by the second term of the objective of (IP-C). The constraint $q_s^{k+1} \leq q_s^k$ combine with the concavity of $g(.)$ (i.e., $g(k) - g(k-1) \geq g(k+1) - g(k)$) implies that the first term in the objective is a telescoping sum which models the ordering cost of cardinality JRP correctly. □

We obtain the natural LP-relaxation of (IP-C) by relaxing the $\{0, 1\}$ constraints on the variables:

$$\text{minimize} \sum_{s=1}^{T}\sum_{k=1}^{n}(g(k) - g(k-1))q_s^k + \sum_{i=1}^{n}\sum_{t=1}^{T}\sum_{s=1}^{t} H_{st}^i x_{st}^i \tag{P-C}$$

$$\text{subject to} \sum_{s=1}^{t} x_{st}^i = 1 \qquad i = 1, \ldots, n, t = 1, \ldots, T \tag{3.12}$$

$$x_{s,t}^i \leq z_s^i \qquad i = 1, \ldots, n, s = 1, \ldots, T, t = s, \ldots, T \tag{3.13}$$

$$q_s^{k+1} \leq q_s^k \qquad k = 1, \ldots, n-1, s = 1, \ldots, T \tag{3.14}$$

$$\sum_{k=1}^{i} z_s^k \leq \sum_{k=1}^{i} q_s^k \qquad i = 1, \ldots, n, s = 1, \ldots, T \tag{3.15}$$

$$x_{st}^i, z_s^i, q_s^k \geq 0$$

Now let's take the dual of (P-C). Let b_t^i, l_{st}^i, v_s^k and w_s^i be the dual variables

corresponding to first, second, third and forth set of constraints in (P-C), respectively. The dual of the (P-C) is:

$$\text{maximize} \quad \sum_{i=1}^{N}\sum_{t=1}^{T} b_t^i \tag{D-C}$$

$$\text{subject to} \quad b_t^i \leq H_{st}^i + l_{st}^i, \qquad i = 1,..,N, t = 1,..,T, s = 1,..,t \tag{3.16}$$

$$v_s^k - v_s^{k-1} + \sum_{i=k}^{N} w_s^i \leq g(k) - g(k-1), \qquad k = 2,...,N-1, s = 1,...,T \tag{3.17}$$

$$-v_s^{N-1} + w_s^N \leq g(N) - g(N-1), \qquad s = 1,...,T \tag{3.18}$$

$$v_s^1 + \sum_{i=1}^{N} w_s^i \leq g(1), \qquad s = 1,...,T \tag{3.19}$$

$$\sum_{t=s}^{T} l_{st}^i - \sum_{k=i}^{n} w_s^k \geq 0, \qquad i = 1,...,N, s = 1,...,T \tag{3.20}$$

$$l_{st}^i, v_s^k, w_s^i \geq 0$$

Following the same strategy as was used in the previous section, we will use (D-C) to bound the holding cost incurred by our algorithm.

Next, we will show that (P-C) is equivalent to (P) in the case of the cardinality JRP.

Lemma 15. *For the cardinality JRP, (P-C) is equivalent to (P), in particular* $Z_{P-C} = Z_P$.

Proof. The proof follows the same outline as in Lemma 10. First we show how to convert an optimal solution $(\hat{x}, \hat{z}, \hat{q})$ of (P-C) to a solution (x, y) of (P). We first set $x = \hat{x}$. Next we set the values of y. Fix a time period s, and we first establish an ordering $s(i)$ of the items from the optimal solution of (P-C) so that $\hat{z}_s^{s(1)} \geq \hat{z}_s^{s(2)} \geq \ldots \geq \hat{z}_s^{s(n)}$. Next define the sequence of sets $S(i) := \{s(1), \ldots, s(i)\}, i = 1, \ldots n$. We let $y_s^{S(n)} = \hat{z}_s^{s(n)}$, $y_s^{S(i)} = \hat{z}_s^{s(i)} - \hat{z}_s^{s(i+1)}$ for $i = 1, \ldots, n-1$, and $y_s^S = 0$ for all other set S. It is easy to check that this solution is feasible for (P) and the objective value is the same by construction.

For the converse, we show that we can convert an optimal solution $(\hat{x}, \hat{y})$ to (P) to a solution (x, z, q) to (P-C) with the same cost. Let $(\hat{x}, \hat{y})$ satisfy the conditions of Lemma 9 , which implies that for each t, there is an ordering of N such that the only sets that can be positive are $\hat{y}_t^{\emptyset}, \hat{y}_t^{S_1}, \ldots, \hat{y}_t^{S_n}$ where $\emptyset \subset S_1 \subset S_2 \subset \ldots \subset S_n = N$. We first set $x = \hat{x}$. We now let $z_s^i = \max\{x_{st}^i : t = s \ldots T\}$ and $q_t^i = \sum_{k=i}^n \hat{y}_t^{S_k}$. One can check that all of the constraints of (P-C) are satisfied. The holding costs of the solutions are the same. The ordering cost of the the solution (x, z, q) is:

$$\begin{aligned}
\sum_{t=1}^{T}\sum_{i=1}^{n}(g(i)-g(i-1))q_t^k &= \sum_{t=1}^{T}\sum_{i=1}^{n}(g(i)-g(i-1))\sum_{k=i}^{n}\hat{y}_t^{S_k} \\
&= \sum_{t=1}^{T}\sum_{k=1}^{n}\sum_{i=1}^{k}(g(i)-g(i-1))\hat{y}_t^{S_k} \\
&= \sum_{t=1}^{T}\sum_{k=1}^{n}\hat{y}_t^{S_k}\sum_{i=1}^{k}(g(i)-g(i-1)) \\
&= \sum_{t=1}^{T}\sum_{k=1}^{n}\hat{y}_t^{S_k}g(k)
\end{aligned}$$

Thus the two solutions have equal cost. Since $(\hat{x}, \hat{y})$ was optimal for (P),

then the optimal cost of (P-C) at most that of (P), completing the proof of the converse. □

3.4.2 Algorithm

In this section we give a 5-approximation algorithm for the cardinality JRP. First, let $\hat{\mathbf{x}}, \hat{\mathbf{z}}$ and $\hat{\mathbf{q}}$ denote an optimal solution to the linear programming relaxation (P-C). Next we construct the corresponding solution (x, y) in (P) that gives the same solution for the cardinality JRP, as in the proof of the previous lemma. Specifically, we let $x = \hat{x}$ and set the values of y as follows. Fix a time period s, we first establish an ordering $s(i)$ of the items from the optimal solution of (P-C) so that $\hat{z}_s^{s(1)} \geq \hat{z}_s^{s(2)} \geq \ldots \geq \hat{z}_s^{s(n)}$. Next define the sequence of sets $S(i) := \{s(1), \ldots, s(i)\}, i = 1, \ldots n$. We let $y_s^{S(n)} = \hat{z}_s^{s(n)}$, $y_s^{S(i)} = \hat{z}_s^{s(i)} - \hat{z}_s^{s(i+1)}$ for $i = 1, \ldots, n-1$, and $y_s^S = 0$ for all other set S.

We make the problem "continuous" by spreading $\hat{z}_s^i$ in (P-C), and therefore y_t^S in (P), uniformly across the interval $[t, t+1)$, creating a density $f^i(\tau)$ for all items $i = 1..n$ and time $\tau \in [1, T)$. We define the set of half-points P_i for item i, as the set of all τ such that $\int_1^\tau f^i(s)\, ds = 0.5k$ for some $k \in \mathbb{Z}^+$. Let $P = \cup_{i=1}^n P_i$ be the set of all half-points, ordered from 1 to $|P|$ in increasing order. Let $p = |P|$. The j^{th} point corresponds to an interval $[a_j, b_j]$ and item type i_j such that $\int_{a_j}^{b_j} f^{i_j}(s)\, ds = 0.5$. We will give each j a label l_j in the algorithm. Also, we assume that we can order anywhere in $[1, T+1)$ and then round down.

Algorithm 3: Labeling Algorithm for cardinality JRP

Solve (P-C)
Generate order points P
for $k \leftarrow 1$ **to** N **do**
  for $j \leftarrow 1$ **to** p **do**
    if *j is unlabeled and $\exists$ an order of size $k-1$ and $\nexists$ an order of size k in* $[a_j, b_j]$ **then**
      Order i_j at the latest order of size $k-1$ in $[a_j, b_j]$
      set label $l_j = k$
Round any order at time τ to $\lfloor \tau \rfloor$
Serve every demand point (i, t) from the nearest order including item i

3.4.3 Analysis

We will use the labels of the orders made by the algorithm to bound its cost versus the optimal value of (P-C). We begin by proving several properties of the labeling algorithm.

Lemma 16. *Every $j \in P$ receives a label.*

Proof. Assume there exists $j \in P$ without a label. Let k be the size of the largest order in $[a_j, b_j]$. We know $k \neq N$ or else i_j is in the order and j was labeled. We know that this order existed in the $k + 1^{st}$ pass of the algorithm since we are incrementally building up orders. This implies that j should have been added to that order in that pass by construction, or another order of size k at a later time in its interval. Either way, j would get labeled, which is a contradiction. □

As in the analysis of the algorithm for tree JRP, we define the *active interval* for demand (i, t) be $[s_1, t]$, where s_1 is be the earliest time period where $x^i_{st} > 0$.

Lemma 17. *Every demand (i, t) is served from its active interval $[s_1, t]$*

Proof. Consider a demand (i, t) and its active interval $[s_1, t]$. From feasibility of x, we know that there must be at least 2 half-points in the interval $[s_1, t + 1)$. Therefore, by Lemma 16, we must have had an order i somewhere between these two half-points. Since the order is rounded down in time, we guarantee that the order is in $[s_1, t]$. □

Lemma 18. *For any time $\tau \in [1, T]$ and $k \in N$, there are at most 2 points in P whose intervals contain τ and are both labeled k.*

Proof. First we show that we cannot have two points j, j' with labels k such that $[a_j, b_j] \subseteq [a_{j'}, b_{j'}]$. If $b_j < b_{j'}$, then when we are processing j' in the k^{th} pass, there exists an order of size k in the interval for j'. Therefore, the algorithm would skip it. If $b_j = b_{j'}$, then there are two cases depending on what is processed first. If j is processed first, then the same logic holds. If j is processed second, then it must be the case that the order where $i_{j'}$ was added occurred in $[a_{j'}, a_j)$ or else there is an order size k in j's interval and we don't process it. However, since i_j was added to an order of size $k - 1$ in its interval, then it must be that j' could have been ordered at a later point in time which is what the algorithm mandates.

Now assume for contradiction that there are three orders j, j' and j'' with label k, all of which contain some τ in their interval. From the previous argument, we can assume that $a_j < a_{j'} < a_{j''} \leq \tau \leq b_j < b_{j'} < b_{j''}$. We can see that the order where i_j was added must occur in $[a_j, a_{j'})$, or else there would be a size k order in the interval of j', and thus it would get passed over in the k-labeling stage. Now consider the following two cases of where the order of j' occurred. If it occurs in $[a_{j'}, b_j)$, this implies that j did not order at the latest possible order of size $k - 1$. If the order of j' occurred within $[b_j, b_{j'}]$, this implies that j'' is

processed when there is an order of size k in its interval, and so the algorithm would skip it in the k^{th}. Therefore, we cannot give a label of k to j', which is a contradiction. □

Now we are ready to prove our main result.

Theorem 7. *The labeling algorithm is a 5-approximation algorithm for the cardinality JRP.*

Proof. We first bound the holding cost for each demand (i, t) by b_t^i. This implies the holding cost it at most 1 times the optimal cost, since the sum over all demand points gives the dual objective. By Lemma 17, (i, t) is served by an order s such that $x_{st}^i > 0$. From complementary slackness, this implies that $b_t^i = H_{st}^i + l_{st}^i$. Since $l_{st}^i \geq 0$, then the holding cost paid by (i, t) is $H_{st}^i \leq b_t^i$.

Now we proceed to bound the ordering cost of our solution. For every point j, we assigned it a label $l_j = k$ for some k. This means that item i_j is the k^{th} item added to some (unrounded) order in the interval $[a_j, b_j]$. Thus, this interval needs to pay $g(k) - g(k-1)$ in order to account for itself. We therefore want to take at most the k^{th} slice, $g(k) - g(k-1)$ of the ordering cost for the order corresponding to point j. We can use lower slices due to the concavity of g. If we are able to do this such that no slice gets counted for twice, then we ensure that we can pay for at least half the ordering costs, since the weight of every interval is 0.5 and for every interval an item is added to an order by Lemma 16. We will use a *charging scheme* on variables y_{tau}^S that has a positive density. More specifically, given a set S where $y_\tau^S > 0$, we give the cost $y_\tau^S g(1)$ to the item $i \in S$ with the smallest label at time τ, we give $y_\tau^S(g(2) - g(1))$ to the item $i \in S$ with the second smallest label at time τ, and so on. However, we show that each

slice will be used at most twice by Lemma 18. Thus, we need twice the optimal fractional ordering cost to pay for half of the marginal ordering cost incurred by any $j \in P$ in our solution, and so we need at most 4 times the optimal fractional ordering cost in total.

Combining the two bounds yields the result. □

CHAPTER 4

CONCLUSION AND OPEN PROBLEMS

In this thesis, we develop LP-based approximation algorithms for problems in scheduling and inventory management. In machine scheduling, we consider a general class of single-machine scheduling problem of minimizing the total cost summing over all jobs, and the only requirement on the cost function of each job is that it is a non-negative and non-decreasing function of its completion time. Using the primal-dual method, we give a simple algorithm for this problem that is guaranteed to return a solution that costs at most twice the optimal. To obtain this result, we add an exponential number of valid inequalities to strengthen the natural LP-relaxation, then design a primal-dual method that works on this exponential-sized LP. We then show how to modify our algorithm for scheduling problems where the processing speed of machine can vary over time. Finally, we show how this leads to a $(2+\epsilon)$-approximation algorithm for this problem, for any $\epsilon > 0$.

In inventory management, we consider two generalizations of the classical Joint Replenishment Problem (JRP): the tree JRP and the cardinality JRP. Using the LP-rounding technique, we give novel algorithms for the tree JRP and cardinality JRP that are guaranteed to generate a solution with cost within a factor of three and five, respectively, of the optimal.

Here we mention several open problems related to the work in this thesis:

- Is there an approximation algorithm with better performance guarantee for the general min-sum single machine scheduling problem ($1||\sum f_j$)? In particular, is a polynomial time approximation scheme (PTAS) possible?

- Is there constant factor approximation algorithm for the single machine min-sum scheduling problem with release dates, allowing preemption ($1|r_j, pmtn| \sum f_j$), or is such a result impossible? The current best result is a $O(\log \log nP)$-approximation algorithm by Bansal and Pruhs [9], where P is the sum of the processing time of all jobs. Here, the underlying geometric covering problem has an extra dimension. In particular, $1|r_j, pmtn| \sum f_j$ is modeled as a two-dimensional problem where we are covering the demand of various time interval (modeled as rectangles on the x-y plane with demand) using jobs completing at different times (modeled as rectangles with capacities). On the other hand, $1|| \sum f_j$ is a one-dimensional problem where we are covering the demand of various times (modeled as points on an axis) using jobs completing at different times (modeled as interval on the axis with capacities). There does not seem be a straight forward generalization of the primal-dual algorithm for $1|| \sum f_j$ in Chapter 2 to $1|r_j, pmtn| \sum f_j$.
- Can the result for the single-machine setting be generalized to the multiple machine setting, such as identical parallel machines ($P|pmtn| \sum f_j$)?
- Can the implementation of the primal-dual algorithm be improved? In particular, can we use advanced data structures to speed up the selection of dual variable in each iteration of the growing phase? For specific objective functions, such as weighted tardiness, can we use some structural properties of the objective function to speed up the primal-dual algorithm?
- Can the performance guarantee for the tree JRP and cardinality JRP be improved? In particular, the analysis of ordering cost for cardinality JRP might not be tight: we proved the ordering cost is at most four times the optimal cost, whereas the holding cost is at most the optimal cost.

- Does there exist a constant-factor approximation algorithm for submodular JRP, or is such result impossible? In the LP-rounding algorithms we give for tree JRP and cardinality JRP, we were able to determine the orders for each item type independently and charge the ordering cost using the LP optimal solution. There does not seem to be a straight forward generalization of these ideas when the ordering cost is given by an arbitrary submodular function.

BIBLIOGRAPHY

[1] Foto N. Afrati, Evripidis Bampis, Chandra Chekuri, David R. Karger, Claire Kenyon, Sanjeev Khanna, Ioannis Milis, Maurice Queyranne, Martin Skutella, Clifford Stein, and Maxim Sviridenko. Approximation schemes for minimizing average weighted completion time with release dates. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, pages 32–44, 1999.

[2] Ajit Agrawal, Philip Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized Steiner problem on networks. *SIAM Journal on Computing*, 24:440–456, 1995.

[3] E. Arkin, D. Joneja, and R. Roundy. Computational complexity of uncapacitated multi-echelon production planning problems. *Operations Research Letters*, 8(2):61–66, 1989.

[4] K. R. Baker, Eugene L. Lawler, Jan Karel Lenstra, and A. H. G. Rinnooy Kan. Preemptive scheduling of a single machine to minimize maximum cost subject to release dates and precedence constraints. *Operations Research*, 31:381–386, 1983.

[5] Nikhil Bansal, Niv Buchbinder, and Joseph Naor. A primal-dual randomized algorithm for weighted paging. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, pages 507–517, 2007.

[6] Nikhil Bansal, Niv Buchbinder, and Joseph Naor. Randomized competitive algorithms for generalized caching. In *Proceedings of the 40th Annual ACM Symposium on the Theory of Computing*, pages 235–244, 2008.

[7] Nikhil Bansal, Anupam Gupta, and Ravishankar Krishnaswamy. A constant factor approximation algorithm for generalized min-sum set cover. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1539–1545, 2010.

[8] Nikhil Bansal and Subhash Khot. Optimal long code test with one free bit. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 453–462, 2009.

[9] Nikhil Bansal and Kirk Pruhs. The geometry of scheduling. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science*, pages 407–414, 2010.

[10] R. Bar-Yehuda and S. Even. A linear time approximation algorithm for the weighted vertex cover problem. *Journal of Algorithms*, 2:198–203, 1981.

[11] J. E. Beasley. Or library, distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41:1069–1072, 1990.

[12] Dimitris Bertsimas and John N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, Belmont, Massachusetts, 1997.

[13] Tim Carnes and David Shmoys. Primal-dual schema for capacitated covering problems. In Andrea Lodi, Alessandro Panconesi, and Giovanni Rinaldi, editors, *Integer Programming and Combinatorial Optimization*, number 5035 in Lecture Notes in Computer Science, pages 288–302, 2008.

[14] Robert D. Carr, Lisa Fleischer, Vitus J. Leung, and Cynthia A. Phillips. Strengthening integrality gaps for capacitated network design and covering problems. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 106–115, 2000.

[15] L.M.A. Chan, A. Muriel, Z.J.M. Shen, D. Simchi-Levi, and C.P. Teo. Effective zero-inventory-ordering policies for the single-warehouse multiretailer problem with piecewise linear cost structures. *Management Science*, pages 1446–1460, 2002.

[16] Vašek Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4:233–235, 1979.

[17] Richard Congram, Chris Potts, and Steef van de Velde. An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing*, 14.

[18] Leah Epstein, Asaf Levin, Alberto Marchetti-Spaccamela, Nicole Megow, Julián Mestre, Martin Skutella, and Leen Stougie. Universal sequencing on a single machine. In *Proceedings of the 14th Conference on Integer Programming and Combinatorial Optimization*, number 6080 in Lecture Notes in Computer Science, pages 230–243, 2010.

[19] P. Erdős. Gráfok páros körüljárású részgráfjairól (On bipartite subgraphs of graphs, in Hungarian). *Mat. Lapok*, 18:283–288, 1967.

[20] A. Federgruen and M. Tzur. The joint replenishment problem with time-

varying costs and demands: Efficient, asymptotic and ε-optimal solutions. *Operations Research*, pages 1067–1086, 1994.

[21] A. Federgruen and Y.S. Zheng. The joint replenishment problem with general joint cost structures. *Operations Research*, 40(2):384–403, 1992.

[22] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, NY, 1979.

[23] Michel X. Goemans. Improved approximation algorithms for scheduling with release dates. In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 591–598, 1997.

[24] Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24:296–317, 1995.

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

[26] R.L. Graham. Bounds for certain multiprocessor anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[27] Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Danny Segev. Scheduling with outliers. In *Proceedings of APPROX-RANDOM*, pages 149–162, 2009.

[28] Leslie A. Hall, Andreas S. Schulz, David B. Shmoys, and Joel Wein. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Mathematics of Operations Research*, 22:513–544, 1997.

[29] Dorit S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal on Computing*, 11:555–556, 1982.

[30] Dorit S. Hochbaum, editor. *Approximation algorithms for NP-hard problems*. PWS Publishing Company, 1997.

[31] David S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.

[32] D. Joneja. The joint replenishment problem: new heuristics and worst case performance bounds. *Operations Research*, 38(4):711–723, 1990.

[33] E.P.C. Kao. A multi-product dynamic lot-size model with individual and joint set-up costs. *Operations Research*, pages 279–289, 1979.

[34] Narendra Karmarkar and Richard M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science*, pages 312–320, 1982.

[35] Jon Kleinberg and Éva Tardos. *Algorithm Design*. Pearson Education, Boston, Massachusetts, 2006.

[36] Eugene L. Lawler. Optimal sequencing of a single machine subject to precedence constraints. *Management Science*, 19:544–546, 1973.

[37] R. Levi, T.L. Magnanti, E.J. Zarybnisky, et al. *Maintenance scheduling for modular systems-models and algorithms*. PhD thesis, Massachusetts Institute of Technology, 2011.

[38] R. Levi, R. Roundy, D. Shmoys, and M. Sviridenko. A Constant Approximation Algorithm for the One-Warehouse Multiretailer Problem. *Management Science*, 54(4):763, 2008.

[39] R. Levi, R.O. Roundy, and D.B. Shmoys. Primal-Dual Algorithms for Deterministic Inventory Problems. *Mathematics of Operations Research*, 31(2):267–284, 2006.

[40] 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.

[41] M. W. Padberg, T. J. van Roy, and L. A. Wolsey. Valid inequalities for fixed charge problems. *Operations Research*, 33:842–861, 1985.

[42] David Pritchard. Approximability of sparse integer programs. In *Proceedings of the 17th Annual European Symposium on Algorithms*, pages 83–94, 2009.

[43] P. Raghavan and C.D. Thompson. Randomized rounding: a technique for

provably good algorithms and algorithmic proofs. *Combinatorica*, 7:365–374, 1987.

[44] R. Roundy. 98%-effective integer-ratio lot-sizing for one-warehouse multi-retailer systems. *Management science*, pages 1416–1430, 1985.

[45] David B. Shmoys and Éva Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62:461–474, 1993.

[46] G. Stauffer, G. Massonnet, C. Rapine, and J.P. Gayon. A simple and fast 2-approximation algorithm for the one-warehouse multi-retailers problem. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2011.

[47] C.P. Teo and D. Bertsimas. Multistage lot sizing problems via randomized rounding. *Operations Research*, pages 599–608, 2001.

[48] Vijay V. Vazirani. *Approximation Algorithms*. Springer, Berlin, Germany, second edition, 2004.

[49] A.F. Veinott Jr. Minimum concave-cost solution of leontief substitution models of multi-facility inventory systems. *Operations Research*, pages 262–291, 1969.

[50] V. G. Vizing. On an estimate of the chromatic class of a p-graph (in Russian). *Diskretnyĭ Analiz*, 3:25–30, 1964.

[51] H.M. Wagner and T.M. Whitin. Dynamic Version of the Economic Lot Size Model. *Management Science*, 5(1):89–96, 1958.

[52] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

[53] W.I. Zangwill. A backlogging model and a multi-echelon model of a dynamic economic lot size production system-a network approach. *Management Science*, pages 506–527, 1969.