SOME RESOURCE ALLOCATION 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

Chaoxu Tong

February 2016

SOME RESOURCE ALLOCATION PROBLEMS

Chaoxu Tong, Ph.D.

Cornell University 2016

We include three works on resource allocation in this thesis. Proper pricing helps the system to allocate resource efficiently. However, the computational efforts to obtain such prices are not always easy. In airline network revenue setting, high-quality bid prices can be obtained via approximate dynamic programming approach. However, this leads to an exponential-size linear program which is slow to solve. We show that this large linear program is actually equivalent to a compact linear program with appealing interpretation by exploiting the minimum-cost network flow structure of this problem. Computational experiments indicate that our results can speed up the computation for the approximate linear programming approach by a factor ranging between 13 and 135.

In the second chapter, we study how to better serve clients by allocating resources to open facilities with right location and type. We extend the classical *facility location* problem by associating *types* with the facilities and the clients. The types define a partial ordering and a client can only be served by facilities of equal or higher type. We show how to obtain approximation algorithms for two interesting special cases. We also study the problem embedded in time, which is known as the lot-sizing variant. We give algorithm that finds optimal solution for the variant.

In the third chapter, we study how to allocate medical resource in imaging facilities. The most common method to allocate imaging machine time is by a

appointment system. However, despite the best efforts by medical staff, the inherently stochastic nature of the duration of various procedures and unexpected disruptions can bring considerate congestion to the system. In the multi-site setting, we seek to complement appointment scheduling with real-time resource sharing. By monitoring system status and making prediction of future, we can identify potential congestions and divert patients accordingly to prevent these congestions from happening. We provide simulation results showing that this method improves patient waiting time and facilities overtime, which are two important quality measurements.

BIOGRAPHICAL SKETCH

Chaoxu grew up in a small town, Tongling, Anhui in China. He enjoys cooking and movies.

To my parents and my wife.

ACKNOWLEDGEMENTS

A big thanks to David and Shane for being extremely kind and patient advisors. Also, I would like to thank all committee members for providing invaluable feedbacks on this thesis. I am also deeply grateful for the support from my wife and my parents. Thanks all friends I met here at Ithaca for sharing precious time with me. Also thank the passing time for giving me refreshed understanding of myself.

The work in this thesis is supported in part by NSF grants CMMI-0825004, CMII-0969113, CCF-1526067 and CCF-1017688.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

FAST ALP APPROACH FOR NETWORK REVENUE MANAGEMENT PROBLEMS

1.1 Introduction

Bid prices form a powerful tool to obtain high-quality control policies for network revenue management problems. The fundamental idea is to associate a bid price with each flight leg in the airline network, characterizing the opportunity cost of a seat on the flight leg. In this case, an itinerary is opened for sale to customers if the revenue that would be obtained from the sale of the itinerary exceeds the total opportunity cost of the seats consumed by the itinerary; see [42] and [36]. It is known that bid price policies are not necessarily optimal, but their ease of implementation and intuitive appeal make them a popular choice in practice.

Since the bid price characterizes the opportunity cost of a seat on a flight leg, it is natural to expect that the bid price should depend on how much time is left until the departure time. As the departure time of a flight leg approaches, there are fewer opportunities to utilize the capacities and one would intuitively expect that the bid price of a flight leg would decrease, all else being equal. Despite this intuitive expectation, numerous popular models in the literature provide static bid prices that do not depend on how much time is left until departure; see, for example, [36] and [37]. As the sales take place and the departure time approaches, one has to periodically resolve these models to get bid prices that actually change as a function of the remaining time until departure. Recently, [1] proposes a model that naturally provides bid prices that depend on time until departure. This model starts with the dynamic programming formulation of the network revenue management problem, which is difficult to solve since the state variable in this formulation is a high-dimensional vector. The idea is to approximate the value functions with affine functions and choose the slopes and intercepts of the affine approximations by plugging them into the linear programming representation of the dynamic program. The number of decision variables in this linear program is manageable, but the number of constraints grows exponentially with the number of flight legs. Therefore, constraint generation is a natural method for solving the linear program. The slopes of the affine value function approximations are ultimately used as bid prices.

This model is quite appealing as it has theoretical footing in dynamic programming theory and generates bid prices that depend on time left until departure, but its implementation in practice can be difficult. To begin with, solving the linear program by using constraint generation requires iteratively generating constraints and each constraint can be generated by solving a separate integer program. This integer program is a drawback when applying the model to large airline networks. Furthermore, constraint generation methods are known to be slow in obtaining the optimal solution; see [31]. Finally, implementing constraint generation requires a good deal of customized software development and this negatively affects the practical appeal of the model.

In this chapter, we consider the network revenue management model proposed by [1] and eliminate essentially all of the drawbacks that hinder its practical implementation. First, we show that the integer program that needs to be solved for generating constraints can actually be formulated as a minimum-cost network flow problem whose continuous relaxation has no integrality gap. This result holds for an arbitrary airline network topology, allowing us to generate constraints very quickly even for large and complicated airline networks. The minimum-cost network flow formulation of the constraint generation problem is of interest by itself, but we note that constraint generation may still be slow to ultimately obtain the optimal solution, although we can generate each constraint quickly. Second, the number of constraints in the linear program used by [1] grows exponentially with the number of flight legs in the airline network. By using the minimum-cost network flow structure of the constraint generation problem, we a priori reduce this linear program to an equivalent linear program whose size grows only linearly with the numbers of flight legs and itineraries. The reduced linear program completely eliminates the need to solve separate problems to generate constraints. Third, we give an appealing intuitive interpretation for the reduced linear program that clearly shows its relationship to the network revenue management problem we want to solve. This interpretation is likely to enhance the appeal of the model to practitioners. Fourth, our computational experiments indicate that by using the results in this chapter, we can solve the model proposed by [1] up to 135 times faster. Over all of our test problems, the average speed up factor is about 52.

There are a number of models in the literature that are used to compute bid prices. [34] and [42] compute static bid prices by using a deterministic approximation to the network revenue management problem that is formulated under the assumption that all itinerary requests take on their expected values. In the deterministic approximation, there is a capacity constraint for each flight leg, ensuring that the served itinerary requests do not violate the leg capacities. The optimal values of the dual variables associated with these capacity constraints are used as bid prices. [36] gives an analysis of the bid prices obtained by this deterministic approximation. [37] introduces randomness into the deterministic approximation by using samples of the itinerary requests, rather than expected values. [14] and [28] use piecewise linear approximations to the value functions in the dynamic programming formulation of the problem and the policies obtained from their approaches can be viewed as bid price policies. [43] extends the model proposed by [1] to cover the case where each customer makes a choice among the itineraries that are open for sale. [38] computes bid prices that depend on both how much time is left until departure and how much capacity is left on the flight legs, but his approach is more computationally intensive than that of [1]. [22] uses Lagrangian relaxation to relax the capacity constraints in the dynamic programming formulation. The relaxed dynamic program can be solved efficiently and ultimately yields bid prices that depend on time.

The rest of the chapter is organized as follows. In Section 1.2, we begin by formulating the network revenue management problem as a dynamic program. In Section 1.3, we give the linear programming representation of this dynamic program. To obtain tractable approximations to the value functions, we replace the value functions in the linear programming representation by affine approximations. In this case, we obtain a linear programming representation with manageable number of decision variables, but the number of constraints grows exponentially with the number of flight legs. In Section 1.4, we describe how to solve the linear programming representation by using constraint generation and show that each constraint can be generated by solving a minimum-cost network flow problem. In Section 1.5, we exploit the minimum-cost network flow structure to a priori reduce the linear programming representation into a linear program whose numbers of constraints and decision variables grow only linearly with the numbers of flight legs and itineraries. In Section 1.6, we give a practical interpretation of this reduced linear program. In Section 1.7, we present computational experiments that demonstrate the computational benefits from the reduced linear program. In Section 1.8, we conclude.

1.2 Problem Formulation

We have a set of flight legs over an airline network that can be used to serve the requests for itineraries that arrive randomly over time. At the beginning of each time period, we decide whether each itinerary is open for sale or closed. If there is a request for an itinerary that is open for sale, then we serve this itinerary request, generating a revenue and consuming capacities on the flight legs that are included in the requested itinerary. A request for a closed itinerary simply leaves the system. The objective is to find a policy to open the itineraries for sale or close them over time so as to maximize the total expected revenue from the served itinerary requests.

We let $\mathcal{L}$ be the set of flight legs in the airline network and $\mathcal{J}$ be the set of itineraries. If we serve a request for itinerary j, then we generate a revenue of f_j and consume a_{ij} units of capacity on flight leg i. We assume that there are no group reservations so that each served itinerary request consumes at most one unit of capacity on a flight leg. In other words, we have $a_{ij} \in \{0, 1\}$ for all $i \in \mathcal{L}$, $j \in \mathcal{J}$. Thus, an itinerary j is characterized by the set of flight legs $\{i \in \mathcal{L} : a_{ij} = 1\}$ that it uses and the revenue f_j that it generates. In certain revenue management settings, an itinerary is referred to as a product. The available capacity on flight leg i is c_i. The itinerary requests arrive one by one over the time periods $\mathcal{T} = \{1, \ldots, \tau\}$. Arrivals of the itinerary requests at different time periods are independent. The probability that there is a request for itinerary j at time period t is p_{jt}. We assume that $\sum_{j \in \mathcal{J}} p_{jt} = 1$ so that there is exactly one itinerary request at each time period. If there is a positive probability of having no itinerary request at time period t, then we can capture this situation by defining a dummy itinerary ϕ satisfying $f_\phi = 0$, $a_{i\phi} = 0$ for all $i \in \mathcal{L}$ and $p_{\phi t} = 1 - \sum_{j \in \mathcal{J}} p_{jt}$.

We use x_{it} to denote the remaining capacity on flight leg i at the beginning of time period t so that the vector $x_t = \{x_{it} : i \in \mathcal{L}\}$ captures the state of the capacities on the flight legs at the beginning of this time period. We let u_{jt} take value one if we open itinerary j for sale at time period t and take value zero otherwise. The vector $u_t = \{u_{jt} : j \in \mathcal{J}\}$ captures the decisions at time period t. The set of feasible decisions is given by $\mathcal{U}(x_t) = \{u_t \in \{0,1\}^{|\mathcal{J}|} : a_{ij}\, u_{jt} \leq\ x_{it}\ \forall\, i \in \mathcal{L},\ j \in \mathcal{J}\}$, ensuring that if we want to open itinerary j for sale and itinerary j uses flight leg i, then there has to be capacity available on flight leg i. In this case, we can formulate the problem as a dynamic program as

$$V_t(x_t) = \max_{u_t \in \mathcal{U}(x_t)} \left\{ \sum_{j \in \mathcal{J}} p_{jt} \left\{ f_j\, u_{jt} + V_{t+1}(x_t - u_{jt} \textstyle\sum_{i \in \mathcal{L}} a_{ij}\, e_i) \right\} \right\}, \tag{1.1}$$

where we use e_i to denote the $|\mathcal{L}|$-dimensional unit vector with a one in the element corresponding to flight leg i. The boundary condition of the optimality equation above is $V_{\tau+1}(\cdot) = 0$ and the state space can be written as $\mathcal{X} = \prod_{i \in \mathcal{L}} \{0, 1, \ldots, c_i\}$. Assuming that we have access to the value functions $\{V_t(x_t) : x_t \in \mathcal{X},\ t \in \mathcal{T}\}$, it is not difficult to compute the optimal decisions at each time period. If we have $a_{ij} \leq x_{it}$ for all $i \in \mathcal{L}$ and $f_j + V_{t+1}(x_t - \sum_{i \in \mathcal{L}} a_{ij}\, e_i) \geq V_{t+1}(x_t)$, then it is optimal to open itinerary j for sale at time period t when the remaining capacities on the flight legs are given by the vector x_t. Otherwise, it is optimal to close itinerary j.

The size of the state space $\mathcal{X}$ in the optimality equation in (1.1) grows exponentially with the number of flight legs, rendering the computation of the value functions intractable for essentially any practical airline network. In this chapter, we focus on an approximate solution method proposed by [1]. We describe this solution method in the next section.

1.3 Approximate Linear Program

It is a standard result in Markov decision processes that an optimality equation with finite sets of states and decisions can be formulated as a linear program; see [30]. In general, this linear program is not computationally useful by itself since its numbers of decision variables and constraints are proportional to the number of possible states, but the linear program can serve as a starting point for constructing approximations to the value functions. Using c to denote the vector of initial capacities $\{c_i : i \in \mathcal{L}\}$, the linear program corresponding to the optimality equation in (1.1) can be written as

$$\min \quad \vartheta_1(c) \tag{1.2a}$$

$$\text{subject to} \quad \vartheta_t(x_t) \geq \sum_{j \in \mathcal{J}} p_{jt} \Big\{ f_j\, u_{jt} + \vartheta_{t+1}(x_t - u_{jt} \textstyle\sum_{i \in \mathcal{L}} a_{ij}\, e_i) \Big\} \qquad \forall\, x_t \in \mathcal{X},\ u_t \in \mathcal{U}(x_t),\ t \in \mathcal{T}, \tag{1.2b}$$

where the decision variables are $\{\vartheta_t(x_t) : x_t \in \mathcal{X},\ t \in \mathcal{T}\}$. For notational uniformity, we follow the convention that the values of the decision variables $\{\vartheta_{\tau+1}(x_{\tau+1}) : x_{\tau+1} \in \mathcal{X}\}$ in the problem above are set to zero. It is possible to show that the optimal objective value of problem (1.2) is equal to the optimal total expected revenue $V_1(c)$ that is obtained through the optimality equation in (1.1). Therefore, the objective function of problem (1.2) evaluated at any feasible solution provides an upper bound on the optimal total expected revenue.

Problem (1.2) is not computationally useful by itself since the numbers of decision variables and constraints in this problem increase exponentially with the number of flight legs. To deal with this difficulty, [1] proposes approximating the value function $V_t(x_t)$ with an affine function of the form $\tilde{V}_t(x_t) = \theta_t + \sum_{i \in \mathcal{L}} v_{it}\, x_{it}$ for all $t \in \mathcal{T}$, where $\{\theta_t : t \in \mathcal{T}\}$ and $\{v_{it} : i \in \mathcal{L},\ t \in \mathcal{T}\}$ are adjustable parameters. To choose a set of values for $\{\theta_t : t \in \mathcal{T}\}$ and $\{v_{it} : i \in \mathcal{L},\ t \in \mathcal{T}\}$, we plug the approximation $\tilde{V}_t(x_t) = \theta_t + \sum_{i \in \mathcal{L}} v_{it}\, x_{it}$ into problem (1.2) to obtain the linear program

$$\min \quad \theta_1 + \sum_{i \in \mathcal{L}} v_{i1}\, c_i \tag{1.3a}$$

$$\text{subject to} \quad \theta_t + \sum_{i \in \mathcal{L}} v_{it}\, x_{it} \geq \sum_{j \in \mathcal{J}} p_{jt} \left\{ f_j\, u_{jt} + \theta_{t+1} + \sum_{i \in \mathcal{L}} v_{i,t+1}\, [x_{it} - u_{jt}\, a_{ij}] \right\}$$
$$\forall\, x_t \in \mathcal{X},\ u_t \in \mathcal{U}(x_t),\ t \in \mathcal{T}, \tag{1.3b}$$

where the decision variables are $\{\theta_t : t \in \mathcal{T}\}$ and $\{v_{it} : i \in \mathcal{L},\ t \in \mathcal{T}\}$. Similar to problem (1.2), we follow the convention that the values of the decision variables $\theta_{\tau+1}$ and $\{v_{i,\tau+1} : i \in \mathcal{L}\}$ in the problem above are set to zero. The number of constraints in problem (1.3) still increases exponentially with the number of flight legs, but the number of decision variables is $|\mathcal{T}| + |\mathcal{L}|\,|\mathcal{T}|$, which can be manageable for airline networks of practical significance. Therefore, it can be possible to solve problem (1.3) by using constraint generation for practical airline networks.

There are two uses of problem (1.3). First, it is possible to show that the optimal objective value of this problem provides an upper bound on the optimal total expected revenue. In particular, letting $\{\theta_t^* : t \in \mathcal{T}\}$ and $\{v_{it}^* : i \in \mathcal{L},\ t \in \mathcal{T}\}$ be an optimal solution to problem (1.3) and $\hat{\vartheta}_t(x_t) = \theta_t^* + \sum_{i \in \mathcal{L}} v_{it}^*\, x_{it}$ for all $x_t \in \mathcal{X}$,

$t \in \mathcal{T}$, we observe that $\{\hat{\vartheta}_t(x_t) : x_t \in \mathcal{X},\ t \in \mathcal{T}\}$ is a feasible solution to problem (1.2). Since the objective function of problem (1.2) evaluated at a feasible solution provides an upper bound on the optimal total expected revenue, it immediately follows that $\hat{\vartheta}_1(c) = \theta_1^* + \sum_{i \in \mathcal{L}} v_{i1}^* \, c_i$ is an upper bound on the optimal total expected revenue. Such an upper bound on the optimal total expected revenue becomes useful when we try to assess the optimality gap of approximate or heuristic control policies. Second, we can use an optimal solution to problem (1.3) to construct an approximate control policy. Recalling that $\{\theta_t^* : t \in \mathcal{T}\}$ and $\{v_{it}^* : i \in \mathcal{L},\ t \in \mathcal{T}\}$ denote an optimal solution to problem (1.3), the idea is to use $\tilde{V}_t^*(x_t) = \theta_t^* + \sum_{i \in \mathcal{L}} v_{it}^* \, x_{it}$ as an approximation to $V_t(x_t)$. In this case, if we have $a_{ij} \leq x_{it}$ for all $i \in \mathcal{L}$ and $f_j + \theta_{t+1}^* + \sum_{i \in \mathcal{L}} v_{i,t+1}^* \, [x_{it} - a_{ij}] \geq \theta_{t+1}^* + \sum_{i \in \mathcal{L}} v_{i,t+1}^* \, x_{it}$, then we open itinerary j for sale at time period t when the remaining capacities on the flight legs are given by the vector x_t. To gain some insight into this policy, we write the last inequality as

$$f_j \geq \sum_{i \in \mathcal{L}} a_{ij} \, v_{i,t+1}^*.$$

We interpret v_{it}^* in the value function approximation $\tilde{V}_t^*(x_t) = \theta_t^* + \sum_{i \in \mathcal{L}} v_{it}^* \, x_{it}$ as the marginal value or the opportunity cost of a unit of capacity on flight leg i at time period t, in which case, the right side of the inequality above corresponds to the total value of the capacities used by itinerary j. Therefore, the control policy obtained from problem (1.3) compares the revenue from itinerary j with the total value of the capacities consumed by itinerary j and it opens itinerary j for sale if the revenue justifies the total value of the consumed capacities, as long as there are enough seats.

In network revenue management, the marginal value or the opportunity cost of a seat is referred to as the bid price. Intuitively, the bid price of a flight leg, being the marginal value of a unit of capacity, should depend on how much time is left until the time of departure. An attractive feature of problem (1.3) is that it naturally provides bid prices that are indeed dependent on time. Throughout the rest of the chapter, we focus on efficient solution methods for problem (1.3).

1.4 Constraint Generation

Noting that the number of decision variables in problem (1.3) is manageable but the number of constraints grows exponentially with the number of flight legs, a possible solution method for this problem is to use constraint generation. The idea behind constraint generation is to iteratively solve a master problem that has the same decision variables as problem (1.3), but has only a subset of the constraints. After solving the master problem, we check whether any one of the constraints in (1.3b) is violated by the solution to the current master problem. If there is one such constraint, then we add this constraint to the master problem and resolve the master problem. Otherwise, the solution to the current master problem is optimal to problem (1.3) and we stop.

The key to efficient implementation of constraint generation is to be able to check whether the solution to the current master problem violates any one of the constraints in (1.3b). Using the fact that $\sum_{j \in \mathcal{J}} p_{jt} = 1$, we write constraints (1.3b) as

$$\theta_t - \theta_{t+1} \geq \sum_{j \in \mathcal{J}} p_{jt} \Big\{ f_j - \sum_{i \in \mathcal{L}} a_{ij}\, v_{i,t+1} \Big\}\, u_{jt} + \sum_{i \in \mathcal{L}} \left[v_{i,t+1} - v_{it} \right] x_{it}$$

for all $x_t \in \mathcal{X}$, $u_t \in \mathcal{U}(x_t)$, $t \in \mathcal{T}$. In this case, if we let $\{\tilde{\theta}_t : t \in \mathcal{T}\}$ and $\{\tilde{v}_{it} : i \in \mathcal{L},\ t \in \mathcal{T}\}$ be the solution to the current master problem, then we can check whether this solution violates any one of the constraints in (1.3b) by solving

$$\max_{x_t \in \mathcal{X},\ u_t \in \mathcal{U}(x_t)} \left\{ \sum_{j \in \mathcal{J}} p_{jt} \left\{ f_j - \sum_{i \in \mathcal{L}} a_{ij}\, \tilde{v}_{i,t+1} \right\} u_{jt} + \sum_{i \in \mathcal{L}} \left[\tilde{v}_{i,t+1} - \tilde{v}_{it} \right] x_{it} \right\} \tag{1.4}$$

for all $t \in \mathcal{T}$. If the optimal objective value of the problem above exceeds $\tilde{\theta}_t - \tilde{\theta}_{t+1}$ for a particular time period t, then letting $\tilde{x}_t$ and $\tilde{u}_t$ be the optimal solution to problem (1.4), the constraint corresponding to $\tilde{x}_t \in \mathcal{X}$, $\tilde{u}_t \in \mathcal{U}(\tilde{x}_t)$ and $t \in \mathcal{T}$ in problem (1.3) is violated by the solution to the current master problem. We add this constraint to the current master problem and solve the master problem with the added constraint. Therefore, efficient implementation of constraint generation is dependent on our ability to solve problem (1.4) quickly.

Noting the definitions of the state space $\mathcal{X}$ and the set of feasible decisions $\mathcal{U}(x_t)$, the decision variables x_t and u_t in problem (1.4) have integrality constraints. [1] notes that we can relax the integrality constraints on the decision variables x_t without loss of optimality, but we still have the integrality constraints on the decision variables u_t in problem (1.4). Due to such integrality constraints, problem (1.4) is recognized as a drawback when using constraint generation to solve problem (1.3) for large airline networks. In this section, we alleviate such concerns by showing that the continuous relaxation of problem (1.4) has no integrality gap. Therefore, we can generate constraints by solving the continuous relaxation of problem (1.4), which can be done efficiently for practical airline networks.

To show that the continuous relaxation of problem (1.4) has no integrality gap, we exploit the fact that the dual of this continuous relaxation turns out to be a minimum-cost network flow problem with integer cost data. Using the definitions of the state space $\mathcal{X}$ and the set of feasible decisions $\mathcal{U}(x_t)$, we write the continuous relaxation of problem (1.4) as

$$\max \quad \sum_{j \in \mathcal{J}} p_{jt} \Big\{ f_j - \sum_{i \in \mathcal{L}} a_{ij} \, \tilde{v}_{i,t+1} \Big\} u_{jt} + \sum_{i \in \mathcal{L}} [\tilde{v}_{i,t+1} - \tilde{v}_{it}] \, x_{it} \tag{1.5a}$$

$$\text{subject to} \quad x_{it} \le c_i \qquad \forall \, i \in \mathcal{L} \tag{1.5b}$$

$$a_{ij} \, u_{jt} \le x_{it} \qquad \forall \, i \in \mathcal{L}, \; j \in \mathcal{J} \tag{1.5c}$$

$$u_{jt} \le 1 \qquad \forall \, j \in \mathcal{J} \tag{1.5d}$$

$$x_{it} \text{ is free}, \; u_{jt} \ge 0 \qquad \forall \, i \in \mathcal{L}, \; j \in \mathcal{J}. \tag{1.5e}$$

We do not explicitly impose nonnegativity constraints on the decision variables $\{x_{it} : i \in \mathcal{L}\}$ in the problem above, since constraints (1.5c) along with the nonnegativity constraints on the decision variables $\{u_{jt} : j \in \mathcal{J}\}$ ensure that the decision variables $\{x_{it} : i \in \mathcal{L}\}$ are nonnegative. Our goal is to show that the dual of problem (1.5) is a minimum-cost network flow problem. To that end, we associate the dual variables $\{\alpha_{it} : i \in \mathcal{L}\}$, $\{\beta_{ijt} : i \in \mathcal{L}, \; j \in \mathcal{J}\}$, and $\{\gamma_{jt} : j \in \mathcal{J}\}$ respectively with constraints (1.5b), (1.5c) and (1.5d) in problem (1.5). In the dual of problem (1.5), the constraints associated with the decision variables $\{u_{jt} : j \in \mathcal{J}\}$ are given by $\sum_{i \in \mathcal{L}} a_{ij} \beta_{ijt} + \gamma_{jt} \ge p_{jt} [f_j - \sum_{i \in \mathcal{L}} a_{ij} \, \tilde{v}_{i,t+1}]$ for all $j \in \mathcal{J}$. Using the slack variables $\{\mu_{jt} : j \in \mathcal{J}\}$ for these constraints, the dual of problem (1.5) is given by

$$\min \quad \sum_{i \in \mathcal{L}} c_i \, \alpha_{it} + \sum_{j \in \mathcal{J}} \gamma_{jt} \tag{1.6a}$$

$$\text{subject to} \quad \alpha_{it} - \sum_{j \in \mathcal{J}} \beta_{ijt} = \tilde{v}_{i,t+1} - \tilde{v}_{it} \qquad \forall \, i \in \mathcal{L} \tag{1.6b}$$

$$\sum_{i \in \mathcal{L}} a_{ij} \beta_{ijt} + \gamma_{jt} - \mu_{jt} = p_{jt} \Big\{ f_j - \sum_{i \in \mathcal{L}} a_{ij} \, \tilde{v}_{i,t+1} \Big\} \qquad \forall \, j \in \mathcal{J} \tag{1.6c}$$

$$\alpha_{it} \ge 0, \; \beta_{ijt} \ge 0, \; \gamma_{jt} \ge 0, \; \mu_{jt} \ge 0 \qquad \forall \, i \in \mathcal{L}, \; j \in \mathcal{J}. \tag{1.6d}$$

The next proposition shows that problem (1.6) is a minimum-cost network flow problem.

Proposition 1.4.1. *Problem* (1.6) *is a minimum-cost network flow problem.*

Proof We establish the result by showing that all of the decision variables in problem (1.6) can be associated with the arcs in a directed network and all of the constraints in problem (1.6) correspond to the flow balance constraints of the nodes in this directed network. We consider a network with two sets of nodes $\mathcal{N}_1 = \{i : i \in \mathcal{L}\}$ and $\mathcal{N}_2 = \{j : j \in \mathcal{J}\}$ and an additional sink node. Constraints (1.6b) and (1.6c) are respectively the flow balance constraints for the nodes in $\mathcal{N}_1$ and $\mathcal{N}_2$. The flow balance constraint for the sink node is redundant and it is omitted in problem (1.6). The decision variable α_{it} in problem (1.6) corresponds to an arc that connects the sink node to node $i \in \mathcal{N}_1$. If $a_{ij} = 1$, then the decision variable β_{ijt} corresponds to an arc from node $i \in \mathcal{N}_1$ to node $j \in \mathcal{N}_2$. On the other hand, if $a_{ij} = 0$, then the decision variable β_{ijt} corresponds to an arc from node $i \in \mathcal{N}_1$ to the sink node. Therefore, the decision variable β_{ijt} does not appear in the flow balance constraint for node $j \in \mathcal{N}_2$ when $a_{ij} = 0$. The decision variable γ_{jt} corresponds to an arc from the sink node to node $j \in \mathcal{N}_2$. The decision variable μ_{jt} corresponds to an arc from node $j \in \mathcal{N}_2$ to the sink node. The demands of nodes $i \in \mathcal{N}_1$ and $j \in \mathcal{N}_2$ are respectively $\tilde{v}_{i,t+1} - \tilde{v}_{it}$ and $p_{jt}\,[f_j - \sum_{i \in \mathcal{L}} a_{ij}\,\tilde{v}_{i,t+1}]$. □

Figure 1 shows the structure of problem (1.6) for a small airline network. The upper portion of this figure shows the airline network. The flight legs are labeled by $\{1, 2\}$ and represented in solid arcs, whereas the itineraries are labeled by $\{a, b, c\}$ and represented in dotted arcs. The lower portion of the figure shows the minimum-cost network flow problem corresponding to problem (1.6). Each arc in Figure 1 corresponds to a decision variable in problem (1.6) and each node with the exception of the sink node corresponds to a constraint.

Proposition 1.4.1 immediately implies that problem (1.5) has no integrality gap. In particular, problems (1.5) and (1.6) form a primal-dual pair so that an optimal dual solution to problem (1.6) is an optimal solution to problem (1.5). Since problem (1.6) is a minimum-cost network flow problem with integer cost data, there exists an integer-valued optimal dual solution to problem (1.6) and this integer-valued optimal dual solution can be obtained by solving problem (1.6) by using the simplex algorithm; see [39]. Therefore, there exists an integer-valued optimal solution to problem (1.5), establishing that this problem has no integrality gap.

We do not make any assumptions in Proposition 1.4.1 on the structure of the underlying airline network, implying that problem (1.5) has no integrality gap for any airline network structure. Furthermore, we can solve the dual of problem (1.5) by using minimum-cost network flow algorithms, which are quite efficient. These observations indicate that constraint generation is efficient to implement even for large airline networks with arbitrary topologies. It is also worthwhile to observe that it is necessary to have $a_{ij} \in \{0, 1\}$ for all $i \in \mathcal{L}$, $j \in \mathcal{J}$ for Proposition 1.4.1 to hold. Otherwise, constraints (1.6c) cannot be interpreted as flow balance constraints.

The fact that problem (1.5) has no integrality gap is of interest by itself, but we also build on this result in the next section to show that we can reduce problem (1.3) to a linear program whose size grows linearly with the numbers of flight legs and itineraries.

1.5 Reducing the Number of Constraints

By the discussion in the previous section, if we use constraint generation to solve problem (1.3), then we can generate each constraint efficiently. However, although generating each constraint is efficient, we may still end up generating a large number of constraints to ultimately obtain the optimal solution by using constraint generation. In this section, we establish a complementary result, showing that we can a priori reduce problem (1.3) to a linear program whose numbers of constraints and decision variables grow only linearly with the numbers of flight legs and itineraries. This result significantly enhances the computational tractability of problem (1.3).

To facilitate our discussion, we use v to denote the collection of decision variables $\{v_{it} : i \in \mathcal{L},\ t \in \mathcal{T}\}$ in problem (1.3) and define $\Pi_t(v)$ as

$$\Pi_t(v) = \max_{x_t \in \mathcal{X},\ u_t \in \mathcal{U}(x_t)} \left\{ \sum_{j \in \mathcal{J}} p_{jt} \left\{ f_j - \sum_{i \in \mathcal{L}} a_{ij}\, v_{i,t+1} \right\} u_{jt} + \sum_{i \in \mathcal{L}} [v_{i,t+1} - v_{it}]\, x_{it} \right\}. \tag{1.7}$$

The notation $\Pi_t(v)$ suggests that the quantity on the right side above depends on the whole collection $v = \{v_{it} : i \in \mathcal{L},\ t \in \mathcal{T}\}$, but $\Pi_t(v)$ actually depends only on v_t and v_{t+1}. Our choice of notation for $\Pi_t(v)$ is motivated by notational brevity. With this definition of $\Pi_t(v)$, constraints (1.3b) in problem (1.3) can succinctly be written as $\theta_t - \theta_{t+1} \geq \Pi_t(v)$ for all $t \in \mathcal{T}$ and we can write problem (1.3) as

$$\min \quad \theta_1 + \sum_{i \in \mathcal{L}} v_{i1}\, c_i \tag{1.8a}$$

$$\text{subject to} \quad \theta_t - \theta_{t+1} \geq \Pi_t(v) \qquad \forall\, t \in \mathcal{T}, \tag{1.8b}$$

where the decision variables are $\{\theta_t : t \in \mathcal{T}\}$ and $\{v_{it} : i \in \mathcal{L},\ t \in \mathcal{T}\}$. We make two useful observations in problem (1.8). First, the values of the decision variables $\{\theta_t : t \in \mathcal{T}\}$ and $\{v_{it} : i \in \mathcal{L},\ t \in \mathcal{T}\}$ have to satisfy $\theta_t = \theta_{t+1} + \Pi_t(v)$ for all $t \in \mathcal{T}$ in an optimal solution to problem (1.8). Adding the last equality over all $t \in \mathcal{T}$ and noting the convention that the value of the decision variable $\theta_{\tau+1}$ is set to zero, it follows that $\theta_1 = \sum_{t\in\mathcal{T}} \Pi_t(v)$ in an optimal solution to problem (1.8). Second, we observe that problems (1.4) and (1.7) have the same structure. Therefore, Proposition 1.4.1 implies that the continuous relaxation of problem (1.7) has no integrality gap. By using the same argument that we use just before Proposition 1.4.1, we write the dual of the continuous relaxation of problem (1.7) as

$$\Pi_t(v) = \min \quad \sum_{i\in\mathcal{L}} c_i\, \alpha_{it} + \sum_{j\in\mathcal{J}} \gamma_{jt} \tag{1.9a}$$

$$\text{subject to} \quad \alpha_{it} - \sum_{j\in\mathcal{J}} \beta_{ijt} = v_{i,t+1} - v_{it} \qquad \forall\, i \in \mathcal{L} \tag{1.9b}$$

$$\sum_{i\in\mathcal{L}} a_{ij}\,\beta_{ijt} + \gamma_{jt} \geq p_{jt}\left\{f_j - \sum_{i\in\mathcal{L}} a_{ij}\, v_{i,t+1}\right\} \qquad \forall\, j \in \mathcal{J} \tag{1.9c}$$

$$\alpha_{it} \geq 0,\ \beta_{ijt} \geq 0,\ \gamma_{jt} \geq 0 \qquad \forall\, i \in \mathcal{L},\ j \in \mathcal{J}, \tag{1.9d}$$

in which case, the problem above and problem (1.7) share the same optimal objective value and this common optimal objective value is denoted by $\Pi_t(v)$.

In the next proposition, we build on the two observations above to reduce problem (1.3) to a linear program whose numbers of constraints and decision variables grow linearly with the numbers of flight legs and itineraries.

Proposition 1.5.1. *Problem* (1.3) *is equivalent to the problem*

$$\min \quad \sum_{t\in\mathcal{T}}\sum_{i\in\mathcal{L}} c_i\,\alpha_{it} + \sum_{t\in\mathcal{T}}\sum_{j\in\mathcal{J}} \gamma_{jt} + \sum_{i\in\mathcal{L}} c_i\, v_{i1} \tag{1.10a}$$

$$\text{subject to} \quad \alpha_{it} - \sum_{j\in\mathcal{J}} \beta_{ijt} = v_{i,t+1} - v_{it} \qquad \forall\, i\in\mathcal{L},\ t\in\mathcal{T} \tag{1.10b}$$

$$\sum_{i\in\mathcal{L}} a_{ij}\beta_{ijt} + \gamma_{jt} \geq p_{jt}\Big\{f_j - \sum_{i\in\mathcal{L}} a_{ij}\, v_{i,t+1}\Big\} \qquad \forall\, j\in\mathcal{J},\ t\in\mathcal{T} \tag{1.10c}$$

$$\alpha_{it}\geq 0,\ \beta_{ijt}\geq 0,\ \gamma_{jt}\geq 0,\ v_{it} \text{ is free} \qquad \forall\, i\in\mathcal{L},\ j\in\mathcal{J},\ t\in\mathcal{T}. \tag{1.10d}$$

In particular, problems (1.3) *and* (1.10) *have the same optimal objective value and given an optimal solution to one problem, we can construct an optimal solution to the other.*

Proof Since problems (1.3) and (1.8) are equivalent to each other, we show that problem (1.8) is equivalent to problem (1.10). We use Z^* and ζ^* to respectively denote the optimal objective values of problems (1.8) and (1.10). First, we show that $Z^* \leq \zeta^*$. Since the optimal objective value of problem (1.9) gives $\Pi_t(v)$, problem (1.10) is equivalent to minimizing $\sum_{t\in\mathcal{T}} \Pi_t(v) + \sum_{i\in\mathcal{L}} c_i\, v_{i1}$ over v, which implies that $\zeta^* = \min_v\{\sum_{t\in\mathcal{T}} \Pi_t(v) + \sum_{i\in\mathcal{L}} c_i\, v_{i1}\}$. So, letting v^* be the optimal solution to the last optimization problem and defining θ^*_t as $\theta^*_t = \sum_{s=t}^{\tau} \Pi_s(v^*)$, the solution $\{\theta^*_t : t\in\mathcal{T}\}$ and $v^* = \{v^*_{it} : i\in\mathcal{L},\ t\in\mathcal{T}\}$ is feasible to problem (1.8) and we obtain $Z^* \leq \sum_{t\in\mathcal{T}} \Pi_t(v^*) + \sum_{i\in\mathcal{L}} c_i\, v^*_{i1} = \zeta^*$.

Second, we show that $\zeta^* \leq Z^*$. We let $\{\theta^*_t : t\in\mathcal{T}\}$ and $v^* = \{v^*_{it} : i\in\mathcal{L},\ t\in\mathcal{T}\}$ be an optimal solution to problem (1.8). To construct a feasible solution to problem (1.10), we solve problem (1.9) for each $t\in\mathcal{T}$ after replacing the right side of constraints (1.9b) with $\{v^*_{i,t+1} - v^*_{it} : i\in\mathcal{L}\}$ and the right side of constraints (1.9c) with $\{p_{jt}\,[f_j - \sum_{i\in\mathcal{L}} a_{ij}\, v^*_{i,t+1}] : j\in\mathcal{J}\}$. Letting $\{\alpha^*_{it} : i\in\mathcal{L}\}$, $\{\beta^*_{ijt} : i\in\mathcal{L},\ j\in\mathcal{J}\}$ and $\{\gamma^*_{jt} : j\in\mathcal{J}\}$ be an optimal solution to problem (1.9) for each $t\in\mathcal{T}$, we have

$\Pi_t(v^*) = \sum_{i\in\mathcal{L}} c_i\, \alpha^*_{it} + \sum_{j\in\mathcal{J}} \gamma^*_{jt}$ for all $t \in \mathcal{T}$. Also, noting constraints (1.9b) and (1.9c), the solution $\{\alpha^*_{it} : i \in \mathcal{L},\ t \in \mathcal{T}\}$, $\{\beta^*_{ijt} : i \in \mathcal{L},\ j \in \mathcal{J}\ , t \in \mathcal{T}\}$, $\{\gamma^*_{jt} : j \in \mathcal{J},\ t \in \mathcal{T}\}$ and $\{v^*_{it} : i \in \mathcal{L},\ t \in \mathcal{T}\}$ is feasible to problem (1.10), but not necessarily optimal. Therefore, we obtain

$$\zeta^* \le \sum_{t\in\mathcal{T}} \sum_{i\in\mathcal{L}} c_i\, \alpha^*_{it} + \sum_{t\in\mathcal{T}} \sum_{j\in\mathcal{J}} \gamma^*_{jt} + \sum_{i\in\mathcal{L}} c_i\, v^*_{i1} = \sum_{t\in\mathcal{T}} \Pi_t(v^*) + \sum_{i\in\mathcal{L}} c_i\, v^*_{i1} = \theta^*_1 + \sum_{i\in\mathcal{L}} c_i\, v^*_{i1} = Z^*,$$

where the second equality uses the fact $\theta^*_1 = \sum_{t\in\mathcal{T}} \Pi_t(v^*)$ is always satisfied by an optimal solution to problem (1.8). Thus, we obtain $Z^* = \zeta^*$ and we can use the construction above to get an optimal solution to one of the problems (1.3) and (1.10) by using an optimal solution to the other. □

The proof of Proposition 1.5.1 shows that if $\{\alpha^*_{it} : i \in \mathcal{L},\ t \in \mathcal{T}\}$, $\{\beta^*_{ijt} : i \in \mathcal{L},\ j \in \mathcal{J},\ t \in \mathcal{T}\}$, $\{\gamma^*_{jt} : j \in \mathcal{J},\ t \in \mathcal{T}\}$ and $v^* = \{v^*_{it} : i \in \mathcal{L},\ t \in \mathcal{T}\}$ is an optimal solution to problem (1.10), then letting $\theta^*_t = \sum_{s=t}^{\tau} \Pi_s(v^*)$, the solution $\{\theta^*_t : t \in \mathcal{T}\}$ and $\{v^*_{it} : i \in \mathcal{L},\ t \in \mathcal{T}\}$ is optimal to problem (1.3). Therefore, we can recover an optimal solution to problem (1.3) from problem (1.10).

1.6 Practical Interpretation

At first glance, it is difficult to see an intuitive relationship between problem (1.10) and the network revenue management problem that we are interested in solving. In this section, we give a practical interpretation for problem (1.10) that clarifies this relationship. This practical interpretation also becomes useful to construct an efficient solution method for problem (1.10) later in this section.

Associating the dual variables $\{w_{it} : i \in \mathcal{L},\ t \in \mathcal{T}\}$ and $\{y_{jt} : j \in \mathcal{J},\ t \in \mathcal{T}\}$ respectively with constraints (1.10b) and (1.10c) in problem (1.10), we write the dual of this problem as

$$\max \quad \sum_{t \in \mathcal{T}} \sum_{j \in \mathcal{J}} p_{jt} \, f_j \, y_{jt} \tag{1.11a}$$

$$\text{subject to} \quad w_{i1} = c_i \qquad \forall\, i \in \mathcal{L} \tag{1.11b}$$

$$w_{it} = w_{i,t-1} - \sum_{j \in \mathcal{J}} p_{j,t-1} \, a_{ij} \, y_{j,t-1} \qquad \forall\, i \in \mathcal{L},\ t \in \mathcal{T} \setminus \{1\} \tag{1.11c}$$

$$a_{ij} \, y_{jt} \leq w_{it} \qquad \forall\, i \in \mathcal{L},\ j \in \mathcal{J},\ t \in \mathcal{T} \tag{1.11d}$$

$$y_{jt} \leq 1 \qquad \forall\, j \in \mathcal{J},\ t \in \mathcal{T} \tag{1.11e}$$

$$w_{it} \text{ is free},\ y_{jt} \geq 0 \qquad \forall\, i \in \mathcal{L},\ j \in \mathcal{J},\ t \in \mathcal{T}. \tag{1.11f}$$

Constraints (1.11b) and (1.11c) are associated with the decision variables $\{v_{it} : i \in \mathcal{L},\ t \in \mathcal{T}\}$ in problem (1.10), whereas the constraints (1.11d) and (1.11e) are respectively associated with the decision variables $\{\beta_{ijt} : i \in \mathcal{L},\ j \in \mathcal{J},\ t \in \mathcal{T}\}$ and $\{\gamma_{jt} : j \in \mathcal{J},\ t \in \mathcal{T}\}$. We observe that the constraints associated with the decision variables $\{\alpha_{it} : i \in \mathcal{L},\ t \in \mathcal{T}\}$ in the dual of problem (1.10) are given by $w_{it} \leq c_i$ for all $i \in \mathcal{L}$, $t \in \mathcal{T}$, but constraints (1.11b) and (1.11c) already imply that $c_i = w_{i1} \geq w_{i2} \geq \ldots \geq w_{i\tau}$ for all $i \in \mathcal{L}$. Therefore, the constraints associated with the decision variables $\{\alpha_{it} : i \in \mathcal{L},\ t \in \mathcal{T}\}$ are redundant in problem (1.11) and they are omitted. We observe that the size of the problem above increases linearly with the numbers of flight legs and itineraries.

We view the decision variable w_{it} in problem (1.11) as the expected remaining capacity on flight leg i at the beginning of time period t and the decision variable y_{jt} as the probability with which we open itinerary j for sale at time period t. At time period t, we open each itinerary j for sale with probability y_{jt}, independent of the other itineraries. With probability $1 - y_{jt}$, itinerary j is closed at time period t. To make a sale for itinerary j at time period t, we need to have itinerary j open and have a request for this itinerary, which happens with probability $p_{jt}\, y_{jt}$. Thus, the objective function of problem (1.11) accounts for the total expected revenue from the served itinerary requests. Constraints (1.11b) initialize the expected remaining capacities. Noting that $\sum_{j \in \mathcal{J}} p_{jt}\, a_{ij}\, y_{jt}$ is the expected capacity consumption on flight leg i at time period t, constraints (1.11c) compute the expected remaining capacities at the next time period as a function of the expected remaining capacities and itinerary sales at the current time period. Constraints (1.11d) ensure that the expected capacity consumption on flight leg i at time period t conditional on the fact that there is a request for itinerary j does not exceed the expected remaining capacity on flight leg i.

The number of constraints in problem (1.11) is substantially smaller than the number of constraints in problem (1.3), but problem (1.11) still has $|\mathcal{L}||\mathcal{T}| + |\mathcal{L}||\mathcal{J}||\mathcal{T}|$ constraints, which may be too many to solve this problem directly by using a linear programming solver. Noting that constraints (1.11d) can be replaced by nonnegativity constraints on the decision variables $\{w_{it} : i \in \mathcal{L},\ t \in \mathcal{T}\}$ whenever itinerary j does not use flight leg i, if we let L be the maximum number of flight legs that an itinerary uses, then we can reduce the number of constraints to $|\mathcal{L}||\mathcal{T}| + L|\mathcal{J}||\mathcal{T}|$, but this may also be too many. The key observation that allows us to solve problem (1.11) efficiently is that the expected remaining capacities $\{w_{it} : i \in \mathcal{L},\ t \in \mathcal{T}\}$ are initialized to the total available capacities by constraints (1.11b). Since $a_{ij} \in \{0, 1\}$ and y_{jt} is bounded by one, constraints (1.11d) are not active whenever w_{it} exceeds one, which indicates that we do not expect constraints (1.11d) to be tight over a large portion of the selling horizon. This observation motivates solving problem (1.11) by using constraint generation, where we iteratively solve a master problem that has the same decision variables as problem (1.11) and includes all of constraints (1.11b), (1.11c) and (1.11e), but has only a subset of constraints (1.11d). After solving the master problem, we check whether any one of the constraints in (1.11d) are violated by the solution to the current master problem. If there is one, then we add this constraint to the master problem and resolve it.

When compared with solving problem (1.3) by using constraint generation, applying constraint generation on problem (1.11) provides two potential advantages. First, we can simply enumerate over all $i \in \mathcal{L}$, $j \in \mathcal{J}$ and $t \in \mathcal{T}$ to check whether any one of the constraints in (1.11d) are violated by the solution to the current master problem. We do not need to solve a separate problem to identify violated constraints. Second, we may be interested in solving problem (1.11) not only to obtain a control policy, but also to obtain the upper bound on the optimal total expected revenue provided by the optimal objective value of this problem. In practice, it may not be possible to solve problem (1.11) to optimality by using constraint generation. However, since problem (1.11) is a maximization problem and the master problem includes a subset of its constraints, the optimal objective value of the master problem is always an upper bound on the optimal objective value of problem (1.11). Therefore, if we stop constraint generation at an intermediate iteration, then the optimal objective value of the master problem naturally provides an upper bound on the optimal total expected revenue. This observation does not immediately apply when we solve problem (1.3) by using constraint generation since this problem is a minimization problem, but as Proposition 3 in [1] shows, we can use the slack variables of the constraints in problem (1.3) to obtain an upper bound on the optimal total expected revenue even when we stop constraint generation for problem (1.3) at an intermediate iteration.

Problem (1.11) has a surprising connection to the earlier literature. [22] propose problem (1.11) as a deterministic linear programming approximation to the network revenue management problem. The strongest result they can give for problem (1.11) is that the optimal objective value of this problem is greater than or equal to the optimal objective value of problem (1.3). Their result gives the impression that problem (1.3) is a stronger approximation for the network revenue management problem than problem (1.11) as it potentially yields a tighter upper bound, but our findings in this chapter show that this is not the case and problems (1.3) and (1.11) are equivalent to each other. Another interesting observation is that [22] obtain problem (1.11) by using Lagrangian relaxation to relax the capacity availability constraints in the dynamic programming formulation in (1.1). This observation naturally raises the question of whether using Lagrangian relaxation on the dynamic programming formulation of a general decision making problem, as is done in [22], is equivalent to using affine approximations to the value functions, as is done in [1]. It is not too difficult to see that this is not true in general. To give a simple example, if the decisions made at one time period has no effect on the future decisions, then using a value function approximation of zero at each time period would still give the optimal policy since there is no need to use value functions to assess the impact of the current decisions on the future. So, we can obtain the optimal policy trivially by using affine value function approximations with zero slopes and zero intercepts. In contrast, relaxing certain constraints through Lagrangian relaxation at each time period may certainly result in suboptimal decisions, especially when there are integrality constraints on the decisions. Thus, using affine value function approximations is not equivalent to using Lagrangian relaxation in the dynamic programming formulation of a general decision making problem, but in the network revenue management setting, our results in this chapter establish such an equivalence.

There may be other ways to show the equivalence between problems (1.3) and (1.11). Subsequent to our work, [40] gave an alternative proof for the equivalence between problems (1.3) and (1.11) by aggregating the decision variables in the dual of problem (1.3). In their work, [40] take problems (1.3) and (1.11) as given and show how to obtain the optimal solution to one problem by using the optimal solution to the other. In contrast, we follow a constructive approach, showing the equivalence between problems (1.3) and (1.11) as we construct problem (1.11).

1.7 Computational Experiments

In this section, we provide computational experiments that compare the solution times when we use constraint generation to solve problems (1.3) and (1.11).

1.7.1 Experimental Setup

In our test problems, we work with two different types of airline networks. In the first type of network, we have a single hub serving N spokes. There is a flight leg from the hub to each spoke and another one from each spoke to the hub. Thus, the number of flight legs is $2N$. It is possible to go from each origin to each destination in the airline network so that the number of possible origin-destination pairs is $N(N + 1)$. Going from one spoke to another requires taking two flight legs through the hub. Going from a spoke to the hub or from the hub to a spoke is possible through a direct flight leg. Figure 1.2.a shows the structure of the first type of network with $N = 8$. In the second type of network, we have two hubs serving a total of N spokes. The first half of the spokes are connected to the first hub and the second half of the spokes are connected to the second hub. There is a flight leg from each spoke to a hub and another flight leg from the corresponding hub to each spoke. There are also two flight legs that connect the two hubs in two directions. Thus, the number of flight legs is $2N+2$. It is possible to go from each origin to each destination in the airline network, in which case, the number of possible origin-destination pairs is $(N+1)(N+2)$. Going from one spoke to another requires taking two or three flight legs, whereas going from a spoke to a hub or from a hub to a spoke requires taking one or two flight legs. It is possible to go from one of the hubs to the other through a direct flight leg. Figure 1.2.b shows the structure of the second type of network with $N = 8$.

We have a high-fare and a low-fare itinerary associated with each origin-destination pair in the airline network. The fare associated with a high-fare itinerary is four times the fare associated with the corresponding low-fare itinerary. The arrival probabilities are calibrated so that the probability of getting a request for a high-fare itinerary increases over time, whereas the probability of getting a request for a low-fare itinerary decreases. Since the total expected demand for the capacity on flight leg i is given by $\sum_{t \in \mathcal{T}} \sum_{j \in \mathcal{J}} a_{ij}\, p_{jt}$, we measure the tightness of the leg capacities by

$$\alpha = \frac{\sum_{i \in \mathcal{L}} \sum_{t \in \mathcal{T}} \sum_{j \in \mathcal{J}} a_{ij}\, p_{jt}}{\sum_{i \in \mathcal{L}} c_i}.$$

In our test problems, we vary the number of time periods in the selling horizon over $\{600, 800, 1{,}000\}$, the number of spokes in the airline network over $\{8, 12\}$ and the tightness of the leg capacities over $\{1.0, 1.3, 1.6\}$. This experimental setup yields 18 test problems for each type of airline network.

1.7.2 Implementation of Constraint Generation

We experimented with a number of constraint generation strategies for solving problem (1.3). The following strategy consistently performed the best for our test problems. Letting $\{\tilde{\theta}_t : t \in \mathcal{T}\}$ and $\{\tilde{v}_{it} : i \in \mathcal{L},\ t \in \mathcal{T}\}$ be the solution to the current master problem, to check whether this solution violates any of constraints (1.3b) in problem (1.3), we solve problem (1.4) for each time period. If the optimal objective value of problem (1.4) exceeds $\tilde{\theta}_t - \tilde{\theta}_{t+1}$ for a time period t, then using $\tilde{x}_t$ and $\tilde{u}_t$ to denote an optimal solution to problem (1.4), constraint (1.3b) corresponding to $\tilde{x}_t \in \mathcal{X}$, $\tilde{u}_t \in \mathcal{U}(\tilde{x}_t)$ and $t \in \mathcal{T}$ is violated. We add this constraint to the master problem. Once we find five violated constraints, we stop looking for others. We solve the master problem with the added constraints and this concludes one iteration of constraint generation. As the iterations progress, the constraints added in the earlier iterations may not be relevant any more and become loose. If the number of loose constraints exceeds 70% of the number of constraints in the master problem, then we remove all loose constraints from the master problem. Finally, [1] shows that the decision variables $\{v_{it} : i \in \mathcal{L},\ t \in \mathcal{T}\}$ satisfy $v_{i1} \geq v_{i2} \geq \ldots \geq v_{i\tau}$ for all $i \in \mathcal{L}$ in an optimal solution to problem (1.3). We add all of these constraints to the master problem before we even start constraint generation. Adding these constraints at the beginning significantly speeds up the overall performance of constraint generation. The same observation is also made by [1].

We solve problem (1.11) by using constraint generation as well. For our test problems, the best performing constraint generation strategy we found is as follows. We add all of constraints (1.11b), (1.11c) and (1.11e) to the master problem at the beginning. Therefore, we only look for violations of constraints (1.11d).

After solving the master problem at the current iteration, we enumerate over all $i \in \mathcal{L}$, $j \in \mathcal{J}$ and $t \in \mathcal{T}$ to find which of constraints (1.11d) are violated by the solution to the current master problem. We add all of the violated constraints to the master problem and solve the master problem with the added constraints. This concludes one iteration of constraint generation. We note that although we solve problem (1.11) by using constraint generation, generating constraints for this problem is trivial.

1.7.3 Computational Results

Tables 1.1 and 1.2 summarize our main computational results, where the two tables respectively focus on the test problems with one and two hubs. In these tables, the first column shows the characteristics of the test problem by using the triplet (τ, N, α), where τ is the number of time periods in the selling horizon, N is the number of spokes and α characterizes the tightness of the leg capacities. Recalling that we have a high-fare and a low-fare itinerary for each origin-destination pair, if there are N spokes in the airline network, then the numbers of flight legs and itineraries are respectively $2N$ and $2N(N + 1)$ for the airline networks with one hub, whereas the numbers of flight legs and itineraries are respectively $2N + 2$ and $2(N + 1)(N + 2)$ for the airline networks with two hubs. The second to fifth columns in Tables 1.1 and 1.2 show the performance of constraint generation when applied to problem (1.3). In particular, the second column shows the CPU seconds required to solve problem (1.3) to optimality. The third column shows the CPU seconds required to solve problem (1.3) with 1% optimality gap. The fourth column shows what percentage of the CPU seconds is spent on generating constraints. The remaining percentage of the CPU seconds is spent on solving the master problem. The fifth column shows the total number of constraints generated to solve problem (1.3) to optimality. The interpretations of the sixth, seventh, eighth and ninth columns are similar to those of the previous four, but these columns focus on constraint generation when applied to problem (1.11). The last two columns in Tables 1.1 and 1.2 compare the solution times for problems (1.3) and (1.11). In particular, the tenth column shows the ratio between the CPU seconds required to solve problems (1.3) and (1.11) to optimality. The eleventh column shows the ratio between the CPU seconds required to solve problems (1.3) and (1.11) with 1% optimality gap. Our computational implementation is carried out by using Gurobi 4.5 as the linear programming solver. In our computational experiments, we focus on demonstrating the computational savings obtained by solving problem (1.11) by using constraint generation instead of solving problem (1.3). We do not test the performance of the policies obtained from these problems. [1] and [38] compare the performance of the policies with a variety of benchmark strategies and report quite favorable results.

The results in Tables 1.1 and 1.2 indicate that problem (1.11) provides significant computational savings over problem (1.3) in terms of CPU seconds. Over all of our test problems, the longest CPU seconds for problem (1.11) is 33 seconds, but problem (1.3) may take up to 1,931 seconds for some of the larger test problems with large number of time periods in the selling horizon and large number of spokes in the airline network. Similar observations apply if we are interested in obtaining a solution with 1% optimality gap. By using problem (1.11), we can obtain a solution with 1% optimality gap within 30 seconds in the worst case. For most of the test problems, we can obtain a solution with 1% optimality gap within three seconds. On the other hand, problem (1.3) may take up to 449 seconds to solve with 1% optimality gap. The ratio between the CPU seconds required to solve problems (1.3) and (1.11) can be as high as 135. The average of the ratios between the CPU seconds required to solve the two problems comes out to be 52. If we are interested in obtaining a solution with 1% optimality gap, then the average ratio between the CPU seconds is 71.

To get a feel for the problem characteristics that affect the CPU seconds for problems (1.3) and (1.11), we observe that the CPU seconds for both problems increase as the number of time periods in the selling horizon and the number of spokes increase, which is not surprising, since these problem characteristics directly affect the numbers of decision variables and constraints in problems (1.3) and (1.11). On the other hand, we observe that the CPU seconds for both problems also increase as the leg capacities get tighter. Although the numbers of decision variables and constraints do not depend on the tightness of the leg capacities, we need to generate more constraints to obtain the optimal solution and this translates into longer CPU seconds. It is also worthwhile to note that the largest values for the ratios between the CPU seconds required to solve problems (1.3) and (1.11) correspond to the test problems with tight leg capacities. Therefore, although it takes more time to solve either of the two problems as the leg capacities get tighter, problem (1.11) is affected less. It is encouraging that problem (1.11) provides the largest computational savings when problem (1.3) is particularly difficult to solve.

To sum up, our computational results indicate that using constraint generation to solve problem (1.11) is significantly more efficient than working with problem (1.3) directly. Problem (1.11) maintains its advantage across a variety of sizes of test problems and the gaps in the CPU seconds are most noticeable for the larger test problems with tighter leg capacities. Given these considerations, reducing problem (1.3) to problem (1.11) and solving the latter problem by using constraint generation is clearly a viable alternative to solving problem (1.3) by using constraint generation.

The CPU seconds reported for problem (1.11) in Tables 1.1 and 1.2 correspond to the case where we solve this problem by using constraint generation as described in Section 1.7.2. A natural question is how much benefit we obtain from using constraint generation to solve problem (1.11). To answer this question, Table 1.3 compares the CPU seconds for problem (1.11) when we solve this problem by using constraint generation and when we solve this problem directly by using a linear programming solver. The left and right sides of Table 1.3 respectively focus on the test problems with one and two hubs. The first column in this table shows the characteristics of the test problem. The second and third columns respectively show the CPU seconds required to solve problem (1.11) to optimality and with 1% optimality gap by using constraint generation. These two columns are identical to the sixth and seventh columns in Tables 1.1 and 1.2. The fourth and fifth columns in Table 1.3 respectively show the CPU seconds required to solve problem (1.11) to optimality and with 1% optimality gap directly by using a linear programming solver. In other words, the CPU seconds in the fourth and fifth columns correspond to the case where we a priori construct all of the constraints in problem (1.11) and directly maximize the objective function of problem (1.11) subject to all of the constraints listed in this problem. The results in Table 1.3 show that it may be faster to solve problem (1.11) directly by using a linear programming solver when the number of spokes is small or when the leg capacities are not too tight. However, if the number of spokes in the airline network is large and the leg capacities are tight, then it is significantly faster to solve problem (1.11) by using constraint generation. The benefits from constraint generation are especially noticeable when the CPU seconds are on the large side.

Finally, Table 1.4 shows the CPU seconds for problem (1.11) on larger test problems. These test problems involve 20 or 24 spokes and a single hub. Similar to our other test problems, we vary the number of time periods in the selling horizon over $\{600, 800, 1{,}000\}$ and the tightness of the leg capacities over $\{1.0, 1.3, 1.6\}$. The first column in Table 1.4 shows the characteristics of the test problem. The second and third columns respectively show the CPU seconds required to solve problem (1.11) to optimality and with 1% optimality gap. The CPU seconds given in Table 1.4 correspond to the case where we solve problem (1.11) by using constraint generation and our results show that we can use constraint generation to obtain the optimal solution to problem (1.11) within several minutes. It turns out that even for the largest test problems, we can obtain the optimal solution to problem (1.11) within eight minutes. For the test problems with 24 spokes, when we try to solve problem (1.11) directly by using a linear programming solver, we would either not be able to obtain the optimal solution within 10 minutes or run out of two gigabytes of memory. Furthermore, we tried to solve problem (1.3) by using constraint generation, but we could not get a solution with 1% optimality gap within a time limit of two hours. Therefore, exploiting the equivalence between problems (1.3) and (1.11) and solving problem (1.11) by using constraint generation provides a viable approach for dealing with large test problems.

1.8 Conclusions

In this chapter, we considered the approximate linear programming approach for network revenue management problems. This approach ends up with a linear program whose number of constraints increases exponentially with the number of flight legs. This linear program is commonly solved by using constraint generation. Each constraint can be generated by solving a separate integer program. The necessity to solve integer programs to generate constraints and the slow convergence behavior of constraint generation methods are practical drawbacks for using the approximate linear programming approach on network revenue management problems. Our goal in this chapter was to address these drawbacks. We showed that we can generate constraints for the linear program by solving minimum-cost network flow problems. Furthermore, by exploiting the minimum-cost network flow structure, we showed that we can a priori reduce the number of constraints in the linear program from exponential in the number of flight legs to linear. Computational experiments indicated that our results can provide substantial savings in terms of computation time.

The approximate linear programming approach finds applications in other settings, such as inventory distribution, vehicle routing and joint replenishment. In these settings, it is customary to formulate the problem as a dynamic program with a high-dimensional state variable and use affine approximations to the value functions. It would be of interest to explore whether the results that we showed in this chapter can be extended to enhance the computational performance of the approximate linear programming approach when applied in settings other than network revenue management.

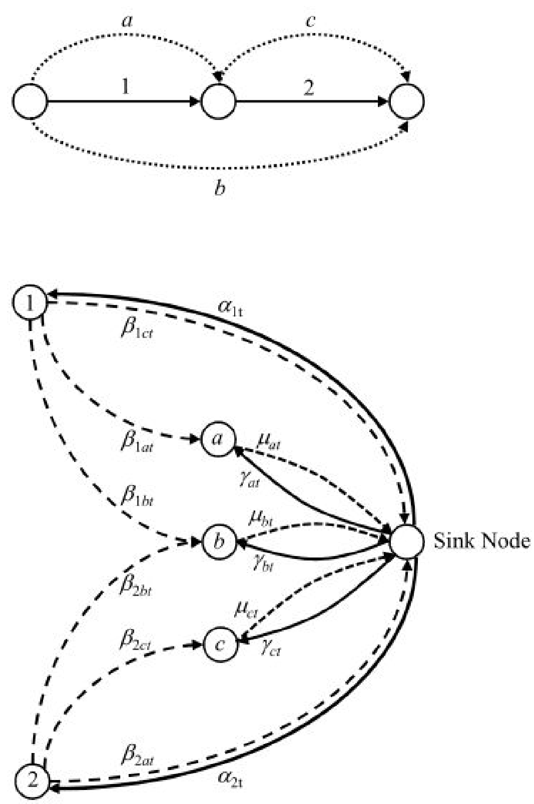

Figure 1.1: Minimum-cost network flow problem corresponding to problem (1.6).

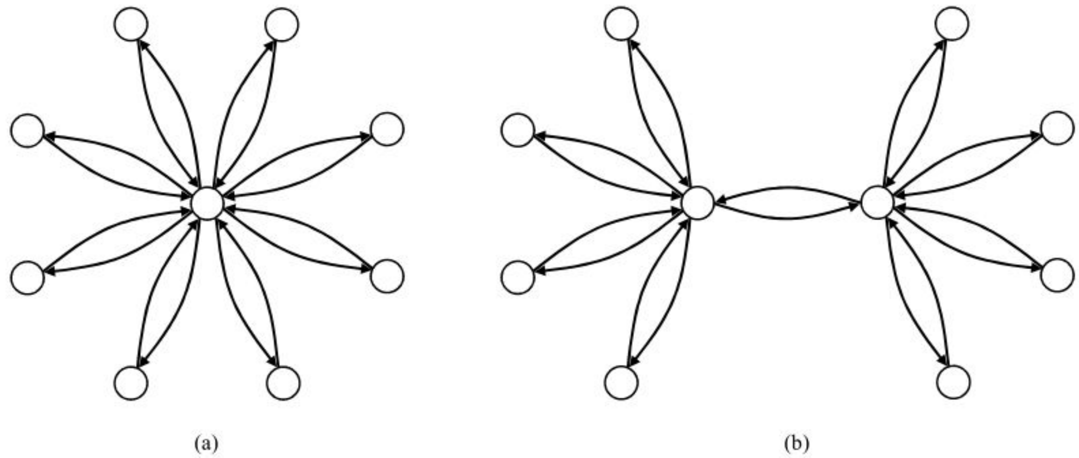

Figure 1.2: Structure of the airline network with one and two hubs for the case with $N = 8$.

Test problem (τ, N, α)	Cons. gen. for prob. (1.3) Total secs.	1% secs.	% cons. gen.	# cons.	Cons. gen. for prob. (1.11) Total secs.	1% secs.	% cons. gen.	# cons.	Ratio of secs. Total secs.	1% secs.
(600, 8, 1.0)	27.95	12.70	14.45	1,587	1.96	0.48	16.84	2,181	14.26	26.46
(600, 8, 1.3)	134.61	37.84	8.86	3,754	3.09	0.53	11.33	2,190	43.56	71.40
(600, 8, 1.6)	163.79	61.85	7.68	4,640	2.48	2.11	12.90	2,241	66.04	29.31
(600,12, 1.0)	181.00	23.34	14.41	3,616	9.09	1.22	8.03	6,978	19.91	19.13
(600,12, 1.3)	476.16	114.73	11.51	7,007	14.83	13.06	5.12	8,107	32.11	8.78
(600,12, 1.6)	582.61	159.20	11.37	7,894	15.64	14.02	4.86	7,480	37.25	11.36
(800, 8, 1.0)	33.06	22.83	10.74	1,793	2.49	0.75	17.27	2,066	13.28	30.44
(800, 8, 1.3)	232.19	72.45	6.88	4,719	2.96	0.83	15.88	2,176	78.44	87.29
(800, 8, 1.6)	298.00	117.37	5.47	6,137	3.21	0.81	13.40	2,241	92.83	144.90
(800,12, 1.0)	266.51	42.30	12.28	4,315	9.47	1.81	10.56	7,546	28.14	23.37
(800,12, 1.3)	773.73	238.70	8.96	7,403	12.69	1.97	7.96	7,309	60.97	121.17
(800,12, 1.6)	975.30	306.42	8.89	9,267	22.15	20.74	4.61	7,471	44.03	14.77
(1000, 8, 1.0)	58.15	38.53	9.36	2,231	2.72	1.00	20.59	2,336	21.38	38.53
(1000, 8, 1.3)	376.62	118.89	4.54	5,593	3.79	1.02	14.78	2,173	99.37	116.56
(1000, 8, 1.6)	451.43	201.08	4.39	7,166	3.34	1.01	16.47	2,268	135.16	199.09
(1000,12, 1.0)	378.42	63.48	9.46	4,459	11.39	2.39	10.80	7,750	33.22	26.56
(1000,12, 1.3)	1243.83	414.68	6.35	9,356	22.09	3.21	5.57	7,779	56.31	129.18
(1000,12, 1.6)	1556.90	408.71	6.23	11,265	20.84	2.60	5.95	7,476	74.71	157.20
Average									52.83	69.75

Table 1.1: Computational results for the test problems with one hub.

Test problem (τ, N, α)	Cons. gen. for prob. (1.3) Total secs.	1% secs.	% cons. gen.	# cons.	Cons. gen. for prob. (1.11) Total secs.	1% secs.	% cons. gen.	# cons.	Ratio of secs. Total secs.	1% secs.
(600, 8, 1.0)	71.60	14.50	13.63	2,530	3.43	0.71	11.95	3,963	20.87	20.42
(600, 8, 1.3)	188.40	46.57	7.99	5,162	3.49	0.69	12.03	3,253	53.98	67.49
(600, 8, 1.6)	249.05	64.55	7.69	6,447	5.45	0.70	7.71	3,212	45.70	92.21
(600,12, 1.0)	205.05	27.80	13.76	3,820	5.57	1.56	14.90	10,794	36.81	17.82
(600,12, 1.3)	645.48	184.92	10.31	7,636	16.54	1.56	5.02	10,108	39.03	118.54
(600,12, 1.6)	893.31	208.37	13.87	9,747	30.57	21.17	2.72	9,654	29.22	9.84
(800, 8, 1.0)	115.27	25.83	12.01	3,049	3.89	1.03	13.88	3,905	29.63	25.08
(800, 8, 1.3)	280.69	81.41	6.91	5,420	5.37	0.98	10.06	3,234	52.27	83.07
(800, 8, 1.6)	470.34	121.14	5.19	8,502	5.32	1.00	10.34	3,234	88.41	121.14
(800,12, 1.0)	269.02	49.32	9.07	3,464	8.41	2.16	13.08	10,896	31.99	22.83
(800,12, 1.3)	1108.39	239.38	8.43	9,322	29.66	2.18	3.71	9,961	37.37	109.81
(800,12, 1.6)	1587.84	396.69	9.89	11,686	33.38	30.93	4.37	9,639	47.57	12.83
(1000, 8, 1.0)	119.30	43.80	14.74	2,583	3.55	1.32	19.44	3,811	33.61	33.18
(1000, 8, 1.3)	422.91	130.35	5.51	6,146	5.79	1.30	12.61	3,183	73.04	100.27
(1000, 8, 1.6)	758.49	196.56	3.92	10,524	7.58	1.29	8.84	3,233	100.06	152.37
(1000,12, 1.0)	405.09	76.06	6.81	4,148	7.80	2.82	17.82	10,946	51.93	26.97
(1000,12, 1.3)	1909.78	411.63	5.73	12,112	22.58	2.83	6.16	9,235	84.58	145.45
(1000,12, 1.6)	1931.23	449.95	9.80	15,925	30.44	2.87	4.53	9,697	63.44	156.78
Average									51.08	73.12

Table 1.2: Computational results for the test problems with two hubs.

Test problem (τ, N, α)	Cons. gen. for prob. (1.11) Total secs.	Cons. gen. for prob. (1.11) 1% secs.	Dir. solut. for prob. (1.11) Total secs.	Dir. solut. for prob. (1.11) 1% secs.
(600, 8, 1.0)	1.96	0.48	1.01	0.81
(600, 8, 1.3)	3.09	0.53	2.94	1.82
(600, 8, 1.6)	2.48	2.11	3.56	2.09
(600,12, 1.0)	9.09	1.22	8.52	3.00
(600,12, 1.3)	14.83	13.06	22.39	7.62
(600,12, 1.6)	15.64	14.02	36.01	9.18
(800, 8, 1.0)	2.49	0.75	1.79	1.46
(800, 8, 1.3)	2.96	0.83	2.82	1.95
(800, 8, 1.6)	3.21	0.81	5.99	3.61
(800,12, 1.0)	9.47	1.81	8.67	3.33
(800,12, 1.3)	12.69	1.97	18.21	5.72
(800,12, 1.6)	22.15	20.74	58.15	15.43
(1000, 8, 1.0)	2.72	1.00	4.35	3.55
(1000, 8, 1.3)	3.79	1.02	5.32	3.21
(1000, 8, 1.6)	3.34	1.01	8.20	3.87
(1000,12, 1.0)	11.39	2.39	9.17	4.82
(1000,12, 1.3)	22.09	3.21	45.38	10.62
(1000,12, 1.6)	20.84	2.60	82.29	11.27

Test problem (τ, N, α)	Cons. gen. for prob. (1.11) Total secs.	Cons. gen. for prob. (1.11) 1% secs.	Dir. solut. for prob. (1.11) Total secs.	Dir. solut. for prob. (1.11) 1% secs.
(600, 8, 1.0)	3.43	0.71	1.79	0.78
(600, 8, 1.3)	3.49	0.69	3.48	1.62
(600, 8, 1.6)	5.45	0.70	6.06	2.42
(600,12, 1.0)	5.57	1.56	5.51	1.90
(600,12, 1.3)	16.54	1.56	27.16	9.43
(600,12, 1.6)	30.57	21.17	47.57	12.61
(800, 8, 1.0)	3.89	1.03	2.16	1.87
(800, 8, 1.3)	5.37	0.98	5.71	1.81
(800, 8, 1.6)	5.32	1.00	8.11	4.17
(800,12, 1.0)	8.41	2.16	6.18	3.42
(800,12, 1.3)	29.66	2.18	56.74	6.60
(800,12, 1.6)	33.38	30.93	83.88	17.36
(1000, 8, 1.0)	3.55	1.32	4.96	1.47
(1000, 8, 1.3)	5.79	1.30	4.51	2.35
(1000, 8, 1.6)	7.58	1.29	10.52	4.36
(1000,12, 1.0)	7.8	2.82	8.71	3.43
(1000,12, 1.3)	22.58	2.83	53.38	7.34
(1000,12, 1.6)	30.44	2.87	110.58	14.19

Table 1.3: CPU seconds for problem (1.11) when we solve this problem by using constraint generation and when we solve this problem directly by using a linear programming solver.

Test problem (τ, N, α)	Cons. gen. for prob. (1.11) Total secs.	1% secs.
(600,20, 1.0)	34.77	3.69
(600,20, 1.3)	42.07	9.19
(600,20, 1.6)	54.17	12.93
(600,24, 1.0)	151.56	5.02
(600,24, 1.3)	124.70	20.69
(600,24, 1.6)	120.11	50.57

Test problem (τ, N, α)	Cons. gen. for prob. (1.11) Total secs.	1% secs.
(800,20, 1.0)	37.69	4.71
(800,20, 1.3)	62.11	5.06
(800,20, 1.6)	74.30	11.21
(800,24, 1.0)	80.79	6.84
(800,24, 1.3)	96.69	18.66
(800,24, 1.6)	454.20	26.17

Test problem (τ, N, α)	Cons. gen. for prob. (1.11) Total secs.	1% secs.
(1000,20, 1.0)	46.67	5.96
(1000,20, 1.3)	78.63	6.00
(1000,20, 1.6)	66.77	12.13
(1000,24, 1.0)	85.91	8.69
(1000,24, 1.3)	164.79	8.93
(1000,24, 1.6)	143.18	26.87

Table 1.4: CPU seconds for problem (1.11) for large test problems.

CHAPTER 2

FACILITY LOCATION WITH TYPES

2.1 Introduction

Although algorithm design for the *facility location problem* and its many variants has been the focus of a significant body of research, there is an important class of generalizations that has received much attention, but only with more limited success. In the facility location problem, one has a set of demand points and a set of potential facility locations, with specified costs for opening, and the aim is to optimize the choice of open facilities so as to minimize the sum of opening costs and the cost of assigning each demand point to an open facility, where the assignment costs derive from an underlying metric space containing all of the locations. This model places no restrictions on the clients served by each facility, which is typically an unreasonable assumption, and so a great deal of work has been done to address further constrained models. The simplest extension is to impose a capacity constraint, but for a wide range of applications, it is more appropriate to distinguish types of service provided and required, and it is on this class of problems that we shall focus. Facility location models are closely linked to a number of inventory models, since one can view a facility as a point in time at which an order is placed, and the assignment cost then corresponds to the cost of filling a particular demand from that order, where one can have fixed ordering costs, unit ordering costs, and inventory holding costs easily incorporated into this setting. In this chapter, we shall consider both facility location and inventory management problems that arise in the setting with type constraints.

Since the facility location problem is NP-hard, much of the algorithmic work in this domain has focused on the design of approximation algorithms, and it has proved to be a fertile ground for the development of many of the now-standard techniques in algorithm design: researchers have applied deterministic and randomized rounding [33, 26, 11], primal-dual methods [18], local search[21], and even greedy-type algorithms [17] to this problem. For the classical facility location problem, the best approximation guarantee currently known is a 1.488-approximation algorithm (where a *ρ-approximation algorithm* is a polynomial-time algorithm that is guaranteed to find a feasible solution of objective function value within a factor of ρ of the optimum), which was obtained by randomized rounding [26]. For the *k-median problem*, in which there are no facility costs but one is limited to opening only k facilities, strong results are known via a range of techniques as well [27, 18, 3], and analogous results (though more limited) are known for the capacitated version of the facility location problem [12, 2, 29, 4]. One particularly notable open problem is to derive good algorithms for the setting in which the opening/ordering costs are submodular set functions of the set of demand points assigned (or for suitably general special cases).

The research presented here aims to derive results in which each facility may have a different serving capability. In the most general case along these lines, for each client, we can specify the set of facilities that can serve it. However, it is easy to reduce the set cover problem to it in an approximation-preserving manner (see appendix); thus, approximation hardness results known for the set cover problem would apply (and hence no sub-logarithmic guarantee is possible unless P=NP). Hence, the issue is to model sufficiently robust classes of problems that capture interesting application settings, and yet by-pass this hardness result. We consider the following classes of problems, in which each facility and client is assigned a type, and there is a specified partial ordering among types so that a client must be served by any open facility of equal or higher type. The structure of the partial ordering will determine the nature of our problems. With a general partial order, one can again reduce set cover to the problem; we consider two structured variants to bypass this.

- **Facility Location with Hierarchical Types.** In this problem, the types form a rooted tree in which a facility of a given type can serve any client that has a type that is a descendant of the facility type in the tree (where a node is trivially a descendant of itself).
- **Pepsi-Coke Facility Location.** In this problem, there are only two types of facilities: Pepsi facilities and Coke facilities. There are three types of clients: Pepsi clients, Coke clients and Anything-will-do clients. As the names suggest, Anything-will-do clients can be served by any facility, but the other clients can only be served by facilities of the same type. The partial ordering here is a "V-shape," where there are no facilities of the bottom type.

A model somewhat weaker than the former setting was considered by Barman and Chawla [5], who introduced a problem called the *redundancy-aware facility location problem*, where each client needs a set of services and all of these services need to be installed at the same facility location in order to serve that client; they gave an LP rounding algorithm with approximation guarantee of 27 for the case that all clients' demanded service sets are laminar. The facility location problem with hierarchical types is the generalization of their problem if we group services belong to the same service set to a single facility. We include more details on the relationship in appendix. In contrast, the first main result of this chapter is a simple primal-dual 3-approximation algorithm for the facility location problem with hierarchical types. We show how to add a simple prioritization rule to the pruning phase of the algorithm of Jain and Vazirani [18] leads to this result. Furthermore, the analysis gives a stronger result; we show that it is a Lagrangian-multiplier-preserving algorithm, in that not only is the sum of the assignment and facility costs at most three times the optimum, but the sum of the assignment and 3 times the facility cost is at most 3 times the optimal cost. We are unable to convert this to an analogous k-median result, but we can augment this analysis with a rescaling technique to obtain a 1.85-approximation algorithm.

For the Pepsi-Coke facility location problem, we add a new dimension to the by-now standard LP rounding-by-clustering techniques that arises due to the fact that when choosing a cluster for one of the Anything-will-do clients j, a facility used by the LP solution for that client, cannot necessarily serve all of the clients served by any facility fractionally serving j. We overcome this difficulty by separately performing two clustering constructions, but then couple the outcomes by relying on an auxiliary matroid intersection computation, where a feasible fractional solution to the matroid intersection instance is derived from the original fractional solution (in a manner similar to a technique used by Swamy [35]). Consequently, we obtain a 3-approximation algorithm for the Pepsi-Coke facility location problem.

Analogous to the development of approximation algorithms for facility location problems, there has been a corresponding thread of research investigating results of a variety of inventory management problems, primarily the *lot-sizing* and the *joint replenishment problem*[23, 24, 25, 7, 10]. In some elements, these problems are sometimes simpler than their facility location equivalents, since the assignment costs, in addition to obeying the triangle inequality, often have an even more refined structure due to the linear nature of ordering/demand time periods. However, these problems have a harder element, since the metric is definitely not symmetric - one cannot serve a demand point earlier in time than the ordering period. Hence, it is natural to ask for the analogs of the two results discussed above. Surprisingly, for the lot-sizing problem with hierarchical types, we show that the problem can be solved dynamic programming in polynomial time, in a manner exactly analogous to a recent result [10]. However, unlike for the unconstrained lot-sizing problem, for which the natural linear programming relaxation is the basis for a primal-dual polynomial-time algorithm as well [25], we give an example for which there is an integrality gap of 4/3 for the lot-sizing problem with hierarchical types.

2.2 Preliminaries

In this section, we introduce some definitions and notation to be used throughout this chapter. Let $\mathcal{F}$ be the set of potential facilities; opening facility $i \in \mathcal{F}$ has associated non-negative cost f_i. Let $\mathcal{D}$ be the set of demand points. We need to assign each client $j \in \mathcal{D}$ to some open facility i and it costs c_{ij}. In addition, there is set of types $\mathcal{T}$, forming a partial ordering. Each facility i has a type $t(i)$ and each client j has type $t(j)$. Facility i is capable of serving client j if and only if $t(j) \leq t(i)$ in the partial ordering. Types indicate the serving capabilities. Denote $\mathcal{F}(j)$ be all facilities that can serve client j.

Suppose we open the set of facilities S in the solution and assign client j to $\sigma(j)$. Then the total cost of this solution is $\sum_{i \in S} f_i + \sum_{j \in \mathcal{D}} c_{\sigma(j)j}$, i.e., the sum of total facility opening cost and total assignment cost. The goal is to find a feasible solution with minimum total cost.

Our algorithm and analysis will rely on the following primal linear-programming (LP) relaxation (P) and its dual (D).

$$
\begin{aligned}
\min \quad & \sum_{i \in \mathcal{F}} f_i y_i + \sum_{j \in \mathcal{D}} \sum_{i \in \mathcal{F}(j)} c_{ij} x_{ij} && \text{(P)} \\
\text{s.t.} \quad & \sum_{i \in \mathcal{F}(j)} x_{ij} \geq 1 \quad \forall j \in \mathcal{D} \\
& x_{ij} \leq y_i \quad \forall j \in \mathcal{D}, i \in \mathcal{F}(j) \\
& x_{ij}, y_i \geq 0 \quad \forall i \in \mathcal{F}, j \in \mathcal{D}
\end{aligned}
$$

$$
\begin{aligned}
\max \quad & \sum_{j \in \mathcal{D}} v_j && \text{(D)} \\
\text{s.t.} \quad & v_j \leq c_{ij} + w_{ij} \quad \forall j \in \mathcal{D}, i \in \mathcal{F}(j) \\
& \sum_{j : i \in \mathcal{F}(j)} w_{ij} \leq f_i \quad \forall i \in \mathcal{F} \\
& v_j, w_{ij} \geq 0 \quad \forall j \in \mathcal{D}, i \in \mathcal{F}(j).
\end{aligned}
$$

Any binary feasible solution to (P) corresponds to a feasible solution. $y_i = 1$ indicates that facility i is open and $x_{ij} = 1$ indicates that client j is served by facility i.

For the dual (D), we can intuitively think v_j as budget of client j and w_{ij} as its contribution towards opening facility i. The first set of constraints say that the budget can cover both assignment cost and contribution toward opening cost. The second set of constraints say that the total contribution toward one facility cannot exceed its opening cost.

2.3 Facility location with hierarchical types

We first consider the facility problem with hierarchical types, in which the types form a rooted tree with the root; type $t \leq t'$ if t' is on the path from t to the root. This models a hierarchical structure where facility higher up can serve more kinds of clients. The assignment cost c_{ij} correspond to a symmetric metric on $\mathcal{F} \cup \mathcal{D}$.

We first present a simple 3-approximation algorithm extending the classical primal-dual algorithm. We next show how to combine this with randomized rounding and cost scaling to achieve a 1.85-approximation.

2.3.1 A simple 3-approximation algorithm

The algorithm follows the standard Jain-Vazirani [18] approach and operates in two phases.

Dual ascent phase In this phase, we simultaneously construct a feasible solution to the dual linear program and a tentative solution to the primal, which will serve as the basis for our final primal solution.

Initially, we set each dual variable $v_j = 0$ and each client is active. We increase v_j for each active client at a uniform rate until one of the following events happens:

- **Event 1** When $v_j = c_{ij}$, for some $i \in \mathcal{F}(j)$ and i is not tentatively open, we cannot increase v_j any more without increasing w_{ij}. Thus, we start to increase w_{ij} along with v_j at the same rate.

- **Event 2** When $\sum_{j:i\in\mathcal{F}(j)} w_{ij} = f_i$, we can no longer increase the corresponding v_j. So we tentatively open facility i and mark an active client j inactive if $v_j \geq c_{ij}$ and $i \in \mathcal{F}(j)$.
- **Event 3** When $v_j = c_{ij}$, for some $i \in \mathcal{F}(j)$ and i is tentatively open, we mark j to be inactive.

The process ends when all clients are inactive.

Definition 1. *We say that client j contributes to facility i if $w_{ij} > 0$. We say that client j is frozen by facility i if we mark j inactive during Event 1 or Event 3 corresponding to facility i.*

For any tentatively open facility i, if we take all budgets of its contributing clients, it is enough to cover the opening cost f_i plus their assignment cost to i. However, the reason we cannot simply open all tentatively open facilities is we may spend the budget of some client multiple times if it contributes to several tentatively open facilities, making it hard to bound our cost against the dual solution. This motivates us to perform the Pruning Phase open facilities more selectively.

Pruning phase In this phase, we maintain a set D which we denote as *center clients*. We consider each type t in our tree $\mathcal{T}$ one by one with top-down ordering, each corresponding to one pass. During the pass for type t, we look at each tentatively open facility i of type t. We permanently open facility i and pay its opening cost f_i if all contributing clients j are not marked centers yet ($j \notin D$). Then we mark all contributing clients of facility i to be centers, adding them to D. On the other hand, if there is one contributing client $j \in D$, then we do not open facility i and move to the next facility.

Finally, we assign each client to the closest open facility i with $i \in \mathcal{F}(j)$.

2.3.2 The analysis

For our solution, let F be the total facility opening cost and let C be the total assignment cost. Let OPT be the optimal solution's cost. Now we want to prove the primal-dual algorithm is a Lagrangian-multiplier-preserving 3-approximation, which means $3F + C \leq 3OPT$.

We show a particular way of assigning clients to open facilities is of low cost, which also bounds the cost of assigning clients optimally to the open facilities. Recall that D is the set of *center clients*. Let S be the set of facilities that we permanently open in the end. For each permanently open facility $i \in S$, let $N(i)$ be the set of its contributing clients. We know $D = \cup_{i \in S} N(i)$. Our pruning phase immediately yields the following.

Fact 2.3.1. *For two permanently open facilities i, ℓ, $N(i)$, $N(\ell)$ are disjoint.*

Thus, for any $i \in S$, we can assign $N(i)$ to i. Note for any $j \in N(i)$, we know $w_{ij} > 0$ and this implies $i \in \mathcal{F}(j)$ so i is capable of serving j. we can pay for the total cost with the budgets of $N(i)$ because $f_i = \sum_{j \in N(i)} w_{ij} = \sum_{j \in N(i)} (v_j - c_{ij})$. Rearranging terms, we get $f_i + \sum_{j \in N(i)} c_{ij} = \sum_{j \in N(i)} v_j$.

Since $N(i), N(\ell)$ are disjoint for different facilities $i, \ell \in S$, we know we never double-spend any client's budget. We next bound the assignment cost of the remaining clients $\bar{D} = \mathcal{D} \backslash D$.

Lemma 2.3.2. *For any $j \in \bar{D}$, there is an open $i \in \mathcal{F}(j)$ and $c_{ij} \leq 3v_j$.*

Proof. Consider the facility i which froze client j. If $i \in S$, then we can just assign j to i and $c_{ij} \leq v_j$ and we are done.

If i is not opened, that means there is a client k contributing to both i and some open facility ℓ. Since k contributes to i, we know $t(k) \leq t(i)$. For same reason, we know $t(k) \leq t(\ell)$. This means in our type tree $\mathcal{T}$, either $t(i) \leq t(\ell)$ or $t(i) \geq t(\ell)$. We already know $t(i) \geq t(j)$ since j is frozen by i, so if $t(\ell) \geq t(i)$, then ℓ is capable of serving j. Suppose on the contrary, $t(\ell) < t(i)$, then it implies in our Pruning Phase, we should consider opening facility i before facility ℓ, since we consider all types in top-down fashion. This contradicts the fact that i is not opened because of facility ℓ. Thus, we know ℓ is capable of serving j.

Finally, we need to bound the cost of assigning j to l. We know

$$c_{j\ell} \leq c_{ij} + c_{ik} + c_{\ell k} \leq v_j + v_k + v_k = v_j + 2v_k.$$

Consider the relationship between v_j and v_k. We know k contributes to i, and so k must become inactive at the same or earlier than when we tentatively open i during Dual Ascent phase. Otherwise, when we tentatively open i, we will freeze j at the same time. On the other hand, since i froze j, we know j becomes inactive at the same time or later than we tentatively open i. So $v_k \leq v_j$ and thus $c_{j\ell} \leq 3v_j$.

□

Theorem 2.3.3. *The primal-dual algorithm is a Lagrangian-multiplier-preserving 3-approximation algorithm.*

Proof. Let F be the total facility opening cost and C be the total assignment cost. Also, for a set of clients P, let $C(P)$ denote the assignment cost for P. Since the budget of D can cover all facility opening costs and the assignment cost for D, we know $F + C(D) = v(D)$. We also know that for any client $j \in \bar{D}$, $C(j) \leq 3v_j$, and thus

$$3F + C \leq 3F + 3C(D) + C(\bar{D}) \leq 3v(D) + 3v(\bar{D}) = 3v((D)) \leq 3OPT.$$

The final inequality comes from the fact v is a feasible solution to the dual LP, and thus yields a lower bound on the optimal LP value, which is in turn a lower bound on the optimal solution's total cost. □

2.3.3 An improved algorithm

In this section, we show how to improve the approximation constant to 1.85. In our analysis for the primal-dual algorithm, we paid an expensive assignment cost on $\bar{D}$ for sending them to facilities three hops away. We use the idea of randomized rounding [11] to open some more facilities so that clients in $\bar{D}$ may be served nearby. To further balance the final facility-opening and assignment costs, we also utilize cost-scaling ideas [9].

Algorithm There are two constants $\alpha > 1, \beta$ that we will specify later.

1. Run the primal-dual algorithm on a modified instance where we rescale each facility opening cost from f_i to f_i/α. Let v^α be the dual budget produced,and let D^α be the set of center clients.
2. Solve the linear program for the original instance and get an optimal fractional solution x^*, y^*. For each facility $i \in \mathcal{F}$, independently open it with probability $\min\{1, \beta y_i^*\}$ if it is not opened by the primal-dual algorithm.
3. Assign each client to its closest open facility i in $\mathcal{F}(j)$.

Analysis Since the primal-dual algorithm will always produce a feasible solution, we only need to bound its total cost.

Let $C(j)$ be the random assignment cost of client j, also let $C^*(j) = \sum_{i \in \mathcal{F}(j)} x^*_{ij} c_{ij}$ be its assignment cost in the optimal fractional solution.

First, we analyze the assignment cost for non-center clients $\bar{D}^\alpha$.

Lemma 2.3.4. *[11] Take any client $j \in \mathcal{D}$; let A_j denote the event that there is at least one facility i with $x^*_{ij} > 0$ opened during randomized rounding. Then we know $P(A_j) \geq 1 - e^{-\beta}$ and $E[C(j)|A_j] \leq C^*(j)$.*

We bound the expected assignment cost for non-center clients $j \in \bar{D}^\alpha$.

Lemma 2.3.5. *For any non-center client $j \in \bar{D}^\alpha$, we have $E[C(j)] \leq (1 - e^{-\beta})C^*(j) + 3e^{-\beta} v^\alpha_j$.*

Proof. When $C^*_j \geq 3v^\alpha_j$, we can always assign j to a facility within distance $3v^\alpha_j$; thus $E[C(j)] \leq 3v^\alpha_j \leq (1 - e^{-\beta})C^*(j) + 3e^{-\beta} v^\alpha_j$.

In the other case when $C^*_j \leq 3v^\alpha_j$, we assign j to its closest facility in $\mathcal{F}(j)$ opened during randomized rounding if A_j happens, and we know $E[C(j)|A_j] \leq C^*(j)$. However, if A_j did not happen, we can still use the facility in the primal-dual algorithm and pay at most $3v^\alpha_j$. Thus,

$$
\begin{aligned}
E[C(j)] &= E[C(j)|A_j]P(A_j) + E[C(j)|\bar{A}_j]P(\bar{A}_j) \\
&\leq C^*(j)P(A_j) + 3v^\alpha_j(1 - P(A_j)) \\
&= 3v^\alpha_j - (3v^\alpha_j - C^*(j))P(A_j) \\
&\leq 3v^\alpha_j - (3v^\alpha_j - C^*(j))(1 - e^{-\beta}) = (1 - e^{-\beta})C^*(j) + 3e^{-\beta} v^\alpha_j
\end{aligned}
$$

□

Now, we can calculate the total cost incurred.

- During the primal-dual algorithm, the (scaled) facility opening cost and assignment cost of center clients D^α can be covered by $v^\alpha(D^\alpha)$. Thus, $\alpha v^\alpha(D^\alpha)$ is enough to cover this part of cost with the original facility opening cost.
- During randomized rounding, we incur expected facility opening cost at most $\beta F^* = \beta \sum_{i\in\mathcal{F}} f_i y_i$.
- For a non-center client $j \in \bar{D}^\alpha$, its expected assignment cost is at most

$$\sum_{j\in\bar{D}^\alpha}(1-e^{-\beta})C^*(j) + 3e^{-\beta}v_j^\alpha = (1-e^{-\beta})C^*(\bar{D}^\alpha) + 3e^{-\beta}v^\alpha(\bar{D}^\alpha)$$

Before we sum all of the costs, note that although we obtain x^*, y^* from the original instance, they are still a feasible LP solution for the instance with scaled facility opening cost. The nice thing is that they provide an upper bound on the LP value of the scaled instance of $F^*/\alpha + C^*$, where $C^* = \sum_{j\in\mathcal{D}, i\in\mathcal{F}(j)} c_{ij}x^*_{ij}$. Now summing all costs, we get that the total cost is at most $\alpha v^\alpha(D^\alpha) + \beta F^* + (1 - e^{-\beta})C^*(\bar{D}^\alpha) + 3e^{-\beta}v^\alpha(\bar{D}^\alpha)$. Taking $\alpha = 3e^{-\beta}$, this is bounded by

$$\alpha(F^*/\alpha + C^*) + \beta F^* + (1-e^{-\beta})C^* = (1+\beta)F^* + (1+2e^{-\beta})C^*.$$

By setting $\beta = 0.85, \alpha = 1.28$, we get 1.85 as the approximation ratio.

2.4 The Pepsi-Coke facility location problem

In the Pepsi-Coke problem, there are only two types of facilities: Pepsi facilities and Coke facilities. There are three types of clients: Pepsi clients $\mathcal{D}_P$, Coke clients $\mathcal{D}_C$ and Anything-will-do clients $\mathcal{D}_A$. As the names suggest, Anything-will-do clients can be served by any facility but the other clients can be served only by facilities of the same type. The type tree is a V shape where no facilities are of the bottom type. The assignment cost c_{ij} is also a symmetric metric.

We present a randomized rounding 3-approximation algorithm here. The main difficulty is that we may open a Pepsi facility for a cluster centered at an Anything-will-do client, and then we cannot serve any Coke facilities in the same cluster. We show how to overcome this difficulty by doing two clusterings and then coupling them using matroid intersection.

2.4.1 The algorithm

The algorithm is based on clustering and randomized rounding. The basic idea is to construct several clusters, where each cluster has one center client and several facilities. Each non-center client is assigned to some cluster. Then we can open one facility in each cluster and assign all clients in that cluster to it. You can find a detailed explanation in [41] Section 5.8.

However, in our case, we need to deal with clients of different types. Naive clustering will leave us with some non-center client not being able to be served via cluster center. We need to do something more delicate.

Now, we describe the algorithms. First, solve the linear programming and get optimal solutions x^*, y^*, w^*, v^*. Assume here that x^*_{ij} is either y^*_i or 0; we will need this later. This can be achieved by making several copies of facilities as in [11].

Definition 2. *We say client j neighbors facility i if $x^*_{ij} > 0$. Denote the neighborhood of client j to be $N(j) = \{i \in \mathcal{F}(j) : x^*_{ij} > 0\}$.*

Let $C^*(j) = \sum_{i\in\mathcal{F}(j)} x^*_{ij}c_{ij}$ be the fractional assignment cost of client j.

Clustering The purpose of clustering is to identify a set of center clients and ensure each non-center client is close to some center client. We will do two clusterings here: one for all Pepsi-only and Coke-only clients and another for Any-will-do clients.

Let D_1 be empty in the beginning. We consider client $j \in \mathcal{D}_P \cup \mathcal{D}_C$ one by one in ascending order by $v^*_j + C^*(j)$. If $N(j)$ does not intersect $N(k)$ for any $k \in D_1$, we add j to D_1. After we considered all clients in $\mathcal{D}_P \cup \mathcal{D}_C$, we will have our center-clients for the first clustering round.

Let D_2 be empty. Then we consider client $j \in \mathcal{D}_A$ one by one in ascending order by $v^*_j + C^*(j)$. If $N(j)$ does not intersect with $N(k)$ for any $k \in D_2$, we add j to D_2. In the end, we will have our center-clients for the second clustering round.

Fact 2.4.1. *Let $D = D_1 \cup D_2$ be the set of center clients from both clustering rounds. For any $j \in D$, $y^*(N(j)) = 1$.*

Rounding We would like to open exactly one facility for each cluster. However, since we have two sets of clusters and there could be some client $j \in D_1$ and $k \in D_2$ sharing the same facility in their neighborhoods, we cannot just randomly open one facility per cluster. If we use z to denote our decision of which facility to open, meaning that $z_i = 1$ if we want to open facility i and $z_i = 0$ otherwise. Then our requirement of opening exactly one facility in each cluster can be expressed as

$$\sum_{i \in N(j)} z_i = 1 \quad \forall j \in D_1, \qquad \sum_{i \in N(j)} z_i = 1 \quad \forall j \in D_2, \qquad z_i \geq 0 \quad \forall i \in \mathcal{F}.$$

Note that, since $N(j)$'s are disjoint within D_1 or D_2, we essentially have a matroid intersection base polytope for two partition matroids. We actually know one fractional point within this polytope: y^*.

From [15], we know that we can decompose, in polynomial time, z^* into a convex combination of polynomial number of vertice of the matroid intersection polytope:s $z^* = \sum_v \lambda_v z_v$. Here $\{\lambda_v\}$ is a distribution over vertices $\{z_v\}$. In addition, we know the matroid intersection polytope is integral [32], and so each z_v is binary. Thus, each z_v corresponds to a set of facilities to open.

We can sample vertex z_v according to distribution $\{\lambda_v\}$ and then open the set of facilities corresponding to z_v. In this way, we make sure that for each center client $j \in D$, we opened exactly one facility in $N(j)$. This rounding method is inspired by [35].

Assignment We describe here one assignment that we will show to be of low cost later. For each center-client j, we know exactly one facility $i \in N(j)$ is opened and we will assign j to i. For any non-center client j, it was classified as non-center during one of the clustering phases because, at the time when we consider j, some center client k has neighborhood $N(k)$ intersecting j's neighborhood $N(j)$. We will assign j to the same facility that k was assigned to.

Here we will show that this assignment is feasible. For anything-will-do clients, they are fine as long as they are served by some facility, so we do not need to worry about them. For a Pepsi-only client j, consider the center client k

it was assigned to. Client k must be a Pepsi-only client. Because we separated the clustering phase of anything-will-do clients and other clients, so k must be a Pepsi-only client or a Coke-only client. But it cannot be Coke-only, because otherwise $N(k)$ are all Coke facilities and it will not intersect with $N(j)$ which consists of only Pepsi facilities. As a result, we know k is assigned to an open Pepsi facility which can also serve j. The same argument will work for a Coke-only client.

2.4.2 The analysis

Now, we can analyze the total cost of our algorithm. Since we open facilities z_v with distribution λ_v,

Lemma 2.4.2. *The expected facility opening cost is exactly $\sum_{i\in\mathcal{F}} f_i y_i^*$.*

Now let us consider assignment cost. For any center-client j, we know that $i \in N(j)$ is opened with probability $y_i^* = x_{ij}^*$, so the expected assignment cost of j is exactly $C^*(j)$. For a non-center client, we have this lemma.

Lemma 2.4.3. *The expected assignment cost of a non-center client j is at most $2v_j^* + C^*(j)$.*

Proof. Take any non-center client j; from the last section, we know that there is a center client k that prevented us from making j a center. This means that there is a facility $i \in N(j) \cap N(k)$. Suppose that during the rounding phase, we opened facility $\ell \in N(k)$. Then we will assign j to ℓ. We have $c_{j\ell} \leq c_{ij} + c_{ik} + c_{\ell k}$. By complementary slackness of optimal LP solution, we know that $c_{ij} \leq v_j^*, c_{ik} \leq v_k^*$, so $c_{j\ell} \leq v_j^* + v_k^* + c_{\ell k}$. Because $c_{\ell k}$ has expectation $C^*(k)$, we know

$E[c_{j\ell}] \leq v_j^* + v_k^* + E[c_{\ell k}] = v_j^* + v_k^* + C^*(k)$. Since we considered k before j, we have $v_k^* + C^*(k) \leq v_j^* + C^*(j)$, thus, $E[c_{j\ell}] \leq 2v_j^* + C^*(j)$. □

With all this, we know the total expected cost is at most

$$\sum_{i \in \mathcal{F}} f_i y_i^* + \sum_{j \in \mathcal{D}} C^*(j) + \sum_{j \notin D} 2v_j^* \leq 3OPT.$$

Note that instead of randomly sampling a facility to open, we can try each z_v and pick the one with lowest total cost. This will guarantee to no more than the expected total cost. In other words, this derandomizes our algorithm and proves the main theorem.

Theorem 2.4.4. *There is a deterministic polynomial 3-approximation algorithm for Pepsi-Coke facility location problem.*

2.5 Lot-sizing with hierarchical types

In this section, we consider a related problem where facilities and clients are embedded in time, and thus may correspond to orders and demands occurring over time. Let order i occur at time $\tau(i)$ and demand j occur at time $\tau(j)$. We still have the hierarchical type tree as in *facility location with hierarchical types*. For a client j to be served by facility i, in addition to the type constraint $t(j) \leq t(i)$, we also have the time constraint, $\tau(i) \leq \tau(j)$. For the assignment cost c_{ij}, we have a non-decreasing property: for two facilities i, i' and client j with $\tau(i) \leq \tau(i') \leq \tau(j)$, we have $c_{ij} \geq c_{i'j}$. Also, $c_{ij} = \infty$ if $\tau(i) > \tau(j)$.

2.5.1 A dynamic programming algorithm

We now describe a polynomial-time dynamic programming (DP) formulation to solve the lot-sizing problem with hierarchical types. For this, we need some new notation. Considering a type tree $\mathcal{T}$, we define $\mathcal{T}(t)$ to be the subtree of $\mathcal{T}$ rooted at t if $t \in \mathcal{T}$, and $\emptyset$ otherwise. We also use $C_{\mathcal{T}}(t)$ to denote the set of children of t in $\mathcal{T}$. For convenience we add a dummy facility χ of the root type occurring at $\tau(\chi) = -\infty$ with opening cost $f_\chi = 0$ such that $c_{\chi j} = \infty$ for any client j.

Now, for any facility i, any type t satisfying $t \leq t(i)$, and any time $\tau > \tau(i)$ we define $\mathcal{L}(i, \mathcal{T}(t), \tau)$ to be the cost of the optimal solution for the subset of the clients and facilities contained in the interval $[\tau(i), \tau)$ that have types contained in the subtree rooted at t, i.e., $\mathcal{T}(t)$, with the extra assumption that i has opening cost $f_i = 0$. Thus, the optimal overall cost of the lot-sizing problem is $\mathcal{L}(\chi, \mathcal{T}, \infty)$. Note that there are at most $|\mathcal{D}|$ values for the third element that are of interest, and therefore there are a total of $O(|\mathcal{F}||\mathcal{T}||\mathcal{D}|)$ distinct sub-problems.

The base case of the DP is when the type tree has depth zero, and therefore the optimal solution has cost 0. Let us calculate $\mathcal{L}$ using a recursive formula that uses induction on the depth of the type tree, each time finding the latest facility of the root type that is opened in the optimal solution. For this, we can establish the following recurrence.

$$\mathcal{L}(i, \mathcal{T}(t), \tau) = \min\Big\{ \sum_{j : t(j) = t, \tau(i) \leq \tau(j) < \tau} c_{ij} + \sum_{t' \in C_{\mathcal{T}}(t)} \mathcal{L}(i, \mathcal{T}(t'), \tau),$$

$$\min_{i' : t(i') = t, \tau(i) < \tau(i') < \tau} \{ f_{i'} + \mathcal{L}(i, \mathcal{T}(t), \tau(i')) + \sum_{t' \in C_{\mathcal{T}}(t)} \mathcal{L}(\tau(i'), \mathcal{T}(t'), \tau) \} \Big\}$$

The first case of the outer min corresponds to the scenario when no facilities of type t are opened in the optimal solution in the interval $[\tau(i), \tau)$, other than i. Here, all clients of type t will be served by i, and the rest of the solution can be separately calculated for each of the type trees $T(t')$ for all $t' \in C_{\mathcal{T}}(t)$, and be summed to get the total cost.

The second case considers the facility i' that is the last facility of type t opened after i and by time τ. Here, the optimal solution for the interval $[\tau(i'), \tau)$ is calculated as in the previous case, and the optimal solution for the interval $[\tau(i), \tau(i')]$ is calculated by recursing on $\mathcal{L}(\tau(i'), \mathcal{T}(t'), \tau)$.

Together, the two cases cover all possibilities for the latest facility of type t that is opened in the time interval $[\tau(i), \tau)$ and therefore at some point the optimal solution is considered. Each update can be implemented in $O(|\mathcal{F}|+|\mathcal{D}|)$ given that the facilites and clients of each type are maintained sorted by occurence time, and therefore the total runtime is bounded by $O(|\mathcal{F}|^2|\mathcal{T}||\mathcal{D}| + |\mathcal{F}||\mathcal{T}||\mathcal{D}|^2)$.

Theorem 2.5.1. *There exists a polynomial-time dynamic programming algorithm to solve the hierarchical lot-sizing problem.*

2.5.2 An integrality gap example

Interestingly, although this problem admits polynomial solution, we can provide an example where the natural linear program has an integrability gap of $\frac{3}{4}$. We need only two types, say a and b, where a is the root type and b is the only leaf type.

The instance consists of three facilities f_1, f_2, and f_3 and three clients c_1, c_2,

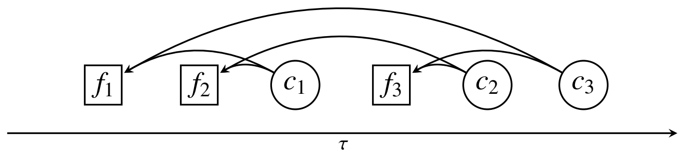

Figure 2.1: The instance of the lot-sizing problem inducing integrality gap of 4/3 for LP and the corresponding support of x. All drawn arcs correspond to half integral values.

and c_3 such that $\tau(f_1) < \tau(f_2) < \tau(c_1) < \tau(f_3) < \tau(c_2) < \tau(c_3)$. Also, facilities f_1 and f_3 and the client c_3 are of type a and the rest of clients and facilities are of type b. Let us say the opening cost for all facilities is a constant c and the holding costs are 0, except for $c_{f_1 c_2} = \infty$. It is easy to verify that these holding costs are non-decreasing.

Any solution that opens two of the facilities will have a total cost of $2c$ and would be optimal. However, in a fractional solution one can open each facility halfway since each client can be served by two facilities without paying a holding cost, i.e., $y_{f_1} = y_{f_2} = y_{f_3} = 1/2$. See Figure 2.1 for an illustration. Such a fractional solution satisfies all constraints of LP and has a cost of $3\frac{c}{2}$, inducing an integrality gap of $\frac{4}{3}$.

Theorem 2.5.2. *The natural LP for lot sizing has integrality gap* $\geq \frac{4}{3}$.

CHAPTER 3

RESOURCE SHARING IN APPOINTMENT SYSTEMS

3.1 Introduction

New York Presbyterian (NYP) Hospital operates a large outpatient MRI service that consists of three nearby sites within Manhattan, with 7 machines in total. They provide services, to hundreds of types of scans and they process roughly 100 patients per day. However, patients sometimes experience long waiting time -about 20% of all patients wait more than 60 minutes. Furthermore, there are still unfinished patients at the closing time of a site, so sometimes the staffs there needs to work a significant amount of overtime. In this chapter, we discuss a real-time queuing control policy to address these issues, and a practical implementation to make this feasible, which takes advantage of the modern-day reachabilities of patients just prior to their scheduled appointment.

Patients need to make appointments in advance to reserve their slots. The slot size, which is of length always an integer multiple of 15 minutes, depends on the type of scan that the patient is scheduling. Due to the high demand of MRI services, these machines often have tight schedules. Together with the high variability in the procedures (which we shall discuss in details below), these tight schedules cause long waiting times and substantial overtime to occur.

There are several types of uncertainties that can cause reality to deviate from the original schedule. There are some Same-Day Add-on Patients (SDAOP) that will show up without appointments. These are mostly urgent cases that need to be taken care of. There are also some same-day cancellations.

The most prevalent uncertainty is associated with scan duration and patient arrival time. MRI scans usually consist of about a dozen of image series, each with different scanning parameters. If the patient being scanned moves too much during any series, the images produced might be too blurred. In this case, the staff will have to redo this series. Each series can take minutes, so redoing one will lengthen the scan duration by a non-trivial amount. Furthermore, the scan procedure involves many human factors. Patients might need extra time to prepare themselves in the scan bed, and some even experience claustrophobia. Patients may need to use the restroom during scans. Even the number of images to acquire depends on the body size of the patients, and varies case by case.

Patient arrival time is another source of uncertainty. Although most patients arrive on or before their appointment time, some may be significantly late. This can be caused by bad traffic or the patient simply leaves home late. Some patients need medical care while being transported, and they will use Access-A-Ride which provides such medical care. However, since Access-A-Ride accommodates multiple patients per day and traffic in Manhattan is unpredictable, patients might come to their appointments significantly early or late.

The staff usually refers to the situation of running behind schedule as congestion. When significant congestion happens at one site, the staff there will call other sites to see whether they can handle more patients. If so, they may choose to send patients to other sites using taxi. This practice may help them balance loads among sites, but it also suffers from several drawbacks. First, the decision is made solely based on current site status. Sometimes, the other site is already congested by the time that the transported patient arrives there. Second, after spending a long time waiting at one site, the patient needs to be transported to another site. This brings inconvenience to both patients and staff.

Inspired by their current practice, we developed a real-time queuing control approach to pool resources across the three sites together. Instead of diverting patients after congestion develops, we predict when and where congestion will happen and react by diverting future patients to other sites. Frequently, this can prevent congestion to happen. Also, since we are diverting future patients who haven not shown up yet, they can go directly to the other site. This saves time and effort for both patients and staff. We will describe our approach in details and show computational results. In fact, our simulation result based on 3 years of historical data shows that diverting just 3 patients per day can lead to significant improvement in the number of patients experiencing very long waiting time staff overtime.

The most closely related literature to our problem is in appointment scheduling. There are excellent surveys on this topic [16, 8]. In this setting, usually it is assumed that the health-care providers already know about patients that must be seen on a particular day, together with all stochastic characteristics associated with each patient. The question is to decide the optimal schedule (appointment times of each patient) to balance patient waiting time, resource idle time and overtime. Several articles [19, 13, 6] have studied this problem and proposed optimal algorithms under certain assumptions. However, the results are not very applicable in our case. The reason is that we are working with outpatient facility, and we cannot really decide appointment times for patients. Nevertheless, important insights can be drawn from the literature to help make better schedules.

3.2 System dynamics

Our approach is to divert patients proactively to prevent congestion from happening. If we identify a beneficial diversion, we will call the patient l time before his appointment time and offer the patient the opportunity to be diverted, when the parameter l is a lead time that can be adjusted. The patient can accept or decline this diversion. In fact, we propose a system where a patient must first "opt in", or volunteers to be called for a potential diversion. The patient's "volunteer" status is determined before the scan day and only volunteers will be diverted. To make sure patients are incentivized to cooperate, we aim to pick diversions that will make the patient to be diverted wait less.

We use a discrete-event system (see [20] for reference) to describe our system. We are going to describe the entities and their associated states. Then, we will describe all of the events and how they change system states. Finally, we will describe the inputs of this model and how we use data to fit these inputs.

3.2.1 Entities and states

There are multiple sites in the system:

- Each site has one or more machines.
- Each machine can process at most one patient at any time. Once a scan is started, it cannot be preempted (that is, that patient's scan must be completed before the next patient's scan to start)
- There is a single waiting room at each site to accommodate patients who

have arrived but have not yet started their scans.

- Each site has a opening time and a closing time for each day. The site is supposed to only operate only inside this time window.

We make a simplifying assumption here: all machines are homogeneous and are capable to conduct any type of scan. In reality, this is not true, but we believe the effects of the few cases in which this becomes relevant do not justify the complicating in our overall model.

The central entity is an appointment. Each appointment is associated with one patient and has the following states:

- Appointment time
- Slot size
- Appointment status. This could be
  - ”scheduled but has not shown up”
  - ”arrived and preparing for scan”
  - ”ready for scan”
  - ”under scan”
  - ”scan completed”
  - ”cancelled”
- Assigned site. This may change due to diversion.
- Type of scan.
- Machine occupied. Only valid when patient is ”under scan”.
- Various timestamps

  - Arrival time
  - Ready time
  - Scan begin time
  - Scan completion time

Same-Day Add-On Patient (SDAOP) will appear at a site without an appointment. Similarly in our system, there is no appointment for them until their arrival.

Last but not least, we also track the current time in the system.

3.2.2 Events

The following a complete enumeration of all events that we consider:

- Patient arrival. This happens when a patient arrives at the assigned site. Note this can be a SDAOP.
- Patient ready. This happens when a patient has finished his preparation after arriving at the site. The preparation may include completing paperwork, changing clothes and being put on an intravenous therapy (IV).
- Cancellation. This happens when a patient notifies the hospital that he will not fulfill the appointment.
- Scan begin. When a machine is idling, the patient who is ready with earliest appointment time will be started.
- Scan completion.

- Diversion. This happens when we successfully divert a patient to another site.

In numerical experiments, we shall assume there are no ”no-shows”. In practice, a no-show is quite rare since it is not easy to get an appointment, and we believe this assumption is appropriate. In practice, we can deal with a no-show by changing the appointment’s status to ”cancelled” when the patient does not arrive within a certain threshold of the appointment time. If the patient does show up later, we will change the status to ”arrived and preparing for scan”.

In numerical experiments, we also assume all diversions are successful. The justification for this is that we will experiment with different fractions of volunteers and a low fraction of volunteers can be partially interpreted as unsuccessful diversions.

3.2.3 Model inputs

The model needs the following inputs:

- the schedule at the beginning of the day, including appointments across all sites;
- the set of patients who are volunteers;
- the distribution of arrival time per appointment;
- the distribution of preparation time per appointment;
- the distribution of scan duration per appointment;
- the set of SDAOPs and their arrival times;

- cancellations.

Here we describe an ideal scenario where we know the distributional information for each appointment. In practice, we need to use historical data to fit these distributions and we cannot hope to reach the granularity of the per appointment level. For scan duration, we fit one distribution for each type of scan. For arrival time and preparation time, we fit one distribution for the whole population.

We are making (admittedly substantial) assumptions on these sources of randomness. For example, when a site is running behind schedule, the staff may work faster. In fact, scan duration is dependent on the current status of a site (in terms of the congestion being observed). There are many factors in reality and we cannot hope to model all of them. So we focus on constructing a model that is qualitatively similar enough to reality, and show that our results are robust under a range of parameters influencing the behavior of the system.

These model inputs are used to simulate daily operations, and some of them are not available to our diversion policy. For example, our policy does not know about SDAOPs and cancellation until they happen.

We will discuss how to fit these inputs in the section on numerical experiments.

3.2.4 Metric

There are two things we care about: patient waiting time and site overtime. For one patient, if the appointment time is a and the scan begins at time t, then waiting time is defined to be

$$(t-a)^+.$$

For each machine, the overtime o is defined to be the amount of time this machine is working outside the operating time window of the site where the machine resides.

We are not only interested in average waiting time and overtime, we are interested in the distribution of waiting time and overtime. For example, we pay particular attention to extreme waiting times (for example, ≥60 minutes or ≥90 minutes of waiting time) since they are very important for the service quality perceived by patients. As a result, we will demonstrate the impact of diversions via multiple statistics on waiting time and overtime.

3.3 Policy

In this section, we are concerned with the policy we use to identify beneficial diversions. Before going into details, we will first discuss some potential benefits of diversions. First, we can at least balance workload between sites and let them share the pain when congestion happens. This alone can reduce extreme waiting time and extreme overtime. Second, we can make more efficient use of machine time. The reason for this is that it allows us to make use of the slot of canceled appointment or any gap in schedule;when a site is on schedule, idle time can develop whenever a scan finishes sooner than expected. But when a site is behind schedule, idle time is unlikely to occur. Thus, when we let sites share the pain, we can make more efficient use of machine time across all sites.

Now we shall discuss the details how we identify beneficial diversions.

3.3.1 Objective

At the end of the day, given a outcome δ, we wish to measure its desirability with an objective function. Let $W(\delta)$ be the waiting time vector (one component per patient) and $O(\delta)$ be the overtime vector (one component per machine). We define the following objective

$$C(\delta) = \|W(\delta)\|_2 + \lambda\|O(\delta)\|_2,$$

where λ is a parameter controlling how much weight we put on overtime compared to waiting time.

This objective captures both metrics of interest. The choice of L_2-norm here expresses our preference for eliminating cases of extremely long waiting time and extremely long overtime. This will also encourage diversions that allow sites to share the pain.

Another aim of this objective function is to prevent the system from diverting patients to sites that are supposed to be closed. If we use L_1-norm, this might not be achieved. Suppose that Site A closes earlier than Site B. If Site B is congested, diversion to Site A when it is supposed to be closed can keep the total overtime the same; while reducing waiting time. However, L_2-norm can prevent this from happening (to a larger extent when compared with the L_1-norm).

3.3.2 Choosing diversions

For our given lead time l, l time units before the appointment time of each patient who is a volunteer, we need to decide whether to divert this patient to another site. However, the outcome is stochastic in nature whether we divert the patient or not. Consider a diversion D. Suppose without D (and without any future diversions), we will reach random outcome δ. With D (but still no future diversions), we will reach random outcome δ'. Let $P(\delta), P(\delta')$ be the random waiting times for the patient to be diverted. We say D is a *valid* diversion if both

$$p_c = \mathbb{P}[C(\delta') < C(\delta)] \geq \theta_c$$
$$p_p = \mathbb{P}[P(\delta') < P(\delta)] \geq \theta_p$$

where θ_c, θ_p are two thresholds controlling how aggressive/conservative we are. High thresholds ensure we will see improvement for sure, but there might be only very few diversion options.

The criterion is to make sure a diversion will lead to an improved objective value and less waiting time for a diverted patient with sufficiently high probability.

It is impossible to compute $\mathbb{P}[C(\delta') < C(\delta)]$ and $\mathbb{P}[P(\delta') < P(\delta)]$ precisely. So, we need to approximate them, we decide to use simulation to estimate these probabilities.

We will generate a random vector ξ that includes

- the scan durations of all patients;
- the arrival time of all patients; and
- the preparation time of all patients.

We sample all of these from the corresponding empirical distribution, possibly conditioning on information in hand at current time (for example, for an in-progress patient, we know how long the patient has been under scan). We omit cancellation and SDAOP since they are low probability events and hard to predict.

With ξ, we can simulate the system with/without diversion D and assess the outcome. To make the estimation more accurate, we will, in total, generate k random vectors $\xi_1, \ldots, \xi_k$, each corresponding to one scenario. And let $\delta(\xi_1), \ldots, \delta(\xi_k)$ be outcomes without D and $\delta'(\xi_1), \ldots, \delta'(\xi_k)$ be outcomes with D. We use the fraction of scenarios where $C(\delta'(\xi_i)) < C(\delta'(\xi_i))$ as the estimated objective improvement probability $\tilde{p}_c$. We use the fraction of scenarios where $P(\delta'(\xi_i)) < P(\delta'(\xi_i))$ as the estimated less waiting time (for a diverted patient) probability $\tilde{p}_p$. If $\tilde{p}_c \geq \theta_c$ and $\tilde{p}_p \geq \theta_p$, we say this diversion is *approximately valid*. Here, we are comparing pairs of outcomes coupled with same random vectors. This technique is called Common Random Numbers and can help reduce variance when comparing outcomes.

If there is no *approximately valid diversion*, we will keep the patient at his/her current site. Otherwise, we will choose the diversion with highest estimated objective improvement probability $\tilde{p}_c$.

This approach basically greedily diverts patients. This is obviously suboptimal. However, modeling this decision problem as Stochastic Dynamic Programming leads to a formulation that is very hard to solve. The simple approach we take is easy to compute and easy to interpret. It is also extensible that we can incorporate other desirable property in choosing diversions.

3.4 Computational Results

We now present computational results for our diversion policy. Note that we are omitting confidence intervals for the plots in this section. There are two reasons for this. First, the plots in this section mostly contain multiple lines, and so confidence intervals will substantially clutter the plots. Second, for showing the statistical significance of the improvement, it is more proper to plot the change of performance with confidence intervals rather than to plot the performance itself. We give two example plots at the end of this section, together with two tables showing the change of performance with (95%) confidence intervals.

3.4.1 Experimental Setup

Scenarios In this section, we describe the two scenarios that we experiment with, the historical data we have, and the choice of parameters for our policy.

The first scenario is based on the real-life case of New York Presbyterian (NYP) Hospital. NYP Hospital has three different sites for outpatient MRI across Manhattan. They are named after their addresses: York Ave., 55th St. and West 84th St. (See Figure 3.1). The first two sites are very close to each other; they are only a 15 minute walking distance apart. The one on the west side is a bit further; it is a 15 minute driving distance without traffic from the other two. A patient usually arrives at the site 30 minutes before his appointment time to prepare, so that he can be ready by his appointment time. Thus, if we call the patient 1 hour before his appointment time, he would have 30 minutes to react to the change. This might be reasonable given the short distance

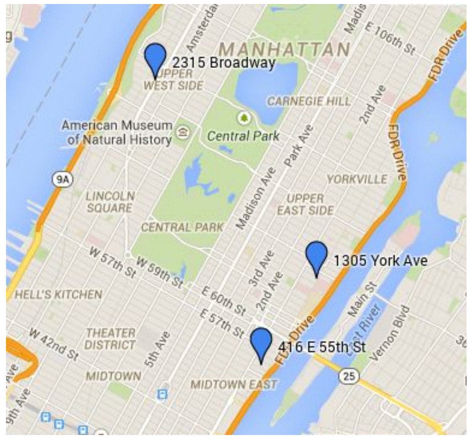

Figure 3.1: Locations of MRI facilities of New York Presbyterian. There are three in total represented as blue bubbles. Two of them are on the east side and one is on the west side.

between the three sites. We also experiment with 90 minutes lead time, which gives the patient 60 minutes to react.

These sites have different numbers of MRI machines. West 84th has only one machine. 55th St. has two machines. York Ave. has four machines. Since they have different processing capacity, a different number of appointments are scheduled for each. With 4 machines at York Ave, there is a substantial amount of resource pooling ability present, making patients there in general wait less.

The second scenario we consider is a system with 7 sites in which each has one MRI machine. The reason is that we want to explore the impact of diversion on a set of small cooperating imaging facilities. Also, in this case, all sites are homogeneous, so we can isolate the impact of resource pooling without worrying about imbalance between processing capacities across sites.

Data We have three years of historical data for MRI scans at NYP Hospital. For each of these scans, we have the type of scan, its appointment time, when the patient arrived, when the scan began, and when the scan finished. The data we have is not perfect, but we are able to fit reasonable model inputs based on it.

- Initial schedule for the day. We generate synthetic schedules where the number of scans of each type is proportional to the number of historical scans of that type.
- The set of patients who are volunteers. We make the fraction of volunteers a parameter that varies across experiments. This can help us understand the amount of impact that diversions have with different levels of flexibility from patients.
- Patient arrival time. We fit one empirical distribution on the difference between patient's arrival time and appointment time.
- Preparation time. We do not actually have the timestamp data for this. We overcome this difficulty by considering the first patient processed each day, and use the difference between their arrival time and scan begin time as the preparation time. Since the first patient does not need to stay in the waiting room, it is reasonable to assume that all of the time between the arrival time and scan begin time is used for preparation. We fit one empirical distribution based on this.
- Scan duration. For each type of scan, we fit an empirical scan duration distribution from historical data.
- The set of SDAOPs and their arrival time. We calculate the rate of SDAOPs in the history and generate SDAOPs using a Possion process.
- Cancellations. We calculate the historical probability of cancellation, and randomly select appointments that will be cancelled later.

Ideally, we would like to test our policy by replaying history with the real scan duration and actual arrival time for each historical appointment. However, the data we obtained is incomplete. There are some critical fields missing and we cannot exactly reconstruct history. Furthermore, there are occasional errors on timestamps that can greatly impact how the daily schedule appears. To prevent these data issues from interfering with our experiments, we chose to experiment with our constructed synthetic schedules.

One might improve the model inputs by predicting the scan duration and arrival time better using more information. For example, the patient's health condition, diagnosis, home address and other information might be helpful to make a better prediction. However, we do not have access to these sensitive personal data. But we believe that if we could use machine learning techniques to make better predictions of patient's arrival time and scan duration, our diversion policy would perform better.

Parameters There are several parameters we need to choose for setting up our simulation and for our optimization procedure. We describe our choice of them here.

- Volunteer probability. We vary the percent of patients who are willing to be volunteers. This will allow us to see the effect of our policy with varying levels of flexibility.
- Lead time. We experiment with both a 60 minute and a 90 minute lead time.
- Overtime weight $\lambda = 10$, this reflects the fact overtime is quite valuable.

- $\theta_c = 0.7, \theta_p = 0.7$. We choose the confidence threshold on objective value and waiting time of diverted patient to be neither too aggressive nor too conservative.
- The number of samples to evaluate quality of diversions $k = 100$.
- The number of simulated days to estimate policy performance per experiment $k' = 100$.

3.4.2 3 Sites Results

We first examine the impact of diversions with a 60 minute lead time. Figure 3.2 shows how extreme waiting cases are reduced. The first thing to notice is that different sites, when no patients volunteer, and hence our policy has no effect, already perform quite differently. This is caused by the different number of machines at each site. York Ave. has 4 machines, and so it already enjoys substantial pooling effect. This is why patients generally wait less there. However, as we can see in Figure 3.2, diversions reduce extreme waiting cases for all sites, most prominently at sites with fewer machines.

Since we are optimizing the L_2-norm of the waiting time vector, it is understandable that we can reduce extreme waiting time, potentially by increasing waiting time for other patients. Hence, it is interesting to look at how we are doing with respect to mean waiting time.

Figure 3.3 shows that we actually reduced mean waiting time for all sites and the reduction for 84th St. and 55th St. is again quite significant. Not only that, we also reduced mean overtime for all sites, and so the staff can go home

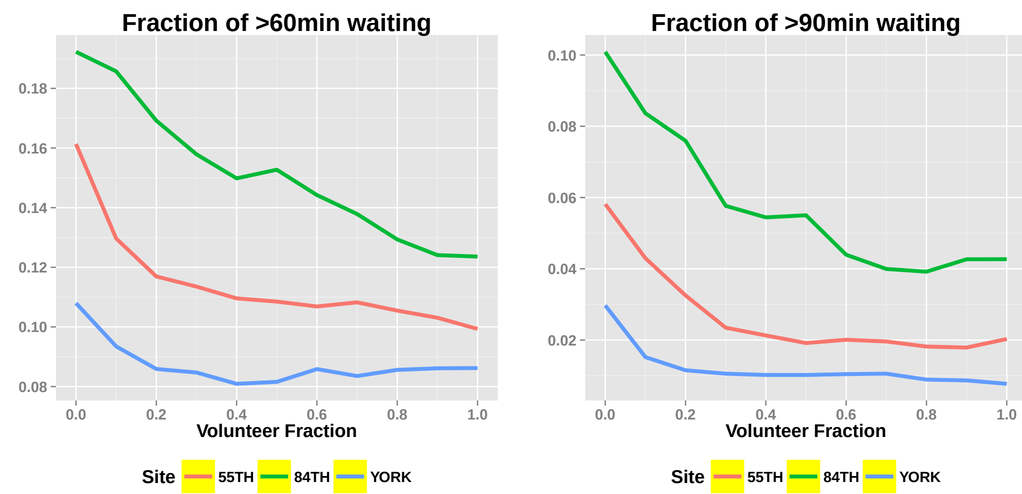

Figure 3.2: The impact on extreme waiting by diversions.

earlier. This means that we can finish the same amount of work within a shorter amount of working time.

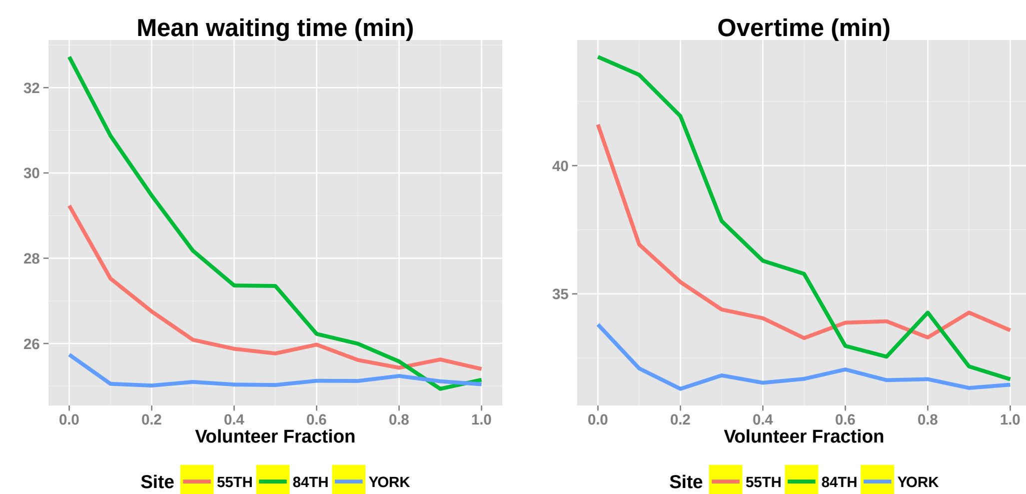

Figure 3.3: The impact on mean waiting times and mean site overtimes by diversions.

The explanation for why all three of these metrics improve can be seen in Figure 3.4. By diverting patients to a non-congested site, we actually reduce idle time there. Because we have made use of some machine time that would be otherwise wasted in idling, we can finish more work in the same amount of time, resulting in less backlog and congestion.

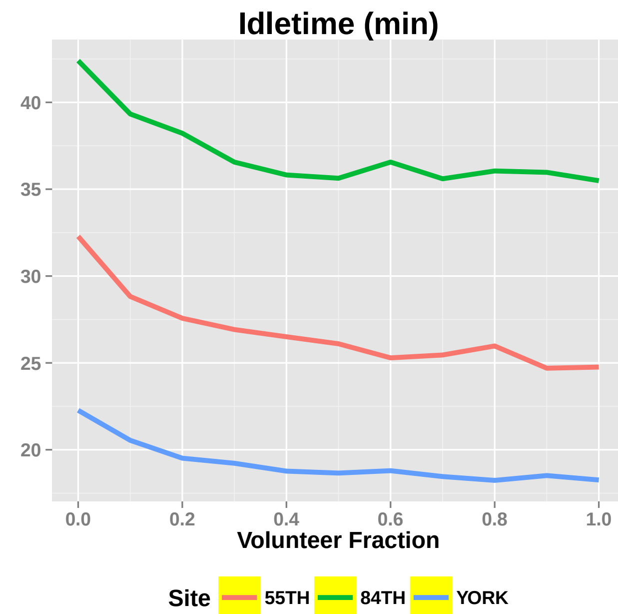

Figure 3.4: The impact on mean idle times.

Figure 3.5 shows that the most diversions are made diverting a patient into York Ave, and patients who are diverted into York Ave. enjoy the greatest reduction in waiting time. This is quite intuitive since the pooling effect at York Ave. gives it more processing capability. Consequently, it has more slack to take in additional patients than the other two sites have. It is important to note that no matter which site the patient is diverted to, he always enjoys significant reduction in waiting time: this should serve as great incentive for patients to volunteer provided that they have the flexibility.

Overall, we can see across all the figures, that the system performance improves fastest when we only have very small fraction of patients as volunteers. In other words, diversions can rapidly improve system performance with very

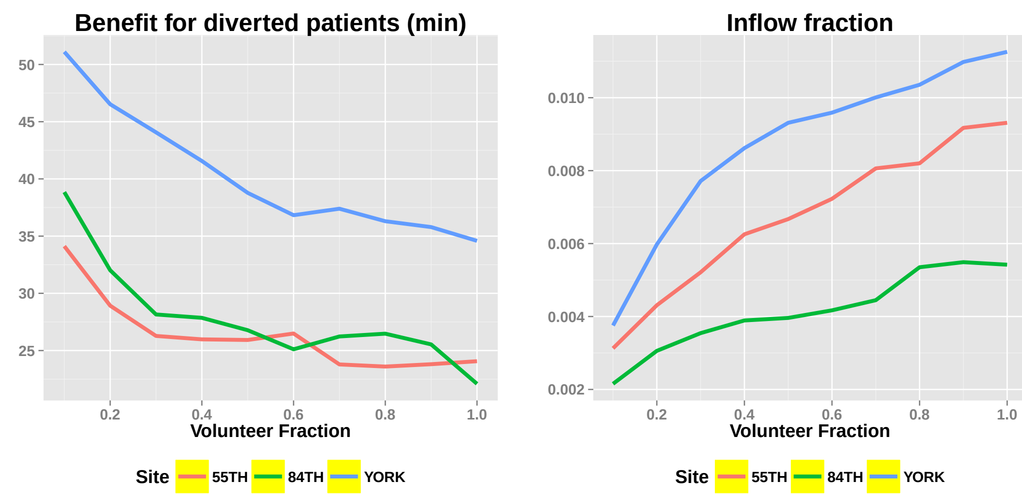

Figure 3.5: The amount of diversion into each site and reduction of waiting time enjoyed being diverted into each site.

little flexibility. This is desirable since benefits seen in a limited trial can give then practitioner more confidence to implement this more broadly.

Figure 3.6 shows the overall system performance for both a 60 minute lead time and a 90 minute lead time cases. The results closely mimic the discussion above. Overall, with a 60 minute lead time and 40% volunteers, we reduce the fraction of over 60 minute waiting from 13.5% to 9.7%, and the fraction of over 90 minute waiting from 4.7% to 1.8%. Both overtime and idle time are also significantly improved.

However, we would like to stress that even with a 90 minute lead time, the diversions still bring much improvement, and indeed not that different from the performance with a 60 minute lead time. This has a quite important practical implication, since a longer lead time makes it easier for patients to be diverted to different sites.

We also want to note that, as Figure 3.7 shows, we only diverted a very small

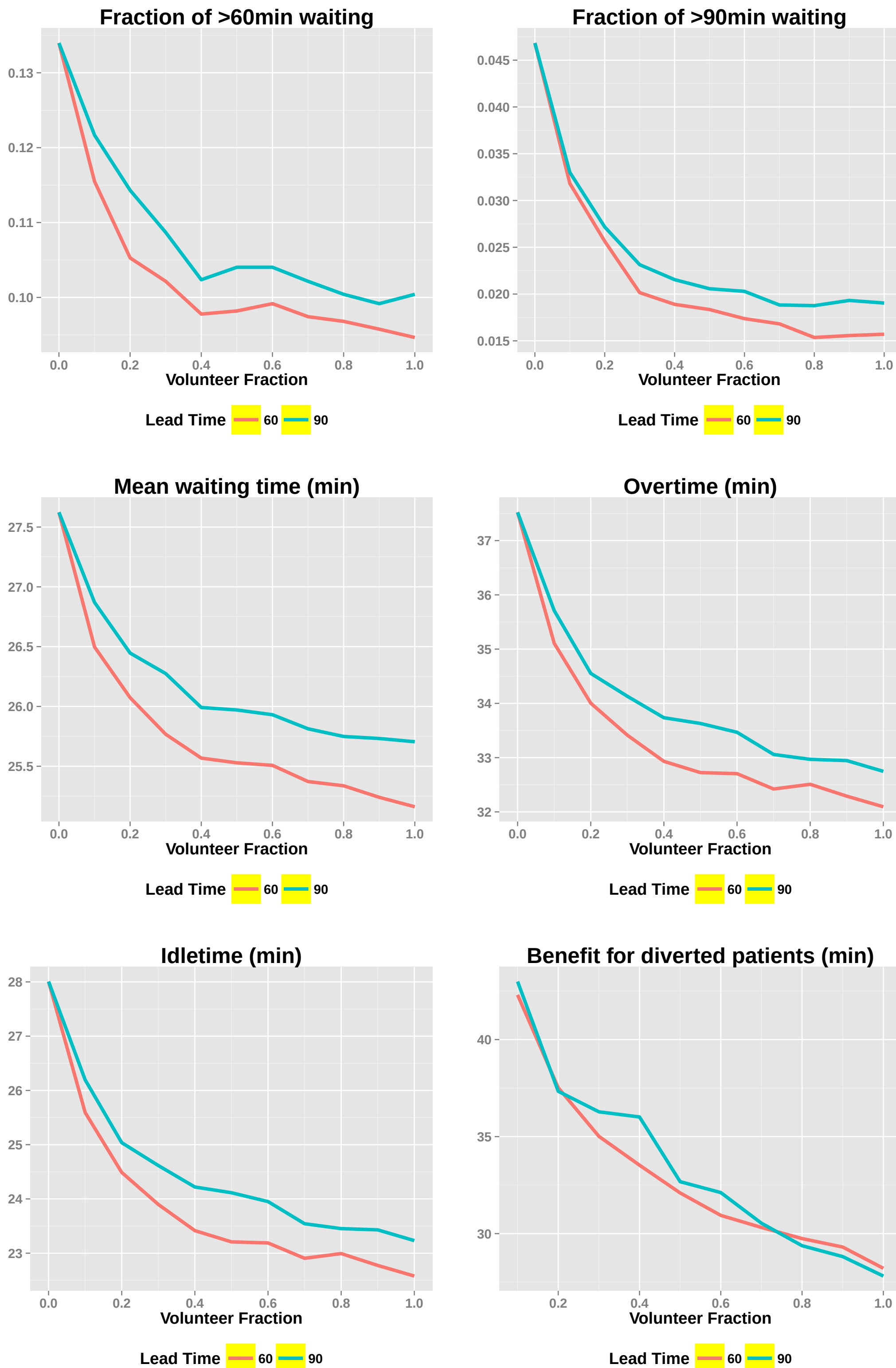

Figure 3.6: System performance improvement by diversion.

fraction of the patients. Even if all patients are volunteers, we only diverted fewer than 3% of them. We believe that this is a desirable property, since fewer changes are easier to implement. Also, since there are only 3 changes per day on average, the staff can use their domain knowledge to better facilitate those changes.

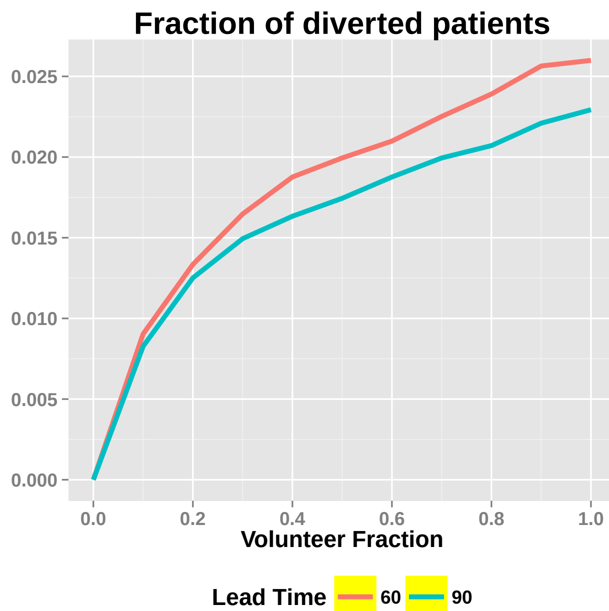

Figure 3.7: Fraction of patients diverted.

Overall, we think diversions are shown to be an efficient way to improve system performance for NYP Hospital's outpatient MRI sites. It can significantly improve patient waiting time and site overtime, without incurring too many diversions.

3.4.3 7 Sites Results

The 3 site case, although based on a real system currently functioning, already contains a pooling effect due to the fact that some sites have more than one machines. We would like to isolate the impact of just diversions in creating the pooling effect. Thus, it makes sense to conduct experiments on the case where instead of having some big sites and some small sites, we have 7 small sites, each with one machine. This system has worse performance initially because no pooling effect is present. We shall examine the impact of diversions. Note that all sites are totally homogeneous now: they have the same number of machines, the same amount of load and the same opening time. As a result, we do not need to look at system performance at a per site level.

Figure 3.8 shows that when compared to 3 sites case, diversions have a greater impact on this system. In this case, we have significant improvement even on mean waiting time. This is exactly because diversions provide the only source of a pooling effect.

From Figure 3.9, we see that we may potentially divert 6% patients if everyone volunteers. This is substantially more when compared with 3 sites case. This shows that there are many more beneficial diversion opportunities when there is no pooling initially. Also, for diverted patients, each of them enjoys a greater reduction in waiting time when compared to the 3 sites case.

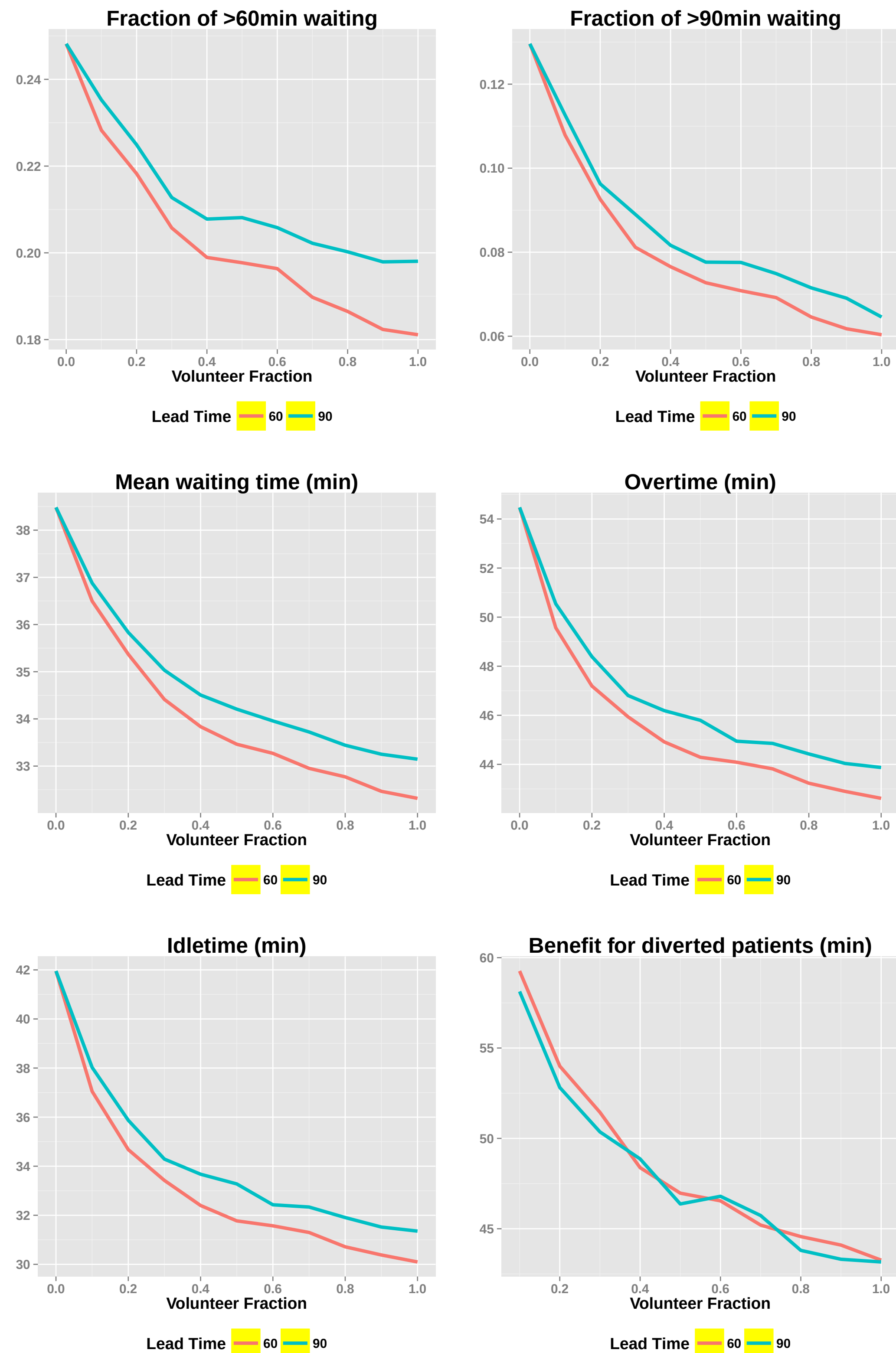

Figure 3.8: The impact of diversions on system performance of a system with 7 sites, each with one machine.

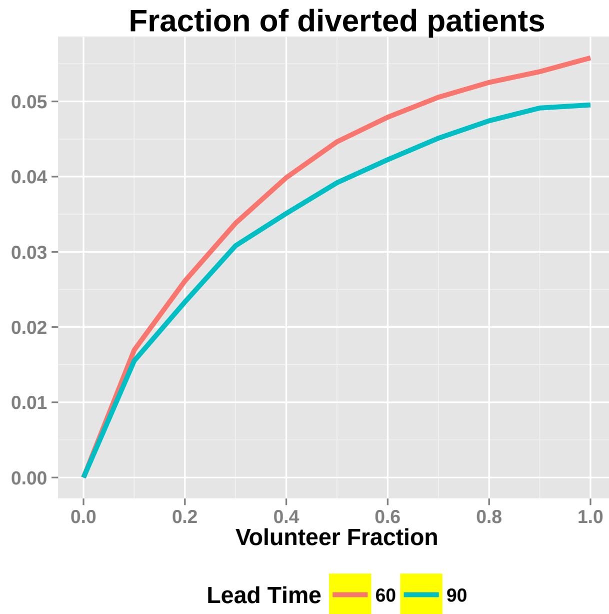

Figure 3.9: Fraction of diverted patients in 7 sites case.

3.4.4 On confidence intervals

Figure 3.10 shows the impact of our diversion policy for the 3 site case with a 60 minute lead time, by examining the change of performance instead of simply the performance. The confidence intervals show that the performance improvement from our diversion policy is statistically significant. Also, note that as we increase the fraction of volunteers, the variance also increases. The reason is that, when there are more volunteers, there are more diversions which induce greater variance. If we examine the performance improvement for other metrics or for other levels of granularity, similar patterns can be observed.

For more information, see Table 3.1 and Table 3.2.

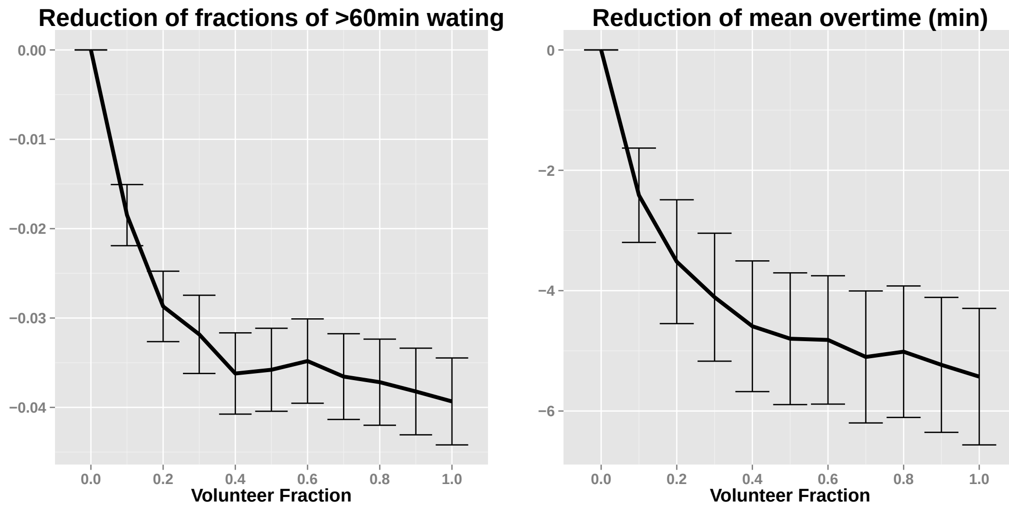

Figure 3.10: The impact of diversion policy for the 3 site case with a 60 minute lead time. The left plot is showing the reduction of fractions of over 60 minute waiting. The right plot is showing the reduction of mean overtime.

3.5 Continuity of system performance with respect to lead time

In the experiment, we considered two lead times. It is an interesting question to ask whether the system's performance changes continuously with respect to the lead time. Here we give a proof for the following similar but simplified setting:

- There are k sites, each with one machine.
- There are n patients. Patient i has appointment time a_i and will arrive punctually.
- The service duration d_i of patient i follows an exponential distribution, possibly with different means.

For a technical reason, we will first slightly modify our instance: we will use a new appointment time $\tilde{a}_i = a_i + \delta u_i$, where δ is a small constant and each u_i

Lead time	Frac. of volunteers	Change of >60min waiting %	Change of >90min waiting %	Change of overtime
60	0.0	0.0% +/- 0.0%	0.0% +/- 0.0%	0.0 +/- 0.0
60	0.1	-1.8% +/- 0.3%	-1.5% +/- 0.2%	-2.4 +/- 0.8
60	0.2	-2.9% +/- 0.4%	-2.1% +/- 0.3%	-3.5 +/- 1.0
60	0.3	-3.2% +/- 0.4%	-2.7% +/- 0.3%	-4.1 +/- 1.1
60	0.4	-3.6% +/- 0.5%	-2.8% +/- 0.3%	-4.6 +/- 1.1
60	0.5	-3.6% +/- 0.5%	-2.8% +/- 0.3%	-4.8 +/- 1.1
60	0.6	-3.5% +/- 0.5%	-2.9% +/- 0.3%	-4.8 +/- 1.1
60	0.7	-3.7% +/- 0.5%	-3.0% +/- 0.3%	-5.1 +/- 1.1
60	0.8	-3.7% +/- 0.5%	-3.1% +/- 0.3%	-5.0 +/- 1.1
60	0.9	-3.8% +/- 0.5%	-3.1% +/- 0.3%	-5.2 +/- 1.1
60	1.0	-3.9% +/- 0.5%	-3.1% +/- 0.3%	-5.4 +/- 1.1
90	0.0	0.0% +/- 0.0%	0.0% +/- 0.0%	0.0 +/- 0.0
90	0.1	-1.2% +/- 0.3%	-1.4% +/- 0.2%	-1.8 +/- 0.6
90	0.2	-2.0% +/- 0.4%	-2.0% +/- 0.3%	-3.0 +/- 0.9
90	0.3	-2.5% +/- 0.4%	-2.4% +/- 0.3%	-3.4 +/- 1.0
90	0.4	-3.2% +/- 0.4%	-2.5% +/- 0.3%	-3.8 +/- 1.0
90	0.5	-3.0% +/- 0.4%	-2.6% +/- 0.3%	-3.9 +/- 1.0
90	0.6	-3.0% +/- 0.4%	-2.7% +/- 0.3%	-4.1 +/- 1.0
90	0.7	-3.2% +/- 0.5%	-2.8% +/- 0.3%	-4.5 +/- 1.0
90	0.8	-3.4% +/- 0.5%	-2.8% +/- 0.3%	-4.6 +/- 1.0
90	0.9	-3.5% +/- 0.5%	-2.8% +/- 0.3%	-4.6 +/- 1.1
90	1.0	-3.4% +/- 0.5%	-2.8% +/- 0.3%	-4.8 +/- 1.1

Table 3.1: The performance improvement for 3 site case with confidence intervals.

is a uniform random variable on $[-1, 1]$. Now, all of the appointment times are different almost surely.

The system will follow the join-shortest-queue policy with lead time l. Consider the decision epoch at time $\tilde{a}_i - l$, when the system is deciding which site to assign patient i. Let $P_{ij}(l)$ be the set of patients that are assigned to site j, but have not completed their scans. The system will calculate, for each site j, the time $C_{ij}(l)$ (at least current time $\tilde{a}_i - l$) to finish patients $P_{ij}(l)$, assuming that the scan duration will take expected value. The system will assign patient i to the site with earliest $C_{ij}(l)$. If there is a tie, the patient will be assigned to the site

Lead time	Frac. of volunteers	Change of >60min waiting %	Change of >90min waiting %	Change of overtime
60	0.0	0.0% +/- 0.0%	0.0% +/- 0.0%	0.0 +/- 0.0
60	0.1	-2.0% +/- 0.4%	-2.2% +/- 0.3%	-4.9 +/- 1.0
60	0.2	-3.0% +/- 0.5%	-3.7% +/- 0.4%	-7.3 +/- 1.1
60	0.3	-4.2% +/- 0.5%	-4.8% +/- 0.4%	-8.5 +/- 1.3
60	0.4	-4.9% +/- 0.6%	-5.3% +/- 0.5%	-9.6 +/- 1.4
60	0.5	-5.0% +/- 0.6%	-5.7% +/- 0.5%	-10.2 +/- 1.4
60	0.6	-5.2% +/- 0.6%	-5.9% +/- 0.5%	-10.4 +/- 1.4
60	0.7	-5.8% +/- 0.6%	-6.0% +/- 0.5%	-10.7 +/- 1.4
60	0.8	-6.2% +/- 0.6%	-6.5% +/- 0.5%	-11.2 +/- 1.5
60	0.9	-6.6% +/- 0.7%	-6.8% +/- 0.5%	-11.6 +/- 1.4
60	1.0	-6.7% +/- 0.7%	-6.9% +/- 0.5%	-11.9 +/- 1.5
90	0.0	0.0% +/- 0.0%	0.0% +/- 0.0%	0.0 +/- 0.0
90	0.1	-1.3% +/- 0.4%	-1.7% +/- 0.3%	-3.9 +/- 0.9
90	0.2	-2.3% +/- 0.5%	-3.3% +/- 0.4%	-6.1 +/- 1.1
90	0.3	-3.5% +/- 0.5%	-4.1% +/- 0.4%	-7.7 +/- 1.2
90	0.4	-4.0% +/- 0.5%	-4.8% +/- 0.5%	-8.3 +/- 1.3
90	0.5	-4.0% +/- 0.6%	-5.2% +/- 0.5%	-8.7 +/- 1.3
90	0.6	-4.2% +/- 0.6%	-5.2% +/- 0.5%	-9.5 +/- 1.4
90	0.7	-4.6% +/- 0.6%	-5.5% +/- 0.5%	-9.6 +/- 1.4
90	0.8	-4.8% +/- 0.6%	-5.8% +/- 0.5%	-10.0 +/- 1.4
90	0.9	-5.0% +/- 0.6%	-6.1% +/- 0.5%	-10.4 +/- 1.4
90	1.0	-5.0% +/- 0.6%	-6.5% +/- 0.5%	-10.6 +/- 1.4

Table 3.2: The performance improvement for 7 site case with confidence intervals.

with lower number.

Let sample path ω include all of the randomness of u_i and d_i, and let $X(l)$ be the random total waiting time of all of the patients. We will prove the following theorem.

Theorem 3.5.1. $\mathbb{E}[X(l)]$ *is continuous with respect to lead time* $l \geq 0$.

Proof. We want to prove $\mathbb{E}[X(l+\epsilon)] \to \mathbb{E}[X(l)]$ when $\epsilon \to 0$. Number all of the patients from 1 to n in non-decreasing order of $\tilde{a}_i$. Note that the waiting time for any patient i can be bounded by the sum of scan durations of previous patients

$\sum_{i'<i} d_{i'}$. This implies that the total waiting time $X(l+\epsilon)$ can be bounded by $\sum_{i'=1}^{n} nd_{i'}$, which is a integrable random variable. Thus, if we can prove $X(l+\epsilon) \to X(l)$ almost surely as $\epsilon \to 0$, the theorem will follow from the Dominated Convergence Theorem.

Now fix a sample path ω and the lead time l. Consider the decision epoch for patient i. Unless there are two sites with empty assigned patients P_{ij}, there will almost surely be a non-zero gap between the earliest completion time $C_{ij}(l)$ and the rest. Let $g_1 > 0$ be the smallest such non-zero gap among all of the patients.

Consider the closest pair of decision epoch and moment when a scan is finished in time, and let g_2 be the time between them. Almost surely, $g_2 > 0$.

We claim that $X(l+\epsilon) = X(l)$, if $|\epsilon| < \min\{g_1, g_2\}$. Actually, we will prove that the two systems will evolve in exactly the same way. Suppose that on the contrary, the two systems diverge when making decisions for patient i. Since $|\epsilon| < g_2$, there is no scan completed between $\tilde{a}_i - l$ and $\tilde{a}_i - \epsilon$, thus $P_{ij}(l) = P_{ij}(l+\epsilon)$ for any site j. In other words, the two systems see the same set of assigned patients. If there is some site with no assigned patients $P_{ij}(l)$, both systems will choose such a site with lowest number. Otherwise, because $|\epsilon| < g_1$, the site with lowest $C_{ij}(l)$ still has lowest $C_{ij}(l+\epsilon)$. The two systems still make the same decision.

Thus, we have proved that the two system evolve exactly the same when ϵ is sufficiently small. This implies that $X(l+\epsilon) \to X(l)$ almost surely when $\epsilon \to 0$, which completes our proof. $\square$

3.6 Conclusion

In this chapter, we looked at the problem of congestion in an appointment system with multiple facilities. We explored the approach to make real-time diversions for resource sharing to balance demands. We investigate the simple approach of making one diversion at a time and use simulation and common random numbers to evaluate proposed changes. This approach is intuitive and simple to implement. Computational results show that this approach bring meaningful improvement leading to fewer cases of patients with extreme waiting times and decreased site overtime, while only a very small number of patients are diverted.

BIBLIOGRAPHY

[1] Daniel Adelman. Dynamic bid prices in revenue management. *Operations Research*, 55(4):647–661, 2007.

[2] Hyung-Chan An, Mohit Singh, and Ola Svensson. Lp-based algorithms for capacitated facility location. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, pages 256–265. IEEE, 2014.

[3] Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristics for k-median and facility location problems. *SIAM Journal on Computing*, 33(3):544–562, 2004.

[4] Manisha Bansal, Naveen Garg, and Neelima Gupta. A 5-approximation for capacitated facility location. In *Algorithms–ESA 2012*, pages 133–144. Springer, 2012.

[5] Siddharth Barman and Shuchi Chawla. Traffic-redundancy aware network design. *CoRR*, abs/1110.4150, 2011.

[6] Mehmet A Begen and Maurice Queyranne. Appointment scheduling with discrete random durations. *Mathematics of Operations Research*, 36(2):240–257, 2011.

[7] Marcin Bienkowski, Jarosaw Byrka, Marek Chrobak, Neil Dobbs, Tomasz Nowicki, Maxim Sviridenko, Grzegorz wirszcz, and NealE. Young. Approximation algorithms for the joint replenishment problem with deadlines. *Journal of Scheduling*, 18(6):545–560, 2015.

[8] Tugba Cayirli and Emre Veral. Outpatient scheduling in health care: A review of literature*. *Production and Operations Management*, 12(4):519, 2003.

[9] Moses Charikar and Sudipto Guha. Improved combinatorial algorithms for facility location problems. *SIAM Journal on Computing*, 34(4):803–824, 2005.

[10] Maurice Cheung, Adam N Elmachtoub, Retsef Levi, and David B Shmoys. The submodular joint replenishment problem. *unpublished*, 2014.

[11] Fabián A Chudak and David B Shmoys. Improved approximation algorithms for the uncapacitated facility location problem. *SIAM Journal on Computing*, 33(1):1–25, 2003.

[12] Fabián A Chudak and David P Williamson. Improved approximation algorithms for capacitated facility location problems. *Mathematical programming*, 102(2):207–222, 2005.

[13] Brian Denton and Diwakar Gupta. A sequential bounding approach for optimal appointment scheduling. *Iie Transactions*, 35(11):1003–1016, 2003.

[14] Vivek F Farias and Benjamin Van Roy. An approximate dynamic programming approach to network revenue management. *Preprint*, 2007.

[15] Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

[16] Diwakar Gupta and Brian Denton. Appointment scheduling in health care: Challenges and opportunities. *IIE transactions*, 40(9):800–819, 2008.

[17] Kamal Jain, Mohammad Mahdian, Evangelos Markakis, Amin Saberi, and Vijay V Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing lp. *Journal of the ACM (JACM)*, 50(6):795–824, 2003.

[18] Kamal Jain and Vijay V Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *Journal of the ACM (JACM)*, 48(2):274–296, 2001.

[19] Guido C Kaandorp and Ger Koole. Optimal outpatient appointment scheduling. *Health Care Management Science*, 10(3):217–229, 2007.

[20] W David Kelton and Averill M Law. *Simulation modeling and analysis*. McGraw Hill Boston, 2000.

[21] Madhukar R Korupolu, C Greg Plaxton, and Rajmohan Rajaraman. Analysis of a local search heuristic for facility location problems. *Journal of algorithms*, 37(1):146–188, 2000.

[22] Sumit Kunnumkal and Huseyin Topaloglu. Computing time-dependent bid prices in network revenue management problems. *Transportation Science*, 44(1):38–62, 2010.

[23] Retsef Levi, Andrea Lodi, and Maxim Sviridenko. Approximation algo-

rithms for the capacitated multi-item lot-sizing problem via flow-cover inequalities. *Mathematics of Operations Research*, 33(2):461–474, 2008.

[24] Retsef Levi, Robin Roundy, David Shmoys, and Maxim Sviridenko. A constant approximation algorithm for the one-warehouse multiretailer problem. *Management Science*, 54(4):763–776, 2008.

[25] Retsef Levi, Robin O Roundy, and David B Shmoys. Primal-dual algorithms for deterministic inventory problems. *Mathematics of Operations Research*, 31(2):267–284, 2006.

[26] Shi Li. A 1.488 approximation algorithm for the uncapacitated facility location problem. *Information and Computation*, 222:45–58, 2013.

[27] Shi Li and Ola Svensson. Approximating k-median via pseudo-approximation. In *proceedings of the forty-fifth annual ACM symposium on theory of computing*, pages 901–910. ACM, 2013.

[28] Joern Meissner and Arne Strauss. Network revenue management with inventory-sensitive bid prices and customer choice. *European Journal of Operational Research*, 216(2):459–468, 2012.

[29] Martin Pal, Éva Tardos, and Tom Wexler. Facility location with nonuniform hard capacities. In *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*, pages 329–338. IEEE, 2001.

[30] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[31] Andrzej P Ruszczyński. *Nonlinear optimization*, volume 13. Princeton university press, 2006.

[32] Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2003.

[33] David B Shmoys, Éva Tardos, and Karen Aardal. Approximation algorithms for facility location problems. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 265–274. ACM, 1997.

[34] Robert Warren Simpson. *Using network flow techniques to find shadow prices for market demands and seat inventory control*. MIT, Department of Aeronautics and Astronautics, Flight Transportation Laboratory, 1989.

[35] Chaitanya Swamy. Improved approximation algorithms for matroid and knapsack median problems and applications. *arXiv preprint arXiv:1310.7834*, 2013.

[36] Kalyan Talluri and Garrett Van Ryzin. An analysis of bid-price controls for network revenue management. *Management Science*, 44(11-part-1):1577–1593, 1998.

[37] Kalyan Talluri and Garrett Van Ryzin. A randomized linear programming method for computing network bid prices. *Transportation Science*, 33(2):207–216, 1999.

[38] Huseyin Topaloglu. Using lagrangian relaxation to compute capacity-dependent bid prices in network revenue management. *Operations Research*, 57(3):637–649, 2009.

[39] Robert J Vanderbei. Linear programming. *Foundations and Extensions. Second Edition-International Series in Operations Research and Management Science*, 37, 2001.

[40] Thomas WM Vossen and Dan Zhang. A dynamic disaggregation approach to approximate linear programs for network revenue management. *Production and Operations Management*, 24(3):469–487, 2015.

[41] David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge University Press, 2011.

[42] Elizabeth Louise Williamson. Airline network seat control. *Unpublished doctoral dissertation, Massachusetts Institute of Technology, Cambridge, MA*, 1992.

[43] Dan Zhang and Daniel Adelman. An approximate dynamic programming approach to network revenue management with customer choice. *Transportation Science*, 43(3):381–394, 2009.