



Approximation algorithms via the primal-dual schema: applications of the simple dual-ascent method to problems from logistics

by Timothy Alan Carnes

This thesis/dissertation document has been electronically approved by the following individuals:

Shmoys, David B (Chairperson)

Henderson, Shane G. (Minor Member)

Kleinberg, Robert David (Minor Member)

APPROXIMATION ALGORITHMS VIA THE
PRIMAL-DUAL SCHEMA: APPLICATIONS OF
THE SIMPLE DUAL-ASCENT METHOD TO
PROBLEMS FROM LOGISTICS

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

Timothy Alan Carnes

August 2010

© 2010 Timothy Alan Carnes

ALL RIGHTS RESERVED

APPROXIMATION ALGORITHMS VIA THE PRIMAL-DUAL SCHEMA:
APPLICATIONS OF THE SIMPLE DUAL-ASCENT METHOD TO PROBLEMS
FROM LOGISTICS

Timothy Alan Carnes, Ph.D.

Cornell University 2010

The primal-dual schema has been employed to provide approximation algorithms for a large variety of combinatorial optimization problems. This technique relies upon simultaneously constructing a primal integer solution that is feasible for the minimization problem of interest, as well as a dual solution whose objective function value serves as a lower bound. By operating in this framework, the advantages of using an LP-based method are obtained, such as generating an instance-dependent approximation ratio that may greatly improve on the theoretical worst-case, while obviating the need to actually solve an LP.

In this thesis, we focus on the application of the simple dual-ascent method, where the feasible dual solution is modified by increasing only a single dual variable at a time, and obtain approximation algorithms for the following problems: capacitated covering problems with lot-sizing applications, the location routing problem with a global limit on locations, and a variant of the generalized assignment problem. For each of these areas, we introduce an exponential number of valid inequalities to strengthen the LP-relaxation, thereby decreasing the integrality gap. As we are only increasing a single dual variable at a time, each of the primal constraints is effectively providing the mechanism by which both the primal and dual solutions can be modified at any given time.

BIOGRAPHICAL SKETCH

Timothy Carnes was born in San Francisco, California in 1983. He received a B.S. in Mathematics from Harvey Mudd College in Claremont, California in 2005. Prior to starting the Ph.D. program in Operations Research at Cornell University in Ithaca, New York, he had lived in California all his life. After completing his Ph.D. he will be beginning work as a postdoctoral fellow at the MIT Sloan School.

This thesis is dedicated to my family.

ACKNOWLEDGEMENTS

I would like to acknowledge and extend my sincere gratitude to my advisor, David Shmoys, for his guidance and advice, and always looking out for my best interest. The research in this document was supported in part by NSF under the grants CCR-0635121, DMS-0732196, CCF-0832782.

TABLE OF CONTENTS

Biographical Sketch	iii
Dedication	iv
Acknowledgements	v
Table of Contents	vi
List of Tables	vii
List of Figures	viii
1 Introduction	1
1.1 Approximation Algorithms	1
1.2 The Primal-Dual Schema	4
1.3 Summary of Results	9
2 Capacitated Covering Problems	13
2.1 Introduction	13
2.2 Minimum Knapsack	17
2.3 Single-Demand Facility Location	20
2.4 Single-Item Lot-Sizing with Linear Holding Costs	23
2.4.1 Time-Dependent Production Costs	32
2.5 Computational Results	33
3 Location Routing Problem	39
3.1 Introduction	39
3.2 Finding an Optimal Routing	41
3.2.1 Application to Real-World Data	45
3.3 The k -Location Routing Problem	46
3.3.1 Routing with Trees, Fixed-Charge Bases	47
3.3.2 Routing with Trees, k -Bases	56
3.3.3 Getting Tours From Trees with No Approximation Loss	60
3.3.4 Computational Results	63
4 Generalized Assignment Problem	69
4.1 Introduction	69
4.2 A New Flow-Based Formulation for the $\{p_j, \infty\}$ Case	70
4.3 A Primal-Dual Approach for the $\{p_j, \infty\}$ Case	73
4.4 A Running Time Bound for the $\{p, \infty\}$ Case	75
5 Conclusion	77
Bibliography	79

LIST OF TABLES

2.1	Probability distributions used to generate lot-sizing data	34
2.2	Numerical results for adding inequalities to lot-sizing formulation	38

LIST OF FIGURES

2.1	Computational results for the single-item lot-sizing problem . . .	36
3.1	A graphical depiction of the location routing problem demonstrating a valid inequality	49
3.2	A visualization of the primal-dual schema for the location routing problem	52
3.3	Computational results for the location routing problem on Ornge data	64
3.4	Computational results for the location routing problem on uniform, random data	65
3.5	Computational results for the location routing problem on clustered, random data	66
3.6	A comparison of optimal tours and primal-dual tours for the location routing problem	68
4.1	A network flow view of the generalized assignment problem with a cut demonstrating a valid inequality	71

CHAPTER 1

INTRODUCTION

1.1 Approximation Algorithms

Ever since the theory of NP -completeness was developed, those faced with designing an algorithm for an NP -hard problem have been forced to choose between creating an efficient method and finding the optimal solution. To achieve both goals simultaneously would prove that $P = NP$, a statement widely believed to be false, though this remains one of the biggest open questions in the field of computer science. In this thesis, we will focus on designing efficient algorithms, where efficiency is defined in terms of the running time being bounded by a polynomial function of the input length. Even though these methods will not necessarily find the optimal solution, the solutions produced are still provably close to optimal for all input instances, and hence these methods are referred to as approximation algorithms.

Approximation algorithms appeared before NP -completeness was established, though were not necessarily presented in such terms. The work of Vizing [34] is one example, where he gave a constructive proof that every graph has an edge coloring where the number of colors used is at most one more than the maximum vertex degree. The idea of developing approximation algorithms as an approach to dealing with NP -hard problems was first proposed in a paper of Johnson [16], where he also provided results for a number of classic problems.

We will formally define a ρ -approximation algorithm as an algorithm that always runs in polynomial time and produces a solution whose value is within a

factor of ρ of the value of an optimal solution. In designing such an algorithm, a key issue is how to show a solution is provably close to optimal when the optimal value is unknown. One approach is to make combinatorial arguments on the structure that an optimal solution must have. Consider the unweighted vertex cover problem, where given a graph one must select a subset of vertices such that every edge has at least one of its endpoints selected, and the goal is to minimize the number of vertices selected. A simple approximation algorithm for this problem is to start with no vertices selected, and as long as there is an uncovered edge, add both of its endpoints to the set of vertices. The set of edges considered by the algorithm must have disjoint endpoints, and so any feasible solution, including the optimal one, must select at least one endpoint from each of these edges. Since the algorithm selects both endpoints on each of these edges, the total number of vertices selected is at most twice the optimal number. Thus the proposed algorithm is a 2-approximation algorithm. While this algorithm is simple to state and to analyze, it has the following disadvantage. If we run the algorithm on any specific problem instance, the above proof will always result in showing that the optimal solution must have chosen at least half as many vertices as we did. Even if we achieve a solution for a given instance that happens to be much closer to optimality, the guarantee we have will still be no better than for the worst case.

Another way to resolve the difficulty of not knowing the optimal value, yet still proving that the value of a solution is close to it, is to obtain a good bound. For a minimization problem, we would want to find a good lower bound on the optimal value, and for a maximization problem we would want a good upper bound, though for this thesis we will consider only minimization problems. This is where the theory of linear programming can be applied quite success-

fully. Nearly any combinatorial optimization problem of interest can be modeled as an integer linear program, and usually in a rather straightforward way. If we wanted to find the exact solution, we could solve the IP, but this cannot generally be done in polynomial time. Instead we can expand the set of feasible solutions by removing the integrality constraints, thereby forming the LP-relaxation of the IP. The optimal solution to the LP-relaxation will have a value that is a lower bound on the value of an optimal solution to the original problem. Furthermore we can solve the LP-relaxation in polynomial time, provided that it is of polynomial size. In fact, even if there are an exponential number of constraints in the LP-relaxation, we can still solve it in polynomial time provided that we can efficiently find a violated constraint when given an infeasible solution.

We now have a general approach for producing an approximation algorithm: formulate an IP, solve the LP-relaxation and produce an integer solution with value no more than a factor of ρ of the value of the optimal fractional solution found. How to find such an integer solution is not a straightforward question. For many problems, it is possible to use the optimal solution to the LP-relaxation to produce a feasible integer solution while gaining at most a factor of ρ in the cost. Such approaches are known as LP-rounding approximation algorithms. Note that if we were to apply an LP-rounding algorithm to a specific problem instance, then we achieve an approximation factor specific to that instance by comparing the integer solution to the optimal solution of the LP-relaxation. There is an inherent limitation in LP-based methods that has to do with how closely the LP-relaxation captures the original IP. The *integrality gap* is the worst-case ratio of the optimal integer solution value to the optimal LP-relaxation solution value over all problem instances. A purely LP-based al-

gorithm cannot hope to achieve an approximation ratio that is any better than the integrality gap. When dealing with a formulation that has a bad integrality gap, one approach is to add extra *valid* inequalities to the program. A valid inequality is simply a constraint that all integer solutions satisfy, but not necessarily all fractional solutions. Adding these valid inequalities does not affect the feasible region of the IP, but can dramatically decrease the feasible region of the LP-relaxation, and hence may greatly reduce the integrality gap.

1.2 The Primal-Dual Schema

Whereas LP-rounding approximation algorithms have many advantages, one disadvantage is that they require solving an LP. While LPs can be solved in polynomial time, it typically takes much longer to do so than to run a purely combinatorial algorithm. It may be possible to more quickly find a good feasible solution to the LP-relaxation through combinatorial methods, but rounding this solution would be of no use since we can only guarantee that the optimal solution to the LP-relaxation provides a lower bound on the optimal value of the original problem. However, if we were to take the dual of the LP-relaxation, then by the weak duality theorem we know that any feasible solution to the dual program provides a lower bound on the optimal value of the LP-relaxation. Thus, if we can produce a feasible integer primal solution with cost at most ρ times the cost of a feasible dual solution, then we will have obtained an LP-based ρ -approximation algorithm. This is the central idea behind the primal-dual schema for approximation algorithms, which has the advantages of an LP-based method, but obviates the need to actually solve an LP.

It is somewhat ironic that the original motivation for the primal-dual method was to provide an alternative means of solving LPs. The origin of the method is in the “Hungarian Method” Kuhn introduced for solving the assignment problem [17]. Dantzig, Ford and Fulkerson [10] generalized Kuhn’s ideas in order to apply them to the problem of solving an LP. The central idea in both cases revolves around trying to satisfy conditions known as *complementary slackness*. To illustrate these conditions, consider the following primal LP

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax \geq b \\ & x \geq 0 \end{aligned}$$

and its dual LP

$$\begin{aligned} \max \quad & b^T y \\ \text{s.t.} \quad & A^T y \leq c \\ & y \geq 0, \end{aligned}$$

where $A \in \mathbb{Q}^{m \times n}$, $c, x \in \mathbb{Q}^n$, $b, y \in \mathbb{Q}^m$, and T denotes the transpose function. Let A_i denote the i th row of A and A^j the j th column of A . We say that solutions x and y satisfy primal complementary slackness conditions if

$$(c_j/\alpha - A^j y)x_j \leq 0 \tag{1.1}$$

holds with $\alpha = 1$ for each $j \in \{1, \dots, n\}$. Note that if x and y are feasible this implies that if $x_j > 0$ then $c_j/\alpha \leq A^j y \leq c_j$. Furthermore, solutions x and y satisfy dual complementary slackness conditions if

$$(A_i x - \beta b_i)y_i \leq 0 \tag{1.2}$$

holds with $\beta = 1$ for all $i \in \{1, \dots, m\}$. Again we have that if x and y are feasible then the above condition ensures that if $y_i > 0$ then $b_i \leq A_i x \leq \beta b_i$.

From the theory of linear programming we know that a pair of feasible primal and dual solutions are both optimal if and only if both primal and dual complementary slackness conditions are satisfied. The idea behind the primal-dual method of Dantzig, Ford and Fulkerson for solving LPs is to begin with a feasible dual solution. Such a solution is typically easy to find, as we can just set $y = 0$ provided that $c \geq 0$, which is true of many optimization problems including all those considered in this thesis. Now that we have a feasible dual solution y , we can see if there is a feasible primal solution x that satisfies the complementary slackness conditions. This can be done by formulating another LP called the *restricted primal* that finds a feasible x which minimizes the amount by which complementary slackness is violated. Either a solution x is found which satisfies complementary slackness in which case we have solved our problem, or else some positive multiple of the optimal solution to the dual of the restricted primal can be added to the current dual solution that will improve the objective function value without violating dual feasibility. In essence we have reduced the problem of solving an LP to that of solving a different LP. However, all is not lost, as the restricted primal can be formulated without including the vector c at all, which effectively reduces a weighted problem to an unweighted one. Additionally, this simpler restricted primal LP can often be solved with purely combinatorial methods.

The primal-dual schema for approximation algorithms is effectively the same as the original method of Dantzig, Ford and Fulkerson. In this case we are working with an LP-relaxation of an IP, and wish to ultimately find a near-optimal integer solution. Unless there is an optimal solution to the LP-relaxation that is also integer, we have no hope of finding a feasible integer primal solution that satisfies the complementary slackness conditions. The solution to this

problem is simply to relax these conditions by allowing $\alpha \geq 1$ and $\beta \geq 1$. We again begin with a feasible dual solution, such as $y = 0$, and now at each iteration we want to determine if there is a feasible integer primal solution that satisfies a relaxed version of complementary slackness. We could set up a restricted primal IP, but since it cannot be solved in polynomial time we must rely on combinatorial methods. A typical approach involves maintaining a primal integer solution that satisfies the primal complementary slackness conditions, but that is infeasible. As the feasible dual solution is improved and more dual constraints become tight, the primal solution is augmented until at some stage it becomes feasible. If the dual complementary slackness conditions were also satisfied, this solution would be optimal. Instead, usually we must show that the relaxed dual complementary slackness conditions are satisfied with some $\beta > 1$. In some cases, to achieve this requires further augmenting the primal solution by zeroing out primal variables that are not needed to maintain primal feasibility. A *reverse delete step* involves considering the primal variables in the reverse order to which they were increased as the algorithm progressed and checking if the solution remains feasible if the variable is set to zero. Once we are able to satisfy the relaxed complementary slackness conditions then we know that the solution produced is within a factor of $\alpha\beta$ of optimal.

Lemma 1. *If x and y are feasible solutions to the primal and dual LPs, and also satisfy (1.1) and (1.2) for some choice of $\alpha, \beta \geq 1$ then*

$$c^T x \leq \alpha\beta(b^T y),$$

and thus x is within a factor of $\alpha\beta$ of optimal.

Proof.

$$c^T x \leq \alpha(A^T y)^T x = \alpha y^T (Ax) \leq \alpha\beta(b^T y),$$

where the inequalities follow directly from conditions (1.1) and (1.2). \square

Many of the polynomial-time optimization algorithms for classic problems in combinatorial optimization can be understood in terms of the primal-dual method. Examples include Kruskal's algorithm for finding the minimum spanning tree, Dijkstra's shortest path algorithm, Ford and Fulkerson's network flow problem as well as the original source of inspiration for the primal-dual method: Kuhn's "Hungarian Method." The first uses of the primal-dual schema for approximation algorithms were developed by Bar-Yehuda & Even and Chvátal for the weighted vertex cover and set cover problems, respectively [4, 9]. Subsequently, this approach has been applied to many other combinatorial problems, such as results of Agrawal, Klein & Ravi [3], Goemans & Williamson [13], Bertsimas & Teo [6] and Levi, Roundy & Shmoys [25] for other related covering problems. Other recent work has been done on covering problems with capacity constraints by Even, Levi, Rawitz, Schieber, Shahar & Sviridenko [11], Chuzhoy & Naor [8] and Gandhi, Halperin, Khuller, Kortsarz & Srinivasan [12].

We say that a primal-dual method is *dual-ascent* provided that throughout the course of the algorithm, the dual variables are never decreased. Then a *simple dual-ascent primal-dual schema* is a dual-ascent primal-dual method where the difference between one feasible dual solution and the next improved one is always the result of increasing a single dual variable. In other words, the algorithm works by increasing one dual variable at a time and is able to produce a feasible integer primal solution that is within a factor of ρ of the resulting dual solution. Although such primal-dual methods result in approximation algorithms for a variety of problems, they are often thought of as being somewhat limited. Indeed, their typical behavior closely resembles a greedy algorithm, as

increasing a single dual variable is akin to finding the cheapest way to satisfy a violated primal constraint.

1.3 Summary of Results

In this thesis, we aim to demonstrate the potential of the simple dual-ascent primal-dual schema, as all of the approximation algorithms proposed fall into this class. The key to the success and applicability of this approach is having strong, valid inequalities to bolster the LP-relaxation. In effect, the inequalities introduced to the LP-relaxation provide the mechanism by which the simple primal-dual schema is able to manipulate the way in which the primal solution is modified until attaining feasibility.

In Chapter 2 we provide results for three capacitated covering problems, with each problem actually being a generalization of the preceding ones. All three results are presented in an effort to demonstrate the simplicity of the approach, as well as its extensibility. The first of these problems, as well as the simplest, is the minimum knapsack problem. Here we are given a set of items, each with a weight and a value. The objective is to find a minimum-weight subset of items such that the total value of the items in the subset meets some specified demand. Carr, Fleischer, Leung & Phillips [7] provided an LP-rounding result for this problem which is a 2-approximation algorithm. However, this involves solving an LP with an exponential number of constraints. We make use of the same underlying LP formulation and achieve a simple dual-ascent method that is also a 2-approximation algorithm.

The next problem considered is the single-demand facility location problem.

The input specifies a set of facilities, each with an opening cost and a capacity, as well as a per-unit serving cost that must be paid for each unit of demand a facility serves. The goal is to open facilities to completely serve a specified amount of demand, while minimizing the total service and facility opening costs. Once again Carr et al. [7] provide a LP-rounding 2-approximation algorithm that relies on solving an exponentially-sized LP. We are able to extend the primal-dual schema to make use of this LP as well to also provide a 2-approximation algorithm for this problem.

The last of the three capacitated covering problems presented is the single-item lot-sizing problem. Here we consider a finite planning period of consecutive time periods. In each time period there is a specified level of demand, as well as a potential order that can be placed with a given capacity and order opening cost. A feasible solution must open enough orders and order enough product so that the inventory level of each time period is never negative. The inventory for a time period is simply the inventory of the previous time period plus however much is ordered in the current time period, minus the demand for that period. However, a per-unit holding cost is incurred for each unit of inventory held over a given time period. Additionally, we may consider a per-unit production cost for the amount of demand requested in each order. For this problem there was no previous LP-based result. There are LP-based techniques which provided approximation algorithms for variants of this problem, but this formulation is set apart by allowing for time-dependent order capacities. Van Hoesel & Wagelmans [32] do provide a fully polynomial approximation scheme for this problem based on dynamic programming and input data rounding. That is to say, for any $\epsilon > 0$, they can provide a $(1 + \epsilon)$ -approximation algorithm. However, the disadvantage of this result is that the running time can

be quite slow for particular performance guarantees, and is dependent on the length of the encoding of the order capacities. We create a new class of valid inequalities for this problem, and using only this style of inequality for our LP formulation extend our primal-dual schema to provide a 2-approximation algorithm in this setting.

In Chapter 3 we examine the location routing problem, which we studied in connection to an application for the company, Ornge. This company manages a fleet of planes based in various locations, and must transport medical patients around the province of Ontario. Deciding where to base their planes and how to route them to requests can be modeled as the location routing problem, where the input provides a set of depots and a set of requests that are all contained in some metric space. We can think of taking the complete graph over all depots and requests, such that the cost of any edge is simply the metric distance between its endpoints. A route is any tour on this graph that includes exactly one depot. The goal is to open a subset of depots and select a set of routes such that each request is included in a route, and the routes selected only go through open depots. This problem can be formulated either as each depot having a particular opening cost, or as having a global constraint on the number of depots that may be opened, or even a combination of both. The primal-dual algorithm of Goemans & Williamson [13] for network design problems can be applied to this problem when there is no global constraint on the number of depots that may be opened, and results in a 2-approximation algorithm. We provide a modification of their algorithm that is a simple dual-ascent method with a stronger type of performance guarantee, and use it in conjunction with Lagrangian relaxation to yield a 2-approximation algorithm which can handle an upper limit on the number of open depots. In contrast to previous approximation algorithms that

have used Lagrangian relaxation, this result is novel in the sense that no binary search is required, nor is there any loss in the approximation factor.

In Chapter 4 we consider the generalized assignment problem. In this problem we have a set of machines and a set of jobs which must be assigned to the machines. For each possible assignment of job to machine, there is an associated cost and processing time. We refer to the makespan as the maximum over the amount of time each machine would take to process all the jobs assigned to it. In this setting we actually have two objectives: minimizing the makespan, and minimizing the cost of the assignment. Shmoys and Tardos [30] provide an LP-rounding bicriteria approximation result. Given a specified cost, C , and makespan, T , they are able to either provide an assignment with cost at most C and makespan at most $2T$, or else they can prove that no assignment exists that has cost at most C and makespan at most T . This problem has an important special case where each machine may only process a subset of jobs, and for each job a machine may serve, the processing time is dependent only on the job. We are able to provide a simple dual-ascent method for this special case that achieves the same performance guarantee as the LP-rounding result. We are able to show the finiteness of this method, though we have not proved a polynomial-bound on the running time. For another important case where each machine uses the same amount of running time for any job that it is allowed to serve, we are able to show the method takes polynomial time. Our result also implies a dual-ascent approach that solves the classic assignment problem, which has not previously been done. While Kuhn's "Hungarian Method" can very naturally be viewed as a primal-dual method, it cannot be readily expressed as dual-ascent.

CHAPTER 2

CAPACITATED COVERING PROBLEMS

This brings us to another advantage of the primal-dual schema – this time not objectively quantifiable. A combinatorial algorithm is more malleable than an algorithm that requires an LP-solver. Once a basic problem is solved using the primal-dual schema, one can also solve variants and generalizations of the basic problem. From a practical standpoint, a combinatorial algorithm is more useful, since it is easier to adapt it to specific applications and fine tune its performance for specific types of inputs.

Vijay V. Vazirani, Primal-Dual Schema Based Approximation Algorithms

2.1 Introduction

We introduce primal-dual schema based on the LP relaxations devised by Carr, Fleischer, Leung & Phillips [7] for the minimum knapsack problem as well as for the single-demand capacitated facility location problem. Our primal-dual algorithms achieve the same performance guarantees as the LP-rounding algorithms of Carr et al., which rely on applying the ellipsoid algorithm to an exponentially-sized LP. Furthermore, we introduce new knapsack-cover inequalities to strengthen the LP relaxation of the more general capacitated single-item lot-sizing problem; using just these inequalities as the LP relaxation, we obtain a primal-dual algorithm that achieves a performance guarantee of 2.

In developing primal-dual (or any LP-based) approximation algorithms, it is important to have a strong LP formulation for the problem. One class of valid inequalities that has proved useful for a variety of problem are called

flow-cover inequalities. Some of the early flow-cover style inequalities were introduced by Padberg, Van Roy & Wolsey [27] for various fixed charge problems. Another large class of flow-cover inequalities was developed by Aardal, Pochet & Wolsey [1] for the capacitated facility location problem. Carr et al. [7] developed a different style of inequalities called *knapsack-cover inequalities* for simpler capacitated covering problems which we use in developing our primal-dual algorithms.

Levi, Lodi & Sviridenko [24] used a subset of the flow-cover inequalities of Aardal et al. [1] to develop a LP-rounding algorithm for the multiple-item lot-sizing problem with monotone holding costs. Our model is not a special case of theirs, however, since our result allows time-dependent order capacities whereas their result assumes constant order capacities across all periods. Additionally, our result allows for time-dependent, per-unit production costs, and it is not clear if their result can handle this added condition. Following this work and the development of our own knapsack-cover inequalities for lot-sizing problems, Sharma & Williamson [29] demonstrated that the result of Levi et al. [24] has an analogue based on our knapsack-cover inequalities as well. Finally, it is worth noting that Van Hoesel & Wagelmans [32] have a FPTAS for the single-item lot-sizing problem that makes use of dynamic programming and input data rounding. The disadvantage of this result is that the running time of the algorithm can be quite slow for particular performance guarantees, and depends on the length of the encoding of the capacity values. Although the primal-dual algorithm we develop is a 2-approximation algorithm, this is a worst-case bound and we would expect it to perform much better on average in practice. Also this is the first LP-based result for the case of time-dependent capacities.

The three models studied in this chapter are generalizations of one another. That is to say, the minimum knapsack problem is a special case of the single-demand capacitated facility location problem, which is a special case of the single-item lot-sizing problem. We present the result in order of generality with the aim of explaining our approach in the simplest setting first. The *minimum knapsack problem* gives a set of items, each with a weight and a value. The objective is to find a minimum-weight subset of items such that the total value of the items in the subset meets some specified demand. In the *single-demand facility location problem* there is a set of facilities, each with an opening cost and a capacity, as well as a per-unit serving cost that must be paid for each unit of demand a facility serves. The goal is to open facilities to completely serve a specified amount of demand, while minimizing the total service and facility opening costs. Finally the *single-item lot-sizing problem* considers a finite planning period of consecutive time periods. In each time period there is a specified level of demand, as well as a potential order with a given capacity and order opening cost. A feasible solution must open enough orders and order enough product so that in each time period there is enough inventory to satisfy the demand of that period. The inventory is simply the inventory of the previous time period plus however much is ordered in the current time period, minus the demand for that period. However, a per-unit holding cost is incurred for each unit of inventory held over a given time period. Additionally, we may consider a per-unit production cost for the amount of demand requested in each order.

The straightforward LP relaxations for these problems have bad integrality gaps, but can be strengthened by introducing valid knapsack-cover inequalities. The inequalities Carr et al. [7] developed for the minimum knapsack problem

are as follows

$$\sum_{i \in F \setminus A} u_i(A) y_i \geq D - u(A) \quad \forall A \subseteq F,$$

where the y_i are the binary decision variables indicating if item i is chosen, $u(A)$ is the total value of the subset of items A , and the $u_i(A)$ can be thought of as the effective value of item i with respect to A , which is the minimum of the actual value and the right-hand-side of the inequality. These inequalities arise by considering that if we did choose all items in the set A , then we still have an induced subproblem on all of the remaining items, and the values can be truncated since we are only concerned with integer solutions. Our primal-dual algorithm works essentially as a modified greedy algorithm, where at each stage the item is selected that has the largest value per cost. Instead of the actual values and costs, however, we use the effective values and the slacks of the dual constraints as costs. Similar to the greedy algorithm for the traditional maximum-value knapsack problem, the last item selected and everything selected beforehand, can each be bounded in cost by the dual LP value, yielding a 2-approximation.

The study of approximation algorithms is not just to prove theoretical performance guarantees, but also to gain sufficient understanding of the mathematical structure of a problem so as to design algorithms that perform well in practice. We examine the performance of the primal-dual algorithm for the single-item lot-sizing problem in a series of computational experiments. We demonstrate the effectiveness of the algorithm by comparing it to other natural heuristics, some of which make use of the optimal LP solution.

The remainder of this chapter is organized as follows. In Section 2.2 we go over the minimum knapsack result in more detail. In Section 2.3 we generalize this result to apply to the single-demand capacitated facility location problem.

In Section 2.4 we generalize the knapsack-cover inequalities to handle the lot-sizing problem, and then present and analyze a primal-dual algorithm for the single-item lot-sizing problem. Finally, in Section 2.5 we present computational results analyzing the performance of the primal-dual algorithm for lot-sizing compared to other heuristics, as well as exploring the effectiveness of utilizing the algorithm to create a better formulation.

2.2 Minimum Knapsack

In the *minimum knapsack problem* one is given a set of items F , and each item $i \in F$ has a value u_i and a weight f_i . The goal is to select a minimum weight subset of items, $S \subseteq F$, such that the value of S , $u(S)$, is at least as big as a specified demand, D . The natural IP formulation for this problem is

$$\begin{aligned} \text{opt}_{MK} &:= \min \sum_{i \in F} f_i y_i && \text{(MK-IP)} \\ \text{s.t.} & \sum_{i \in F} u_i y_i \geq D && (2.1) \\ & y_i \in \{0, 1\} && \forall i \in F, \end{aligned}$$

where the y_i variables indicate if item i is chosen. The following example from [7] demonstrates that the integrality gap between this IP and the LP relaxation is at least as bad as D . Consider just 2 items where $u_1 = D - 1$, $f_1 = 0$, $u_2 = D$ and $f_2 = 1$. The only feasible integer solution chooses both items and has a cost of 1, whereas the LP solution can set $y_1 = 1$ and $y_2 = 1/D$ and incurs a cost of only $1/D$. To remedy this situation we consider using the knapsack-cover inequalities introduced in [7].

The idea is to consider a subset of items $A \subseteq F$ such that $u(A) < D$, and let

$D(A) = D - u(A)$. This means that even if all of the items in the set A are chosen, we must choose enough items in $F \setminus A$ such that the demand $D(A)$ is met. This is just another minimum knapsack problem where the items are restricted to $F \setminus A$ and the demand is now $D(A)$. The value of every item can be restricted to be no greater than the demand without changing the set of feasible integer solutions, so let $u_i(A) = \min\{u_i, D(A)\}$. This motivates the following LP

$$\begin{aligned} \text{opt}_{MKP} &:= \min \sum_{i \in F} f_i y_i && \text{(MK-P)} \\ \text{s.t.} \quad &\sum_{i \in F \setminus A} u_i(A) y_i \geq D(A) && \forall A \subseteq F \quad (2.2) \\ &y_i \geq 0 && \forall i \in F, \end{aligned}$$

and by the validity of the knapsack-cover inequalities argued above we have that every feasible integer solution to (MK-IP) is a feasible solution to (MK-P). The dual of this LP is

$$\begin{aligned} \text{opt}_{MKD} &:= \max \sum_{A \subseteq F} D(A) v(A) && \text{(MK-D)} \\ \text{s.t.} \quad &\sum_{A \subseteq F: i \notin A} u_i(A) v(A) \leq f_i && \forall i \in F \quad (2.3) \\ &v(A) \geq 0 && \forall A \subseteq F. \end{aligned}$$

Our primal-dual algorithm begins by initializing all of the primal and dual variables to zero, which produces a feasible dual solution and an infeasible primal integer solution. Taking our initial subset of items, A , to be the empty set, we increase the dual variable $v(A)$. Once a dual constraint becomes tight, the item corresponding to that constraint is added to the set A , and we now increase the new variable $v(A)$. Note that increasing $v(A)$ does not increase the left-hand-sides of dual constraints corresponding to items in A , so dual feasibility will not

be violated. This process is repeated as long as $D(A) > 0$, and once we finish we call our final set of items S , which is our integer solution. This is a feasible solution to (MK-IP) since $D(S) \leq 0$, which implies $u(S) \geq D$.

Algorithm 1: Primal-Dual for Minimum Knapsack

```

 $y, v \leftarrow 0$ 
 $A \leftarrow \emptyset$ 
while  $D(A) > 0$  do
    Increase  $v(A)$  until a dual constraint becomes tight for item  $i$ 
     $y_i \leftarrow 1$ 
     $A \leftarrow A \cup \{i\}$ 
 $S \leftarrow A$ 

```

Theorem 1. *Algorithm 1 terminates with a solution of cost no greater than $2 \cdot \text{opt}_{MK}$.*

Proof. Let ℓ denote the final item selected by Algorithm 1. Then because the algorithm only continues running as long as $D(A) > 0$ we have that

$$D(S \setminus \{\ell\}) > 0 \Rightarrow D - u(S \setminus \{\ell\}) > 0 \Rightarrow u(S \setminus \{\ell\}) < D.$$

Also, the variable $v(A)$ is positive only if $A \subseteq S \setminus \{\ell\}$, which means

$$\sum_{i \in F \setminus A} u_i(A) y_i = u(A \setminus \{\ell\}) - u(A) + u_\ell(A) \leq 2D(A),$$

where the inequality follows from the definition of $u_\ell(A)$ and of $D(A)$. This implies that the solutions satisfy the relaxed dual complementary slackness conditions (1.2) with $\beta = 2$. We satisfy primal complementary slackness conditions (1.1) (with $\alpha = 1$) since the algorithm only selects items for which the constraint (2.3) has become tight. Thus by Lemma 1 we have a 2-approximation algorithm. □

2.3 Single-Demand Facility Location

In the *single-demand facility location problem*, one is given a set of facilities F , where each facility $i \in F$ has capacity u_i , opening cost f_i , and there is a per-unit cost c_i to serve the demand, which requires D units of the commodity. The goal is to select a subset of facilities to open, $S \subseteq F$, such that the combined cost of opening the facilities and serving the demand is minimized. The natural IP formulation for this problem is

$$\text{opt}_{FL} := \min \sum_{i \in F} (f_i y_i + D c_i x_i) \quad (\text{FL-IP})$$

$$\text{s.t.} \sum_{i \in F} x_i = 1 \quad (2.4)$$

$$u_i y_i \geq D x_i \quad \forall i \in F \quad (2.5)$$

$$y_i \geq x_i \quad \forall i \in F \quad (2.6)$$

$$y_i \in \{0, 1\} \quad \forall i \in F$$

$$x_i \geq 0 \quad \forall i \in F,$$

where each y_i indicates if facility $i \in F$ is open and each x_i indicates the fraction of D being served by facility $i \in F$. The same example from the minimum knapsack problem also demonstrates the large integrality gap of this IP. We once again turn to the knapsack-cover inequalities introduced by Carr et al. [7].

For these inequalities, we once again consider a subset of facilities $A \subseteq F$ such that $u(A) < D$, and let $D(A) = D - u(A)$. This means that even if all of the facilities in the set A are opened, we must open enough facilities in $F \setminus A$ such that we will be able to assign the remaining demand $D(A)$. But certainly for any feasible integer solution, a facility $i \in F \setminus A$ cannot contribute more than $\min\{D x_i, u_i(A) y_i\}$ towards the demand $D(A)$. So if we partition the remaining

orders of $F \setminus A$ into two sets F_1 and F_2 , then for each $i \in F_1$ we will consider its contribution as Dx_i , and for each $i \in F_2$ we will consider its contribution as $u_i(A)y_i$. The total contribution of these facilities must be at least $D(A)$, so if we let \mathcal{F} be the set of all 3-tuples that partition F into three sets, we obtain the following LP

$$\begin{aligned} \text{opt}_{FLP} &:= \min \sum_{i \in F} (f_i y_i + D c_i x_i) && \text{(FL-P)} \\ \text{s.t.} & \sum_{i \in F_1} D x_i + \sum_{i \in F_2} u_i(A) y_i \geq D(A) && \forall (F_1, F_2, A) \in \mathcal{F} \quad (2.7) \\ & x_i, y_i \geq 0 && \forall i \in F, \end{aligned}$$

and by the validity of the knapsack-cover inequalities argued above we have that every feasible integer solution to (FL-IP) is a feasible solution to (FL-P). The dual of this LP is

$$\begin{aligned} \text{opt}_{FLD} &:= \max \sum_{(F_1, F_2, A) \in \mathcal{F}} D(A) v(F_1, F_2, A) && \text{(FL-D)} \\ \text{s.t.} & \sum_{(F_1, F_2, A) \in \mathcal{F}: i \in F_1} D v(F_1, F_2, A) \leq D c_i && \forall i \in F \quad (2.8) \\ & \sum_{(F_1, F_2, A) \in \mathcal{F}: i \in F_2} u_i(A) v(F_1, F_2, A) \leq f_i && \forall i \in F \quad (2.9) \\ & v(F_1, F_2, A) \geq 0 && \forall (F_1, F_2, A) \in \mathcal{F}. \end{aligned}$$

As in Section 2.2 the primal-dual algorithm begins with all variables at zero and an empty subset of facilities, A . Before a facility is added to A , we will require that it become tight on both types of dual constraints. To achieve this we will leave each facility in F_1 until it becomes tight on constraint (2.8), move it into F_2 until it is also tight on constraint (2.9), and only then move it into A . As before the algorithm terminates once the set A has enough capacity to satisfy the demand, at which point we label our final solution S .

Algorithm 2: Primal-Dual for Single-Demand Facility Location

```

 $x, y, v \leftarrow 0$ 
 $F_1 \leftarrow F$ 
 $F_2, A \leftarrow \emptyset$ 
while  $D(A) > 0$  do
  Increase  $v(F_1, F_2, A)$  until a dual constraint becomes tight for facility  $i$ 
  if  $i \in F_1$  then /*  $i$  tight on (2.8) but not on (2.9) */
    | Move  $i$  from  $F_1$  into  $F_2$ 
  else /* else  $i$  tight on (2.8) and (2.9) */
    |  $x_i \leftarrow u_i(A)/D$ 
    |  $y_i \leftarrow 1$ 
    | Move  $i$  from  $F_2$  into  $A$ 
   $S \leftarrow A$ 

```

Clearly Algorithm 2 terminates with a feasible solution to (FL-IP) since all of the demand is assigned to facilities that are fully opened.

Theorem 2. *Algorithm 2 terminates with a solution of cost no greater than $2 \cdot \text{opt}_{FL}$.*

Proof. Let ℓ denote the final facility selected by Algorithm 2. By the same reasoning as in Section 2.2 we have

$$D(S \setminus \{\ell\}) > 0 \Rightarrow D - u(S \setminus \{\ell\}) > 0 \Rightarrow u(S \setminus \{\ell\}) < D.$$

The variable $v(F_1, A)$ is positive only if $A \subseteq S \setminus \{\ell\}$. Recall that at the last step of Algorithm 2, facility ℓ was assigned $D(S \setminus \{\ell\})$ amount of demand. Since $D(A)$ only gets smaller as the algorithm progresses, we have that regardless of what summation above the facility ℓ is in, it contributes no more than $D(A)$. All of the other terms can be upper bounded by the actual capacities and hence

$$\sum_{i \in S \cap F_2} u_i(A) + \sum_{i \in S \cap F_1} Dx_i = u(S \setminus \{\ell\}) - u(A) + u_\ell(A) \leq 2D(A),$$

where the inequality follows from the definition of $u_\ell(A)$ and of $D(A)$. This implies that the solutions satisfy the relaxed dual complementary slackness con-

ditions (1.2) with $\beta = 2$. We satisfy primal complementary slackness conditions (1.1) (with $\alpha = 1$) since the algorithm only selects facilities that are tight on both constraints (2.8) and (2.9). So by Lemma 1 we have a 2-approximation algorithm in this setting as well. \square

2.4 Single-Item Lot-Sizing with Linear Holding Costs

In the single-item lot-sizing problem, one is given a planning period consisting of time periods $F := \{1, \dots, T\}$. For each time period $t \in F$, there is a demand, d_t , and a potential order with capacity u_t , which costs f_t to place, regardless of the amount of product ordered. At each period, the total amount of product left over from the previous period plus the amount of product ordered during this period must be enough to satisfy the demand of this period. Any remaining product is held over to the next period, but incurs a cost of h_t per unit of product stored. Also, we may consider a per-unit production cost associated with the amount of demand requested for a given order, but for now we will ignore this possibility. If we let

$$h_{st} = \sum_{r=s}^{t-1} h_r$$

and set $h_{tt} = 0$ for all $t \in F$, then we obtain a standard IP formulation for this problem as follows

$$\text{opt}_{LS} := \min \sum_{s=1}^T f_s y_s + \sum_{s=1}^T \sum_{t=s}^T h_{st} d_t x_{st} \quad (\text{LS-IP})$$

$$\text{s.t.} \sum_{s=1}^t x_{st} = 1 \quad \forall t \quad (2.10)$$

$$\sum_{t=s}^T d_t x_{st} \leq u_s y_s \quad \forall s \quad (2.11)$$

$$x_{st} \leq y_s \quad \forall s \leq t \quad (2.12)$$

$$y_s \in \{0, 1\} \quad \forall s$$

$$x_{st} \geq 0 \quad \forall s \leq t.$$

where the y_s variables indicate if an order has been placed at time period s , and the x_{st} variables indicate what fraction of the demand d_t is being satisfied from product ordered during time period s . This formulation once again suffers from a bad integrality gap, which can be demonstrated by the same example as in the previous two sections. We introduce new knapsack-cover inequalities to strengthen this formulation.

The basic idea is similar to the inequalities used in sections 2.2 and 2.3. We would like to consider a subset of orders, A , where even if we place all the orders in A and use these orders to their full potential, there is still unmet demand. In the previous cases, the amount of unmet demand was $D(A) = D - u(A)$. Now, however, that is not quite true, since each order s is capable of serving only the demand points t where $t \geq s$. Instead, we now also consider a subset of demand points B , and define $d(A, B)$ to be the total unmet demand in B , when

the orders in A serve as much of the demand in B as possible. More formally

$$d(A, B) := \min d(B) - \sum_{s \in A} \sum_{t \geq s: t \in B} d_t x_{st} \quad (\text{RHS-LP})$$

$$\text{s.t. } \sum_{s=1}^t x_{st} \leq 1 \quad \forall t \quad (2.13)$$

$$\sum_{t=s}^T d_t x_{st} \leq u_s \quad \forall s \quad (2.14)$$

$$x_{st} \geq 0 \quad \forall s \leq t.$$

As before, we would also like to restrict the capacities of the orders not in A . To do this, we define

$$u_s(A, B) := d(A, B) - d(A \cup \{s\}, B), \quad (2.15)$$

which is the decrease in remaining demand that would result if order s were added to A . (This reduces to the same $u_s(A)$ as defined in the previous sections when considered in the framework of the earlier problems.) We once again partition the remaining orders in $F \setminus A$ into two sets, F_1 and F_2 , and count the contribution of orders in F_1 as $\sum_t d_t x_{st}$ and orders in F_2 as $u_s(A, B)y_s$. This leads to the following LP, where once again \mathcal{F} is the set of all 3-tuples that partition F into three sets.

$$\text{opt}_{LSP} := \min \sum_{s=1}^T f_s y_s + \sum_{s=1}^T \sum_{t=s}^T h_{st} d_t x_{st} \quad (\text{LS-P})$$

$$\text{s.t. } \sum_{\substack{s \in F_1, \\ t \in B}} d_t x_{st} + \sum_{s \in F_2} u_s(A, B)y_s \geq d(A, B) \quad (2.16)$$

$$\forall (F_1, F_2, A) \in \mathcal{F}, B \subseteq F$$

$$x_{st}, y_s \geq 0 \quad \forall s, t.$$

Lemma 2. *Any feasible solution to (LS-IP) is a feasible solution to (LS-P).*

Proof. Consider a feasible integer solution (x, y) to (LS-IP) and let $S := \{s : y_s = 1\}$. Now for any $(F_1, F_2, A) \in \mathcal{F}$ and $B \subseteq F$ we know

$$\sum_{\substack{s \in F_1, \\ t \in B}} d_t x_{st} \geq d((F_2 \cap S) \cup A, B),$$

since there is no way to assign demand from B to orders in $(F_2 \cap S) \cup A$ without leaving at least $d((F_2 \cap S) \cup A, B)$ amount of demand unfulfilled. Thus at least that amount of demand must be served by the other orders in S , namely those in F_1 . Let $k := |F_2 \cap S|$ and let s_1, \dots, s_k denote the elements of that set in some order. Furthermore let $S_i := \{s_1, \dots, s_i\}$ for each $1 \leq i \leq k$, so $S_k = F_2 \cap S$. Then by repeated use of (2.15) we have

$$\begin{aligned} \sum_{\substack{s \in F_1, \\ t \in B}} d_t x_{st} &\geq d((F_2 \cap S) \cup A, B) \\ &= d(((F_2 \cap S) \cup A) \setminus S_1, B) - u_{s_1}(((F_2 \cap S) \cup A) \setminus S_1, B) \\ &= d(((F_2 \cap S) \cup A) \setminus S_k, B) - \sum_{i=1}^k u_{s_i}(((F_2 \cap S) \cup A) \setminus S_i, B) \\ &= d(A, B) - \sum_{i=1}^k u_{s_i}(((F_2 \cap S) \cup A) \setminus S_i, B) \\ &\geq d(A, B) - \sum_{s \in F_2 \cap S} u_s(A, B) \\ &\geq d(A, B) - \sum_{s \in F_2} u_s(A, B) y_s, \end{aligned}$$

where the inequalities follow since $u_s(A, B)$ is increasing as elements are removed from A . Thus (x, y) satisfies all of the knapsack-cover inequalities (2.16). \square

The dual of (LS-P) is

$$\text{opt}_{LSD} := \max \sum_{(F_1, F_2, A) \in \mathcal{F}} d(A, B)v(F_1, F_2, A, B) \quad (\text{LS-D})$$

$$\text{s.t.} \quad \sum_{\substack{(F_1, F_2, A) \in \mathcal{F}, B \subseteq F: \\ s \in F_1, t \in B}} v(F_1, F_2, A, B) \leq h_{st} \quad \forall s \leq t \quad (2.17)$$

$$\sum_{\substack{(F_1, F_2, A) \in \mathcal{F}, B \subseteq F: \\ s \in F_2}} u_s(A, B)v(F_1, F_2, A, B) \leq f_s \quad \forall s \quad (2.18)$$

$$v(F_1, F_2, A, B) \geq 0 \quad \forall (F_1, F_2, A) \in \mathcal{F}, B \subseteq F,$$

where we simply divided constraint (2.17) by d_t .

Before we get to a primal-dual algorithm, we must first introduce some notation and associated machinery.

$$e_t := d_t \left[1 - \sum_{r=1}^t x_{rt} \right] \quad \text{- amount of demand currently unsatisfied in period } t$$

We define $\text{Fill}(A, B)$ to be the following procedure that describes how to assign demand from B to orders in A . We consider the orders in A in arbitrary order, and for each order we serve as much demand as possible, processing demands from earliest to latest.

In the previous two sections, there was effectively only one demand, and so we never had to be concerned about how demand is assigned once an item or facility is chosen. Now there are many demand points, and so we must be careful that as we maintain our solution set A , we are serving as much demand as possible from the orders in A . The way the primal-dual algorithm will assign demand will correspond with how the Fill procedure works, and we show that this is a maximal assignment.

Lemma 3. *If we start from an empty demand assignment and run $\text{Fill}(A, B)$, then*

we obtain a demand assignment such that $e(B) = d(A, B)$. Thus, *Fill* produces an assignment that is optimal for (RHS-LP).

Proof. Consider the latest time period $t \in B$ where $e(t) > 0$. All orders at time periods at or before t must be serving up to capacity, since otherwise they could have served more of demand d_t . All orders after time period t could not have served any more demand in B since all demand points in B after t are fully served. □

Just as in the previous two sections, the primal-dual algorithm initializes the variables to zero and the set A to the empty set. As in Section 2.3, we initialize F_1 to be the set of all orders, and an order will become tight first on constraint (2.17), when it will be moved to F_2 , and then tight on (2.18), when it will be moved to A . Unlike in Section 2.3, however, constraint (2.17) consists of many different inequalities for the same order. This difficulty is averted since all the constraints (2.17) for a particular order will become tight at the same time, as is proved below in Lemma 4. This is achieved by slowly introducing demand points into the set B . Initially, B will consist only of the last demand point, T . Every time an order becomes tight on all constraints (2.17), it is moved from F_1 into F_2 , and the demand point of that time period is added to B . In this way we always maintain that F_1 is a prefix of F , and B is the complementary suffix. When an order s becomes tight on constraint (2.18), we move it to A and assign demand to it by running the procedure $\text{Fill}(s, B)$. Additionally we create a *reserve set* of orders, R_s , for order s , that consists of all orders earlier than s that are not in F_1 at the time s was added to A . Finally, once all of the demand has been assigned to orders, we label the set of orders in A as our current solution, S^* , and now enter a clean-up phase. We consider the orders in the reverse order

in which they were added to A , and for each order, s , we check to see if there is enough remaining capacity of the orders that are in both the reserve set and our current solution, $S^* \cap R_s$, to take on the demand being served by s . If there is, then we reassign that demand to the orders in $S^* \cap R_s$ arbitrarily and remove s from our solution S^* . When the clean-up phase is finished we label the nodes in S^* as our final solution, S .

Algorithm 3: Primal-Dual for Single-Item Lot-Sizing

```

 $x_{st}, y_s \leftarrow 0$ 
 $F_1 \leftarrow F$ 
 $F_2, A \leftarrow \emptyset$ 
 $B \leftarrow \{T\}$ 
while  $d(A, F) > 0$  do
    Increase  $v(F_1, F_2, A, B)$  until dual constraint becomes tight for order  $s$ 
    if  $s \in F_1$  then /*  $s$  tight on (2.17) but not on (2.18) */
        Move  $s$  from  $F_1$  into  $F_2$ 
         $B \leftarrow F \setminus F_1$ 
    else /* else  $s$  tight on (2.17) and (2.18) */
         $y_s \leftarrow 1$ 
        Fill( $s, B$ )
        Move  $s$  from  $F_2$  into  $A$ 
         $R_s \leftarrow \{r \in F \setminus F_1 : r < s\}$ 
 $S^* \leftarrow A$  /* start clean-up phase */
for  $s \leftarrow$  last order added to  $A$  to first order added to  $A$  do
    if remaining capacity of orders in  $S^* \cap R_s$  is enough to serve demand of  $s$ 
    then
        Remove  $s$  from solution  $S^*$ 
         $y_s, x_{st} \leftarrow 0$  /* unassign demand of  $s$  */
        Fill( $S^* \cap R_s, F$ ) /* reassign to reserve orders */
 $S \leftarrow S^*$ 

```

Lemma 4. *All of the constraints (2.17) for a particular order become tight at the same time, during the execution of Algorithm 3.*

Proof. We instead prove an equivalent statement: when demand t is added to

B , then for any order $s \leq t$ the slack of the constraint (2.17) corresponding to s and demand t' is h_{st} for any $t' \geq t$. This statement implies the lemma by considering $s = t$, which implies all constraints (2.17) for s become tight at the same time. We prove the above statement by (backwards) induction on the demand points. The case where $t = T$ clearly holds, since this demand point is in B before any dual variable is increased, and hence the slack of constraint (2.17) for order s and demand T is h_{sT} . Now assume the statement holds for some $t \leq T$. If we consider order $s = t - 1$ then by the inductive hypothesis the slack of all constraints (2.17) for s and demand $t' \geq t$ is h_{st} . Hence the slack of all constraints (2.17) for orders $s' \leq s$ decreases by h_{st} between the time t is added to B and when $t - 1$ is added to B . Then by the inductive hypothesis again, we have that for any order $s' \leq s$ and any demand $t' \geq t$, when $t - 1$ is added to B the slack of the corresponding constraint (2.17) is

$$h_{s't} - h_{st} = \sum_{r=s'}^{t-1} h_r - \sum_{r=s}^{t-1} h_r = \sum_{r=s'}^{t-1} h_r - h_{t-1} = \sum_{r=s'}^{t-2} h_r = h_{s',t-1}.$$

Hence the statement also holds for $t - 1$. \square

Define ℓ to be

$$\ell = \ell(F_1, F_2, A) := \max\{s : s \in S \cap F_2\} \cup \{0\},$$

so ℓ is the latest order in the final solution that is also in F_2 , for a given partition, or if there is no such order then ℓ is 0, which is a dummy order with no capacity.

Lemma 5. *Upon completion of Algorithm 3, for any F_1, F_2, A, B such that the corresponding dual variable $v(F_1, F_2, A, B) > 0$, we have*

$$\sum_{s \in S \cap F_2} u_s(A, B) + \sum_{s \in S \cap F_1} \sum_{t \in B: t \geq s} d_t x_{st} < d(A, B) + u_\ell(A, B).$$

Proof. First we consider the case when $S \cap F_2 = \emptyset$. Here the first summation is empty, so we just need to show the bound holds for the second summation. We know that any order $s \in S \cap F_1$ is not in the reserve set for any order in A . This follows since F_1 decreases throughout the course of the algorithm, so since s is in F_1 at this point, then clearly s was in F_1 at any point previously, in particular when any order in A was originally added to A . Hence no demand that was originally assigned to an order in A was ever reassigned to an order in F_1 . But from Lemma 3 we know that only an amount $d(A, B)$ of demand from demand points in B is not assigned to orders in A , thus orders in F_1 can serve at most this amount of demand from demand points in B and we are done.

Otherwise it must be the case that $S \cap F_2 \neq \emptyset$, hence ℓ corresponds to a real order that was not deleted in the clean-up phase. By the way ℓ was chosen we know that any other order in $S \cap F_2$ is in an earlier time period than ℓ , and since these orders were moved out of F_1 before ℓ was added to A , they must be in the reserve set R_ℓ . However, since ℓ is in the final solution, it must be the case that when ℓ was being considered for deletion the orders in the reserve set that were still in the solution did not have sufficient capacity to take on the demand assigned to ℓ . Thus if we let x' denote the demand assignment during the clean-up phase when ℓ was being considered, then

$$\sum_{s \in (S \cap F_2) \setminus \{\ell\}} u_s(A, B) \leq u((S \cap F_2) \setminus \{\ell\}) < \sum_{\substack{s \in S \cap F_2 \\ t \in B: t \geq s}} d_t x'_{st}.$$

None of the orders in A had been deleted when ℓ was being considered for deletion, so only an amount $d(A, B)$ of the demand in B was being served by orders outside of A . As argued previously, none of the orders in F_1 took on demand being served by orders in A , but they also did not take on demand being served by orders in $S \cap F_2$, since these orders were never deleted. Thus

we can upper bound the amount of demand that orders from $S \cap F_2$ could have been serving at the time order ℓ was being considered for deletion as follows

$$\sum_{\substack{s \in S \cap F_2 \\ t \in B: t \geq s}} d_t x'_{st} \leq d(A, B) - \sum_{\substack{s \in S \cap F_1 \\ t \in B: t \geq s}} d_t x_{st}.$$

The desired inequality is obtained by rearranging terms and adding $u_\ell(A, B)$ to both sides. \square

By the definition of $u_\ell(A, B)$, Lemma 5 implies the solutions satisfy relaxed dual complementary slackness conditions (1.2) with $\beta = 2$, just as in the previous sections. We also satisfy primal complementary slackness conditions (1.1) (with $\alpha = 1$) since all of the orders in the solution are tight on all constraints (2.17) and (2.18). Lemma 1 then yields the following theorem.

Theorem 3. *Algorithm 3 terminates with a solution of cost no greater than $2 \cdot \text{opt}_{LS}$.*

2.4.1 Time-Dependent Production Costs

It is worth noting that Algorithm 3 and its corresponding analysis require almost no changes to handle the case of time-dependent, per-unit production costs. The only change this introduces in the dual program (LS-D) is simply the addition of the term p_s to the right-hand-side of constraint (2.17). But this simply adds the same term to all constraints (2.17) for a particular order, so it is straightforward to adjust the algorithm so that Lemma 4 remains true, which is that all constraints (2.17) become tight for an order at the same time. We simply change the way that the set B changes over time, so that we add a demand point t to B precisely when the slack of all constraints (2.17) corresponding to the order at t , are equal to p_t . When $p_t = 0$, this is precisely what the algorithm

did before and at the same time all constraints (2.17) become tight, and if $p_t > 0$ then they will all become tight p_t time units later in the algorithm. The rest of the analysis is the same as before.

2.5 Computational Results

In this section we analyze the performance of Algorithm 3 compared to other reasonable heuristics for the single-item lot-sizing problem, as well as explore how well this algorithm can be used to strengthen the underlying formulation. We generated two different sets of problem data. The first set mirrors the experiments done by Pochet and Wolsey[28] for a nearly identical problem, with the one difference being that we do not require the capacities to be non-decreasing over time. This set of data formed problem instances where capacity was rather constrained, so we also used a second set of instances where capacity was much less-constrained. In both cases, all of the data distributions were uniform, with the ranges shown in Table 2.1. As done by Pochet and Wolsey, to ensure feasibility, we make the added assumption that the demand for any period is no more than the capacity for that period. This assumption is in essence without loss of generality, since any feasible instance can be expressed in this form by the following transformation. For any period where the demand exceeds the capacity, we know that this excess must be served by earlier orders, so we may simply move the excess demand to the previous period, and add the cost of holding this demand over this period to the objective function. We varied the number of time periods from 40 to 1000, and for each value generated ten instances according to the above distributions.

Table 2.1: Ranges for uniform data distributions for the two generated data sets.

Distribution	Demand	Capacity	Order Cost	Holding Cost
(a)	[6, 35]	[16, 25]	[16, 20]	[0.01, 0.05]
(b)	[0, 9]	[16, 38]	[16, 33]	[1, 4]

We now describe the heuristics that were used to compare against the performance of the primal-dual algorithm.

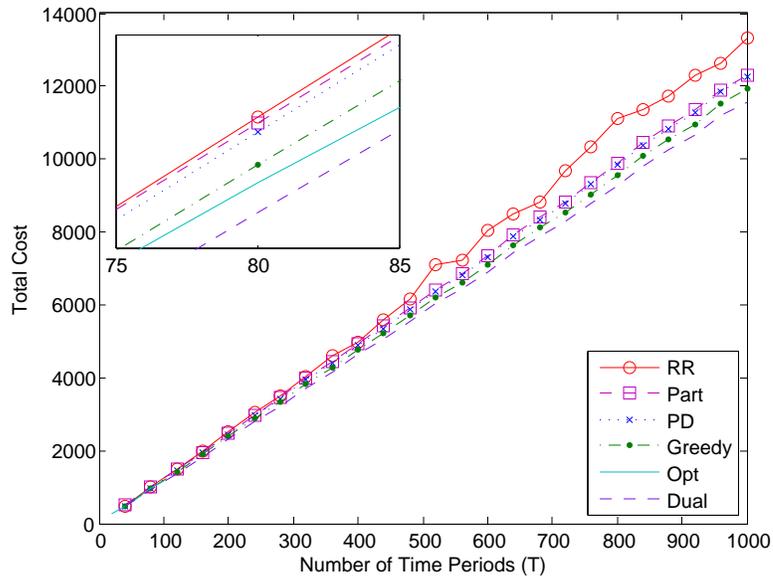
1. *GREEDY*, where orders are selected greedily based upon a notion of cost-effectiveness. At any given time, all orders that have not yet been selected are evaluated, and the one with the cheapest cost per demand served is added to the solution. To calculate this cost, we assume an order processes the remaining unsatisfied demand in order from earliest to latest, just as in the `Fill` procedure of Section 2.4. We then take the holding cost associated with this order serving all of this demand, add it to the ordering cost, and then divide by how much demand would be served. The order which has the lowest value is selected, and the demand it would serve is assigned to it.
2. *RR*, where orders are selected based upon a randomized rounding of the optimal LP solution. Here the LP is first solved, and then each order s is selected to be opened with probability y_s . The resulting ordering cost for this process is the same in expectation as the ordering cost of the LP, but this may lead to an infeasible solution. To fix this, we process the orders in reverse chronological order starting at time period T , and perform the randomized rounding as described above. If an order is added to the solution

then it is assigned all of the demand that the `Fill` procedure would give it. If at any point we require an order to be open in order to be able to serve the remaining demand, then it is opened without any randomization.

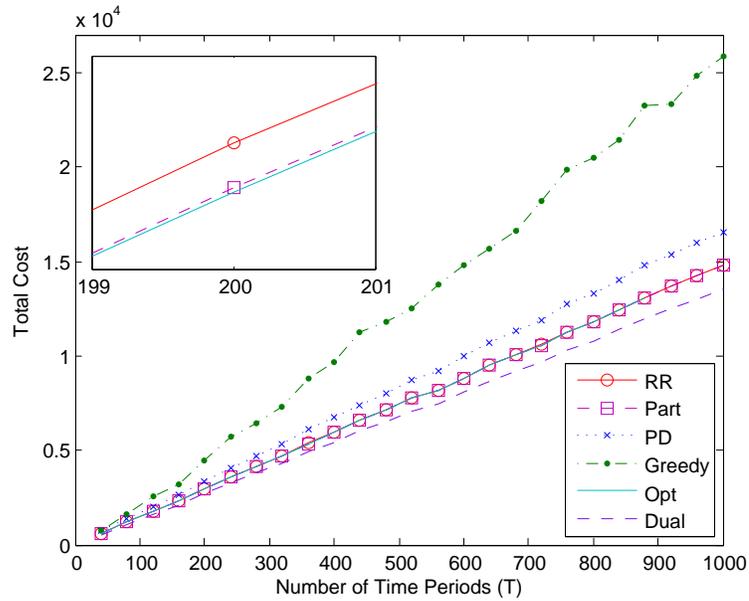
3. *PART*, where the problem is partitioned into many subproblems, and for each subproblem the optimal integer solution is found. The main restriction with finding the optimal integer solution with commercial solvers is that the running time blows up very quickly as the number of time periods increases. Breaking a problem down into more manageable pieces makes finding the optimal integer solution for these parts a feasible option. To this end, first the optimal LP solution is found, and then the amount of inventory at the end of each period is computed. The time period is then broken into intervals, where each interval starts and ends with no inventory in the optimal LP solution. For each of these intervals, the optimal integer solution is found, and we combine all these solutions to produce a solution to the original problem.

The results of running each of the above heuristics along with the primal-dual algorithm on both sets of generated problem instances are summarized in Figure 2.1. In addition a lower bound on the value of the optimal integer solution is shown which is simply the value of the dual solution generated by the primal-dual algorithm. Computing the actual value of the optimal integer solution would have been computationally intractable for how large we took the number of time periods to be.

We see from Figure 2.1(a) that surprisingly the *GREEDY* heuristic performed the best of all, though the primal-dual algorithm was fairly close. The good performance of *GREEDY* may be attributed to the way the data for these problems



(a)



(b)

Figure 2.1: The results of running the heuristics on all of the generated problem instances, along with the lower bound (Dual) generated by the primal-dual algorithm (PD). The optimal solution values (Opt) are shown for the number of time periods it could be reasonably computed, which is approximately 100 for (a) and 200 for (b).

were distributed, which resulted in needing to open a large number of orders to satisfy all the demand. It would be expected that in an optimal integer solution to such a problem, most of the orders opened would be serving close to their capacity. However, with problem instances that have capacity less constrained, it would seem likely that the myopic selection routine of *GREEDY* could backfire, which is exactly what we see in Figure 2.1(a). However, the primal-dual algorithm still manages to remain competitive even though, like the *GREEDY* heuristic, every order it opens was initially assigned all of the demand available to it. This shows the robustness of the performance of the primal-dual algorithm over different types of problem instances.

An additional benefit of using the primal-dual algorithm is that during the course of its execution it generates a polynomial number of knapsack-cover inequalities. Even if one requires an optimal integer solution, this approximation-algorithm could still be useful in both providing a good initial feasible solution, as well as a number of inequalities to strengthen the formulation. This could potentially both decrease the integrality gap as well as reduce the number of branch-and-bound nodes and total time required by the solver. For time periods ranging from 20 to 100 we compared the result of adding the generated knapsack-cover inequalities to the original formulation, the results of which are summarized in Table 2.2. The gap for the generated problem instances started off rather small, and the added knapsack-cover inequalities reduced it by only a very modest amount. There was, however, a fairly significant reduction in the number of branch-and-bound nodes required once the knapsack-cover inequalities had been added to the formulation, which amounted to roughly a factor of 4 difference. This did not result in a great decrease in the amount of time required to solve, although some time savings did occur. It seems likely though

Table 2.2: Numerical results for adding the generated knapsack-cover inequalities to the LP formulation.

Formulation		Number of Time Periods				
		20	40	60	80	100
Original	Gap	1.045	1.027	1.024	1.022	1.018
	Nodes	49	232	5898	$4.802 \cdot 10^4$	$8.076 \cdot 10^5$
	Time(s)	0.09531	0.5109	10.78	132.6	3142
Knapsack-Cover	Gap	1.043	1.025	1.022	1.019	1.016
	Nodes	7	139	3248	$1.999 \cdot 10^4$	$2.187 \cdot 10^4$
	Time(s)	0.09844	0.7328	16.20	169.0	2871

that for instances with larger gaps (especially for those significantly larger than 2) the effect of adding the knapsack-cover inequalities found by the primal-dual algorithm will be much more dramatic.

CHAPTER 3

LOCATION ROUTING PROBLEM

3.1 Introduction

In providing an air ambulance service, Ontario-based company Ornge must transport a number of non-emergency requests that consist of getting picked up at one airport and dropped off at another. These requests are typically known at least a day in advance, which provides ample time to determine a cost-effective solution. To serve the requests, a fleet of planes is based at a set of airports, each of which may be used to pick-up and drop-off patients, provided that at all times the number of patients on board does not exceed the plane's capacity. Furthermore, each plane may be limited as to the total number of requests it is allowed to serve. A solution consists of selecting a subset of planes, and assigning each selected plane a route consisting of a series of pick-ups and drop-offs that is feasible for that plane, such that all requests are satisfied. This is effectively a variant of the vehicle routing problem.

A related problem arises when adding the decision of where it is best to locate bases when providing such a service. Now we have a set of potential locations to open bases, and to simplify we will consider each request as a demand point that must be reached, and not as a pick-up/drop-off pair. To cover a request, it must be the case that we include it in some tour that includes an open base, and the cost of a solution is the cost of opening the specified bases plus the length of all the tours. This problem has much of the same feel as the well-studied facility location problem, but the key difference here is that instead of having each request directly connected to an open facility at some fixed cost

depending only on the facility-request pair, the request now is only required to be involved in some tour that involves a base. Thus the cost of having a request served by a particular base is now also dependent on the other requests connected to the base. Such situations also arise in other applications, such as deciding where to position postal fleet locations. This problem is known as the location routing problem.

We achieve two results in this chapter. For the vehicle routing problem we provide a combinatorial method for generating routes that is effective when plane capacities and the number of requests each plane can serve is limited. Our main result is achieving a primal-dual 2-approximation algorithm for the k -location-routing problem, where the number of bases that can be opened is limited to k . This is the first constant factor approximation algorithm for this problem. The location routing problem has been extensively studied in the context of finding exact solutions, including work by Laporte et al. [18, 20, 21, 22, 23]. One may also refer to Laporte [19] and Min et al. [26] for surveys of previous work. There is a recent paper of Berger, Coullard & Daskin [5] that finds exact solutions through branch-and-price style techniques. Here there are additional constraints placed on the type of tours allowed, such as limiting their total distance, which allows the total number of variables to remain manageable for a natural IP formulation given small inputs. The general location routing problem can actually be approximated using the primal-dual method of Goemans & Williamson [13] for network design problems. However, this approach does not extend to the case where there can be at most k open bases.

We modify the primal-dual approach of Goemans & Williamson to obtain a stronger type of performance guarantee, which allows us to use Lagrangian

relaxation to provide an approximation in this case. Our work builds on a result of Jain & Vazirani [15] where an approximation for uncapacitated facility location was used to create an approximation for the k -median problem through the use of Lagrangian relaxation. In that setting, a bisection search was required to determine the appropriate value to set the Lagrangian parameter so as to open nearly k facilities. Additionally, it was necessary in some cases to take a combination of two solutions to get exactly k facilities open, which resulted in the performance guarantee doubling in value. Remarkably, we are able to determine the appropriate value for the Lagrangian parameter so that exactly k bases are opened without performing a bisection search, and with no loss in the performance guarantee. The rest of this chapter is organized in the following manner. In Section 3.2 we demonstrate how to solve the vehicle routing problem optimally using a set-cover formulation, and provide a combinatorial method for efficiently generating and pricing routes. In Section 3.3.1 we define a simpler model for the location routing problem, and produce a simple dual-ascent method that has a performance guarantee of 2. In Section 3.3.2 we extend this result to apply when a constraint is added limiting the number of locations that may be opened. In Section 3.3.3 we provide the step necessary to generalize the results of the previous sections to hold for the general location routing problem. Lastly, in Section 3.3.4 we provide computational results demonstrating the performance of the primal-dual schema as compared to other heuristics.

3.2 Finding an Optimal Routing

To provide an air ambulance service, an efficient routing of planes must be found to serve a number of transportation requests, which we can model as

a vehicle routing problem. Here, we are given a set of bases, B , and a set of request pick-up/drop-off pairs, R . We consider a route to be a tour that starts at some base, serves some number of requests making sure to perform each pick-up before the associated drop-off, and then returns to the starting base. The planes being used by Ornge can only handle so many requests during a work-day, which we can capture with an upper bound, u , on the number of requests that can be served in a route. We wish to select a set of routes such that each request is covered by some route, and the cost of the selected routes is minimized. For each base $i \in B$, and each subset of requests, $T \subseteq R$, with $|T| \leq u$, let c_{iT} be the cost of the cheapest route serving all of the requests in T from base i . Using this definition we can model the problem with the following set-cover formulation:

$$\begin{aligned}
\text{opt}_{VR} &:= \min \sum_{i \in B} \sum_{\substack{T \subseteq R: \\ |T| \leq u}} c_{iT} x_{iT} && \text{(VR-IP)} \\
\text{s.t.} \quad & \sum_{i \in B} \sum_{\substack{T \subseteq R: \\ |T| \leq u, T \ni r}} x_{iT} \geq 1 && \forall r \in R \quad (3.1) \\
& x_{iT} \in \{0, 1\} && \forall i \in B, T \subseteq R : |T| \leq u
\end{aligned}$$

If u is set to a relatively small constant, as it tends to be in practice, then the size of this formulation may be small enough to solve to optimality in a reasonable amount of time. However, we have not yet specified how to find the objective function coefficients, c_{iT} .

We could use column generation and find the c_{iT} coefficients as needed, but instead we will actually use complete enumeration and find all of them initially, for the following three reasons. Firstly, the size of the problem instances considered by Ornge make this computationally feasible. Secondly, there are many additional constraints that result from applying this to Ornge's planning problem,

which affects the feasibility of a particular route. For example, some patients are infectious and should not be transported with other patients. Also, there is a patient convenience factor, which means that no patient should have to wait on a plane for an excessive amount of time before being dropped off at their destination. By using complete enumeration, we can capture any feasibility constraint that can be checked efficiently given a candidate route. This cannot be achieved using column generation, since the variable found with negative reduced cost might not be feasible according to these additional constraints. Finally, complete enumeration offers one additional advantage in that we are able to effectively use the information we've already calculated for some of the c_{iT} coefficients to more efficiently calculate other coefficients, in a dynamic programming fashion.

We will first answer the question of how many possible routes are associated with each base and request subset pairing. Our method for answering this question will provide a combinatorial insight that will allow us to efficiently enumerate and price all such routes, and thus obtain the cheapest one. Given a subset of requests, T , we can associate each route that serves T with the order in which the requests are picked up. Since there are $|T|!$ ways in which the requests can be ordered, we may restrict our attention to how many routes are associated with each ordering.

Let n_{ij} denote the number of routes associated with an ordering of i requests, which ends in j drop-offs. Clearly, we need only consider n_{ij} for $1 \leq j \leq i \leq u$. If we were to take a route and remove the last pick-up and associated drop-off, we would obtain a *sub-route* associated with an ordering with the last request removed. If we determine the number of routes for a given ordering that have the same sub-route, then we would be able to establish a recurrence relation on

n_{ij} . Another way of asking this same question, is given a particular route, how many ways can we extend it to produce a route with an ordering that has an extra specified request at the end? This last request must be picked up after all of the other requests have been picked up, so for a route ending with j drop-offs, the earliest we could pick up the last request would be immediately before the j drop-offs. If the pick-up was placed in this earliest slot then we would have $j + 1$ choices for when to drop-off the last request in the route, and would end up with a route that ends in $j + 1$ drop-offs. In general, placing the last pick-up immediately before the final k drop-offs gives $k + 1$ choices for where to place the last drop-off, and results in a route ending in $k + 1$ drop-offs. This yields the following relation:

$$n_{ij} = j \sum_{k=j-1}^{i-1} n_{i-1,k}, \quad (3.2)$$

where we define $n_{11} = 1$ and $n_{i,0} = 0$ for all i . Since each n_{ij} is determined by n_{ij} values with strictly smaller i and j , we can calculate all of these values in a dynamic programming table.

Once we have calculated all of the n_{ij} values, we can then give an expression for the total number of routes. For each $i \leq u$ we can count the total number of request subsets of size i , which has $i!$ possible orderings, and each ordering is associated with $\sum_{j=1}^i n_{ij}$ routes. Thus the total number of routes to be considered from each base is given by

$$\sum_{i=1}^u \binom{|R|}{i} \cdot i! \cdot \sum_{j=1}^i n_{ij}.$$

Moreover, this technique for calculating the number of routes for each base also provides a methodology for enumerating and pricing these routes. There are only $|R|$ routes that serve one request, since in this case we have no choice but to pick-up and then drop-off the single request before returning to base. By

calculating the price for these small routes first, we can then use this information in calculating the cost of all the other routes that are extended from these routes.

The situation is further simplified due to additional constraints that arise in practice. There is a limit on the amount of space in a plane, which means only so many patients can be loaded at a time. We can capture this by specifying a capacity of p for the number of pick-ups on board before a drop-off is required, and then the only values of n_{ij} that need be computed are those with $1 \leq j \leq i \leq u$ and $j \leq p$. In the special case where $p = 2$, it can be readily shown that the total number of routes for an ordering of i requests is simply 3 times the number of routes for an ordering of $i - 1$ requests. This simplification is only possible since we are using complete enumeration as opposed to column generation, as discussed earlier.

3.2.1 Application to Real-World Data

The technique developed above is applicable to a real client, with real data. Ornge must make a routing decision every day, based upon the transportation requests it receives. There are typically up to 30 requests that must be serviced each day, and a fleet of about 40 planes are used to provide transportation. Furthermore, each of these planes is assumed not to handle more than 4 requests before returning to base, and due to the size of the planes at most 2 patients can be transported at a time. This means that the largest subset considered has size 4, and thus has $3^{4-1}4! = 648$ possible routes associated with it. Assuming the maximum number of requests that Ornge encounters, which is 30, the total number of variables in (VR-IP) is simply the number of subsets of these

requests with cardinality at most 4, times the number of planes, which equates to 1,277,200 potential variables. There are, however, additional feasibility constraints that reduces the number of variables that need be considered to around 750,000. In a typical run of this approach on real data with 30 requests, it takes about 30 seconds to compute all the cost coefficients and generate the IP model, and about five minutes to solve this model with an off-the-shelf IP solver.

3.3 The k -Location Routing Problem

When considering finding an optimal routing, it may be the case that we could do much better if the bases were in different locations. The ideal location for the bases will depend on how the routing is formed, which suggests we should decide the base locations and the routing simultaneously, and leads to the k -location routing problem. Here we are given a set of bases and a set of requests, that are all contained in a metric space. We must choose at most k bases to open and include each request in a tour that contains an open base. The cost of a solution is simply the metric distance of all the tours. We will consider two different modifications to this problem. First, we consider a fixed-charge model, where instead of restricting there to be k open bases, we will allow any number of bases to be opened, but each base now has an opening cost. Second, in the routing with trees version, instead of having all the requests included in some tour, we simply require that each request be connected to an open base in a tree. We will begin by considering the location routing problem with fixed-charge bases and routing with trees. A previous result of Goemans & Williamson [13] considers this problem, but we provide a stronger result which yields a Lagrangian-preserving performance guarantee. This allows us to lift

this result to the k -location routing problem, but again where we are routing with trees. Finally, we show how to take our routing with trees results, and produce results for routing with tours. This can be easily done by simply short-cutting the tree and losing a factor of 2, but we show that this can be done with the same performance guarantee. For all the variants we consider, we are able to show a performance guarantees of 2.

3.3.1 Routing with Trees, Fixed-Charge Bases

In the input for the location routing problem, we have a set of potential bases, B , and set of requests R , such that $(B \cup R, d)$ comprises a metric space, where the metric $d : (B \cup R) \times (B \cup R) \mapsto \mathbb{R}_+$. Each base, $i \in B$ has an opening cost of f_i . When routing with trees, we simply require that each request has a path to an open base, which can be modeled as a Steiner tree problem on the following graph. For each base $i \in B$ and request $j \in R$ we will have an edge $\{i, j\}$ with cost equal to the metric distance between i and j . Similarly for any two distinct requests, $j, k \in R$, we will have an edge $\{j, k\}$ with cost equal to the metric distance between the two requests. Finally, we will have a root node, r , and for each base $i \in B$ we have an edge $\{r, i\}$ with cost f_i . Thus the vertex set of this graph is $V = \{r\} \cup B \cup R$ and we define the following three edge sets

$$\text{root edges: } E_r := \{\{r, i\} : i \in B\}$$

$$\text{base edges: } E_B := \{\{i, j\} : i \in B, j \in R\}$$

$$\text{request edges: } E_R := \{\{j, k\} : j, k \in R, j \neq k\}$$

The edge set of our graph is simply the union of the three edge sets defined above: $E = E_r \cup E_B \cup E_R$. Each of the nodes corresponding to a base can be

thought of as a Steiner node, so the task is to choose a set of edges such that for each request $j \in R$ there exists a path from r to j . Another way of specifying this requirement is that for any subset of nodes that does not include the root, there must be at least one edge crossing the subset. We can formulate this as the following integer program, where $\delta(S)$ is the set of all edges with exactly one endpoint in S .

$$\begin{aligned} \text{opt}_{LR} &:= \min \sum_{e \in E} c_e x_e && \text{(LR-IP)} \\ \text{s.t.} \quad &\sum_{e \in \delta(S)} x_e \geq 1 && \forall S \in \mathcal{S} \\ &x_e \in \{0, 1\} && \forall e \in E \end{aligned} \quad (3.3)$$

where $\mathcal{S} := \{S \subseteq B \cup R : S \cap R \neq \emptyset\}$. Now suppose \mathcal{A} is a collection of disjoint subsets in \mathcal{S} . Let us extend our notation so that δ can be applied to a set of subsets with the meaning $\delta(\mathcal{A}) := \bigcup_{S \in \mathcal{A}} \delta(S)$. We now propose the following constraint:

$$\sum_{e \in \delta(\mathcal{A})} x_e \geq |\mathcal{A}| \quad \forall \mathcal{A} \in \mathcal{D}, \quad (3.4)$$

where \mathcal{D} is the set of all collections of disjoint subsets in \mathcal{S} . More precisely

$$\mathcal{D} := \{\mathcal{A} \subseteq \mathcal{S} : \forall S_1, S_2 \in \mathcal{A}, S_1 = S_2 \text{ or } S_1 \cap S_2 = \emptyset\}.$$

We will first intuitively demonstrate why the constraints (3.4) are valid, before proving this fact more rigorously. Consider any $\mathcal{A} \in \mathcal{D}$. Denote the set of nodes not contained in any set in \mathcal{A} as the root cluster. For an example, see Figure 3.1. Now consider contracting each set in \mathcal{A} as well as the root cluster, down to a single node. In any feasible solution, these nodes must be connected, or else there will be a request node without a path to the root. With the addition of the root cluster, there are $|\mathcal{A}| + 1$ nodes, so to connect them all we must have

$|\mathcal{A}|$ edges, which implies that any feasible solution will satisfy the proposed constraint.

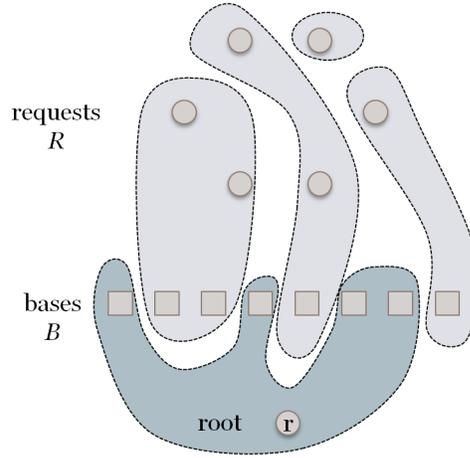


Figure 3.1: A diagram demonstrating the validity of the proposed inequalities (3.4). The lightly shaded regions depict sets in \mathcal{A} and the darker shaded region is the root cluster.

Lemma 6. *The constraints given by (3.4) are valid; any solution to (LR-IP) satisfies (3.4).*

Proof. Consider a feasible solution to (LR-IP), x , and let F be the corresponding set of edges. Fix any $\mathcal{A} \in \mathcal{D}$. Let us define two subsets of F in the following manner. Let F_1 be the set of all edges in F with only one endpoint contained in a subset in \mathcal{A} , and let F_2 be the set of all edges in F for which each endpoint is contained in a different subset in \mathcal{A} . We then have

$$|F_1| + |F_2| = \sum_{e \in \delta(\mathcal{A})} x_e.$$

Now initialize $\mathcal{A}' = \mathcal{A}$. For each edge $e \in F_2$, let us merge the subsets in \mathcal{A}' corresponding to the endpoints of e in an iterative process. When we have finished we will have that $|\mathcal{A}| - |\mathcal{A}'| = |F_2|$. After merging the subsets, we will have that

the only edges left in $\delta(\mathcal{A}') \cap F$ are those in F_1 , hence

$$|F_1| = \sum_{e \in \delta(\mathcal{A}')} x_e = \sum_{S \in \mathcal{A}'} \sum_{e \in \delta(S)} x_e \geq |\mathcal{A}'|,$$

where the inequality holds since x satisfies (3.3). This implies

$$|\mathcal{A}| = |\mathcal{A}| - |\mathcal{A}'| + |\mathcal{A}'| \leq |F_2| + |F_1| = \sum_{e \in \delta(\mathcal{A})} x_e \quad \square$$

Note that (3.4) implies (3.3) by taking \mathcal{A} to be each singleton set of a set in \mathcal{S} .

Note that since (3.4) is valid then clearly the following constraint is also valid.

$$2 \sum_{e \in \delta(\mathcal{A}) \cap E_R} x_e + \sum_{e \in \delta(\mathcal{A}) \setminus E_R} x_e \geq |\mathcal{A}| \quad \forall \mathcal{A} \in \mathcal{D}$$

We will demonstrate a primal-dual schema based on the LP that makes use of the above constraints.

$$\text{opt}_{LRP} := \min \sum_{e \in E} c_e x_e \quad (\text{LR-P})$$

$$\text{s.t. } 2 \sum_{e \in \delta(\mathcal{A}) \cap E_R} x_e + \sum_{e \in \delta(\mathcal{A}) \setminus E_R} x_e \geq |\mathcal{A}| \quad \forall \mathcal{A} \in \mathcal{D} \quad (3.5)$$

$$x_e \geq 0 \quad \forall e \in E$$

The dual of this LP is as follows:

$$\text{opt}_{LRD} := \max \sum_{\mathcal{A} \in \mathcal{D}} |\mathcal{A}| y_{\mathcal{A}} \quad (\text{LR-D})$$

$$\text{s.t. } 2 \sum_{\substack{\mathcal{A} \in \mathcal{D}: \\ \delta(\mathcal{A}) \ni e}} y_{\mathcal{A}} \leq c_e \quad \forall e \in E_R \quad (3.6)$$

$$\sum_{\substack{\mathcal{A} \in \mathcal{D}: \\ \delta(\mathcal{A}) \ni e}} y_{\mathcal{A}} \leq c_e \quad \forall e \in E \setminus E_R \quad (3.7)$$

$$y_{\mathcal{A}} \geq 0 \quad \forall \mathcal{A} \in \mathcal{D}$$

The primal-dual schema proposed will build up a set of edges, F , from which the final solution will be chosen. Initially, this set F is empty. At each point in

the algorithm, we will maintain a set of clusters, \mathcal{C} , which simply corresponds to the set of connected components in the graph when restricting the edge set to F . Since F is initially empty, the set of clusters \mathcal{C} is initially the set of all singleton nodes. We will say a cluster is *active* if it contains a request node and does not contain the root node, and otherwise it will be *inactive*. We will maintain \mathcal{A} to be the set of all active clusters in \mathcal{C} , so $\mathcal{A} := \mathcal{C} \cap \mathcal{S}$, and initially \mathcal{A} is the set of all singleton request nodes. All dual variables are implicitly set to zero to begin. The algorithm will always increase at unit rate the value of precisely one dual variable at any moment, until there are no longer any active clusters. In this sense we will keep a notion of time, which is equivalent to the sum of the current values of the dual variables.

At each point in time, we will increase the dual variable associated with \mathcal{A} until the dual constraint for one of the edges becomes tight, at which point the edge is added to F , and the two corresponding clusters are merged. If several edges become tight simultaneously, then we are free to process the edges in an arbitrary order provided that we process request edges in preference to base edges. We continue this process until \mathcal{A} is empty, so that there will no longer be any active clusters. A visualization of this process is shown in Figure 3.2.

After all clusters have become inactive (by virtue of all request nodes being contained in the root cluster) we begin the cleanup phase. If the root edge $\{r, i\} \in F$ for some $i \in B$, then we say that base i is *paid for*. We now simply remove all base edges $\{i, j\}$ for $i \in B, j \in R$ for which base i is not paid for. After removing all such base edges, we will call the resulting subset of edges F' .

We now make a few general observations about the behavior of the primal-

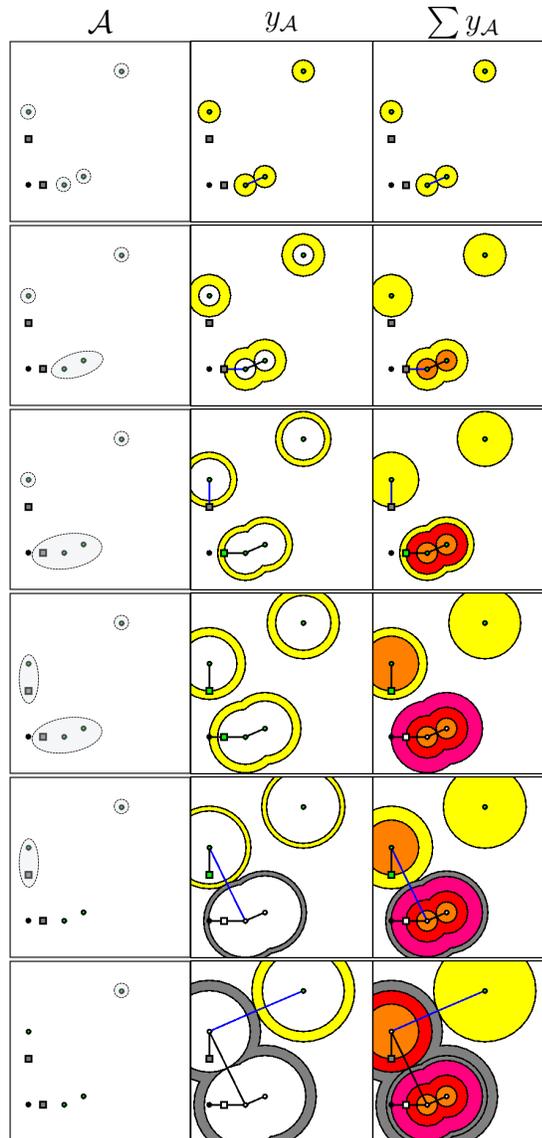


Figure 3.2: A visualization of the primal-dual schema. Here squares represent bases and circles represent requests, with the circle in the lower left being the root node. The cost of an edge corresponds to the distance between its endpoints in the plane. Each row represents a separate iteration, with the first column representing the active dual variable. In the second column the effect of increasing the active dual variable is shown, where the moats indicate the contribution to all the edges they intersect. Finally, in the third column the cumulative effect of the current dual solution is shown.

Algorithm 4: Primal-Dual for Location Routing

```
 $y, x \leftarrow 0$   
 $F \leftarrow \emptyset$   
 $\mathcal{C} \leftarrow \{\{v\} : v \in V\}$   
 $\mathcal{A} \leftarrow \{\{j\} : j \in R\}$   
while  $|\mathcal{A}| > 0$  do  
  Increase  $y_{\mathcal{A}}$  until a dual constraint becomes tight for edge  $e$   
   $F \leftarrow F \cup \{e\}$   
  Remove clusters for endpoints of  $e$  from  $\mathcal{C}$  and add the union  
   $\mathcal{A} \leftarrow \mathcal{C} \cap \mathcal{S}$   
 $F' \leftarrow F$  /* start clean-up phase */  
foreach  $\{i, j\} \in F \cap E_B$  do  
  if  $\{r, i\} \notin F$  then  
     $F' \leftarrow F' \setminus \{i, j\}$ 
```

dual algorithm. Note that initially all request edges and base edges are present in $\delta(\mathcal{A})$ and none of the root edges are. The rate at which the costs of the request and base edges are contributed to by the dual variables remain constant as long as these edges are in $\delta(\mathcal{A})$. Thus for any request edge $e \in E_R$, either e will be added to F by time $c_e/2$ or else the endpoints of e will become part of the same cluster at this time or before. Similarly any base edge $e \in E_B$ will be added to F by time c_e or else the corresponding base and request will have become part of the same cluster at or before this time. The root edges are only contributed to once the corresponding base becomes part of an active cluster. For each base $i \in B$, let $s(i)$ be the cost of the minimum-cost edge connecting this base to a request node. Hence

$$s(i) := \min\{d(i, j) : j \in R\}.$$

Lemma 7. *For each base $i \in B$, if there is a request node $j \in R$ such that the base edge $\{i, j\} \in F$, then $d(i, j) = s(i)$ and no other base edges in F are adjacent to i .*

Proof. Fix a base $i \in B$. Let $j \in R$ be a request for which $d(i, j) = s(i)$ and let

$k \in R$ be any other request. By the definition of $s(i)$ we know $d(i, k) \geq d(i, j)$, and since d is a metric we have

$$d(j, k) \leq d(i, j) + d(i, k) \leq 2d(i, k).$$

Since we choose request edges in preference to base edges, this means that before i connects to any request, that request will be contained in the same cluster as request j . Hence i can only connect to at most one request. Furthermore, if $\{i, k\}$ is still eligible to be added to F , then $\{i, j\}$ must be as well, so the only way $\{i, k\}$ could be added is if $d(i, k) \leq d(i, j) = s(i)$. Therefore if any base edge adjacent to i is added to F , it must have cost $s(i)$, and at most one such edge can be added. \square

The above lemma implies that either a base $i \in B$ will not get paid for, or else it will get paid for by time $s(i) + f_i$. This means that the primal-dual schema is equivalent to performing Kruskal's algorithm on the same graph, but with modified edge weights c'_e where

$$c'_e = \begin{cases} c_e/2, & \text{if } e \in E_R \\ c_e, & \text{if } e \in E_B \\ s(i) + f_i, & \text{if } e = \{r, i\} \in E_r \end{cases}$$

The only difference is that when running Kruskal's algorithm all of the nodes will end up connected, so we may have unpaid for bases that have a root edge but no base edge. If we adjust our definition of a paid for base to be only those bases with degree 2, then running Kruskal's algorithm is equivalent.

Lemma 8. *The set of edges F' comprises a feasible solution to (LR-IP).*

Proof. The only edges that have been removed from F are the base edges corresponding to bases that are not paid for. This means there is no root edge con-

necting to these bases, and by Lemma 7 we know there is only one base edge adjacent to these bases. Therefore any unpaid for base with an adjacent edge in F must be a leaf in the final tree formed by F . Removing these edges simply disconnects these bases without affecting the connectivity of any of the request nodes to the root. \square

Lemma 9. *For the final solution F' returned by the primal-dual algorithm, and any \mathcal{A} corresponding to a positive dual variable, the following condition holds:*

$$|\delta(\mathcal{A}) \cap F'| = |\mathcal{A}|.$$

Proof. Let \mathcal{I} be the set of singleton bases that are paid for but that are not present in any of the clusters in \mathcal{A} , and let C_r denote the root cluster at the time when the dual variable associated with \mathcal{A} was increased. Construct a graph H where the nodes correspond to the clusters in \mathcal{I} and \mathcal{A} as well as C_r , and the edges correspond to those in $\delta(\mathcal{A} \cup \mathcal{I} \cup \{C_r\}) \cap F'$. We will first argue that the graph H is a tree.

We know that all the edges in H correspond to edges that are also in F . Furthermore all the edges that make each cluster in H a connected component are also in F . The final edge set F is acyclic, and since $F' \subseteq F$ then we know that H is acyclic as well. Additionally, F' is a feasible solution, so every request node has a path to the root, and every base that is paid for also connects to the root. This implies that H is connected. Since H is connected and acyclic, it must be a tree.

Since there are $|\mathcal{I}| + |\mathcal{A}| + 1$ nodes, there must be $|\mathcal{I}| + |\mathcal{A}|$ edges in the tree H . However, this is including edges that go between two inactive clusters, which are precisely the edges not counted in $\delta(\mathcal{A}) \cap F'$. The only inactive clusters are

C_r and those in \mathcal{I} , and the only edges between these clusters are the root edges to each paid for base in \mathcal{I} . Therefore

$$|\delta(\mathcal{A}) \cap F'| = |\delta(\mathcal{A} \cup \mathcal{I} \cup \{C_r\}) \cap F'| - |\mathcal{I}| = |\mathcal{I}| + |\mathcal{A}| - |\mathcal{I}| = |\mathcal{A}|. \quad \square$$

Theorem 4. *The final solution, F' , returned by the algorithm satisfies*

$$\sum_{e \in F' \cap E_R} c_e + 2 \sum_{e \in F' \setminus E_R} c_e \leq 2 \sum_{\mathcal{A} \in \mathcal{D}} y_{\mathcal{A}},$$

which implies it is a Lagrangian-preserving 2-approximation algorithm.

Proof. An edge is only added to F if its corresponding dual constraint becomes tight, so we have that

$$\begin{aligned} \sum_{e \in F' \cap E_R} c_e + 2 \sum_{e \in F' \setminus E_R} c_e &= 2 \sum_{e \in F'} \sum_{\substack{\mathcal{A} \in \mathcal{D}: \\ \delta(\mathcal{A}) \ni e}} y_{\mathcal{A}} \\ &= 2 \sum_{\mathcal{A} \in \mathcal{D}} y_{\mathcal{A}} |\delta(\mathcal{A}) \cap F'| \\ &= 2 \sum_{\mathcal{A} \in \mathcal{D}} |\mathcal{A}| y_{\mathcal{A}}, \end{aligned}$$

where the equalities are derived by reversing the order of summation and then applying Lemma 9. □

3.3.2 Routing with Trees, k -Bases

The k -location routing problem has the same constraints as the location routing problem, but now instead of paying an opening cost for each base in the solution, we are simply limited to opening at most k bases, for some specified k . In this section we will continue considering the case where each request is only required to have a path to an open base (and hence the root). The changes

needed in (LR-IP) to model this case correspond to setting the cost of each root edge to zero, and instead imposing a constraint that at most k root edges are used. Because the primal-dual algorithm shown is Lagrangian-preserving, we can use it to approximate the k -location routing problem.

For any input to the k -location routing problem, we can set the cost of each base to zero and apply the primal-dual algorithm. If we get lucky and end up with k or fewer bases in the final solution, then by the same analysis as before we can see that the solution is a 2-approximation for the k -location routing problem as well. For the rest of this section we will assume that this case does not occur.

We now wish to determine a value λ such that if we make all bases have the same cost λ , then the primal-dual algorithm will open exactly k bases. If such a value can be found, then this too would imply a 2-approximate solution to the k -location routing problem. Let z be the optimal value of the location routing problem, and let x_e^* be the optimal solution to the k -location routing problem, then we have

$$\begin{aligned} \sum_{e \in F' \cap E_R} c_e + 2 \sum_{e \in F' \cap E_B} c_e + 2k\lambda &\leq 2z \leq 2 \sum_{e \in E_R \cup E_B} c_e x_e^* + 2\lambda \sum_{e \in E_r} x_e \\ &\leq 2 \sum_{e \in E_R \cup E_B} c_e x_e^* + 2k\lambda, \end{aligned}$$

which implies

$$\sum_{e \in F' \cap E_R} c_e + 2 \sum_{e \in F' \cap E_B} c_e \leq 2 \sum_{e \in E_R \cup E_B} c_e x_e^*,$$

hence the solution is a 2-approximation.

To determine the appropriate value of λ , we would first like to determine for each request edge $e = \{j, k\} \in E_R$, which values of λ will result in edge e being part of the final solution. We know that if edge e gets added to F , it does so at

time $c_e/2$, and otherwise j and k must have already become part of the same cluster by this time. Nodes j and k will become part of the same cluster only if there is a path between j and k where each edge has c'_e value at most $c_e/2$. For some edges, satisfying this condition will be dependent on the value of λ chosen, and for others it will not. We can determine the sensitivity of each edge on the value of λ by performing the following procedure.

We will run the primal-dual algorithm by setting each base cost to be infinite. Naturally no root edge will become tight in this setting, so we will stop the algorithm once there are only two components left in \mathcal{C} corresponding to r and $B \cup R$. For each active cluster $S \in \mathcal{A}$ we will associate a value $v(S)$ corresponding to the lowest $s(i)$ value for any base i contained in the cluster. If the cluster contains no bases then $v(S)$ will simply be equal to the current time. When a request edge $e = \{j, k\} \in E_R$ is added to the solution, and S_j and S_k correspond to the clusters containing j and k , we set $s(e) = \max\{v(S_j), v(S_k)\}$. Once the algorithm stops we set λ_k to be the largest value for which there are at most $|R| - k$ candidate edges with $c_e/2 - s(e) < \lambda_k$. More precisely we set

$$\lambda_k := \max\{\lambda : |\{e \in E_R : c_e/2 - s(e) < \lambda\}| \leq |R| - k\}.$$

Lemma 10. *Request edge $e \in E_R$ is added to the solution if $c_e/2 - s(e) < \lambda$, and if $c_e/2 = s(e)$ then this edge is added to the solution regardless of the value of λ . If $c_e/2 - s(e) = \lambda > 0$ then e can be placed in the solution or not by choice of tie-breaking.*

Proof. If $\lambda > c_e/2 - s(e)$, then the only bases that will be paid for by time $c_e/2$ are those for which $s(i) < s(e)$. By the way in which $s(e)$ was set, we know that there is at least one endpoint of e that no such base can reach by time $c_e/2$. Thus there is an active cluster S with $e \in \delta(S)$ up until time $c_e/2$, and so e will be added to the solution.

If $c_e/2 = s(e)$, then since request edges are processed before base edges, we know that for at least one of the endpoints of e the associated cluster contained no bases. This means that no base can reach this endpoint by time $c_e/2$. So again there is an active cluster S with $e \in \delta(S)$ up until time $c_e/2$, and so e will be added to the solution.

If $c_e/2 - s(e) = \lambda > 0$, then for one of the endpoints of e the associated cluster contains a base that gets tight at time $c_e/2$. The other endpoint is either already inactive or else is in the same situation. We did not specify a preference between processing request edges and root edges. So either we add edge e first in which case it is part of the solution, or else we add the root edge (possibly root edges) first, in which case both endpoints become inactive and e is not part of the solution. \square

Since no request edge is ever deleted from the solution, then each connected component in the subgraph induced by R must have precisely one paid for base in the solution connecting it to the root. If $\lambda_k = 0$, then this means we will add more than $|R| - k$ request edges, and thus open fewer than k bases, even when there are no base-opening costs. This solution is a 2-approximation the k -location routing problem, since we open fewer than k bases and no portion of the cost is attributed to opening the bases. Otherwise, if we set $\lambda = \lambda_k > 0$ then by tie-breaking appropriately we can ensure that exactly $|R| - k$ request edges are added to the solution. Hence the number of bases opened is precisely $|R| - (|R| - k) = k$ as desired, and as shown above this implies a 2-approximation as well.

3.3.3 Getting Tours From Trees with No Approximation Loss

If we wanted each request to be connected to a base through a cycle as opposed to tree, we could take the solution produced by the primal-dual algorithm and duplicate each edge, and then using shortcutting produce a disjoint set of cycles, each containing a base while not more than doubling the cost. Note that it is unnecessary to duplicate the root edges, since these edges simply represent the cost of each base, and are not required to be part of a cycle.

We can actually do better and not lose a factor of 2 in the approximation guarantee. Let us start with the same underlying graph, but make the cost of all root edges equal to *half* of the cost of the corresponding base. We say a solution to the location routing problem with cycles consists of a set of edges such that

- In the induced subgraph on R , the selected edges form a disjoint set of paths.
- The selected edges connect each such path's endpoints to some base, forming a cycle. We consider a singleton request node a path of zero-length, and there must be two copies of an edge leading from such a node to a base.
- There must be two copies selected of each root edge leading to a base that was used to connect at least one path.

The cost of this solution is the cost of the edges selected, which corresponds to the cost of each base used (since the root edge is half the cost, but we took two copies) and the cost of all edges in the cycles.

Lemma 11. *Any solution to the location routing problem with cycles satisfies the following inequality*

$$2 \sum_{e \in \delta(\mathcal{A}) \cap E_R} x_e + \sum_{e \in \delta(\mathcal{A}) \setminus E_R} x_e \geq 2|\mathcal{A}| \quad \forall \mathcal{A} \in \mathcal{D}$$

Proof. The proof will be quite similar to the proof of Lemma 6, but in this case we now require the coefficient of 2 for each request edge. Consider a feasible solution to the location routing problem with cycles, x , and let F be the corresponding set of edges. Note that now x_e will equal 2 for some edges e . Fix any $\mathcal{A} \in \mathcal{D}$. Again we will define two subsets of F , but slightly differently than before. Let

$$F_1 := (\delta(\mathcal{A}) \cap F) \setminus E_R,$$

$$F_2 := (\delta(\mathcal{A}) \cap F) \cap E_R.$$

Now initialize $\mathcal{A}' = \mathcal{A}$. For each edge $e \in F_2$, let us merge the subsets in \mathcal{A}' corresponding to the endpoints of e in an iterative process. When we have finished we will have that $|\mathcal{A}| - |\mathcal{A}'| = |F_2|$. After merging the subsets, we will have that the only edges left in $\delta(\mathcal{A}') \cap F$ are those in F_1 , though this time there may be edges leading between components of \mathcal{A}' . If we consider the subgraph consisting of the nodes R and the edges $F \cap E_R$ then we know that it consists of a disjoint set of paths. Each cluster in \mathcal{A}' must contain at least one of these disjoint paths in its entirety, since there are no longer any edges in E_R leading out of any clusters.

For each cluster in \mathcal{A}' , pick one of the disjoint request paths it contains. This path must have its endpoints connect to some base. If the base is not in the cluster, then this mean there must be two edges in F_1 associated with this cluster (or two copies of one edge in the case of a singleton request path). If the base

is in the cluster, then we root edge must be in F_1 , and the solution must select two copies of it. In either case, there are at least two edges (or two copies of one edge) in F_1 associated with each cluster. Furthermore we associate each copy of an edge in F_1 with the cluster containing the request if it is a base node, and with the cluster containing the base if it is a root node. This means the association is a bijection and hence

$$2 \sum_{e \in \delta(\mathcal{A}) \cap E_R} x_e + \sum_{e \in \delta(\mathcal{A}) \setminus E_R} x_e = 2|F_2| + \sum_{e \in F_1} x_e = 2|F_2| + 2|\mathcal{A}'| \geq 2|\mathcal{A}| \quad \square$$

We now have a LP relaxation to the location routing problem with cycles that is identical to (LR-P), except that the right-hand side of each constraint is now 2 instead of 1. This means the corresponding dual is identical to (LR-D), except that the objective function coefficient of each dual variable is now 2 instead of 1 as well. Thus the final dual solution that the algorithm ends up with is feasible for this new dual program as well, and the corresponding solution has twice the objective function cost. In other words, any feasible solution to (LR-D) has value at most half of the optimal solution with cycles, but only when the costs of the root edges have been set to half the base-opening costs.

Set $c'_e = c_e/2$ for each $e \in E_r$ and $c'_e = c_e$ for each $e \in E_B \cup E_R$. Now if we run the algorithm with the costs c'_e , duplicate each edge in the solution F' and then shortcut as necessary, we produce a solution to the location routing problem with cycles, which we will denote as x . Then we have

$$\sum_{e \in E \setminus E_r} c'_e x_e + 2 \sum_{e \in E_r} c'_e x_e \leq 2 \sum_{e \in F' \setminus E_r} c'_e + 4 \sum_{e \in F' \cap E_r} c'_e.$$

From Theorem 4 we know that this cost is at most 4 times the dual cost, which as argued above is at most 2 times the optimal cost. Thus this procedure is a Lagrangian-preserving 2-approximation algorithm for the location routing

problem with cycles, and so can also be used to produce a 2-approximation for the k -location routing problem with cycles.

3.3.4 Computational Results

In this section we provide some results on the empirical performance of the primal-dual method. Here we use a data set derived from examining several months worth of data from Ornge, as described in Section 3.2.1, as well as some additional simulated data. Since we are now working in the context of the location routing problem, we treat each origin and destination for a request pair in Ornge's dataset as a single request node, resulting in 55 requests. There are currently 26 bases being used, and we may wonder how well we could do with fewer. One way of answering this problem is to pose the k -location routing problem, with $k \leq 26$ and observe how well we can do if forced to select fewer bases. In Figure 3.3 the results of running the primal-dual are compared to both the dual solution it generates as a lower bound, as well as the optimal fractional solution, which is also a lower bound. We also ran computation experiments based on simulated data. For one set, we used 10 bases and 100 requests, where each point was uniformly distributed in the unit square. Results for this generated dataset are shown in Figure 3.4. For the second set, we used 10 bases and 50 requests, where as before the bases were uniformly distributed in the unit square. However, this time we used a bivariate Gaussian distribution to distribute requests around each base. In this manner, each base is more likely to be useful in cutting down the total distance when added to the solutions. Results from this clustered data are shown in Figure 3.5.

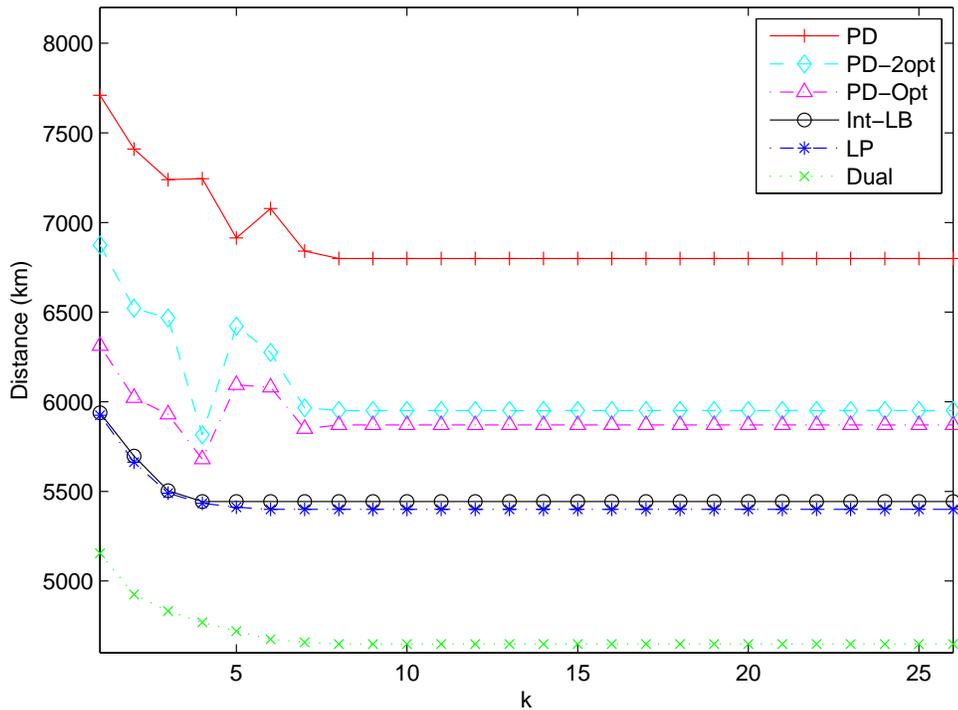


Figure 3.3: Results of running the primal-dual method on data from Ornge.

In each figure we show the solution values of the primal-dual method (PD) as well as the lower bound of the dual solution value (Dual). The tours found by the primal-dual method are the result of performing a naive shortcutting of the trees found. The solution quality can then be improved by running standard local improvement techniques, and here we show the result of applying 2-opt (PD-2opt). This technique simply considers each pair of edges in a tour, and improves the tour by removing these edges and reconnecting the nodes in a better tour when possible. In addition, we also found the optimal tours for the separate clusters induced by the solution found by the primal-dual method (PD-OPT). When these clusters are large, this is not a practical method, but when we are allowed a larger number of bases we can view the primal-dual as a tool to

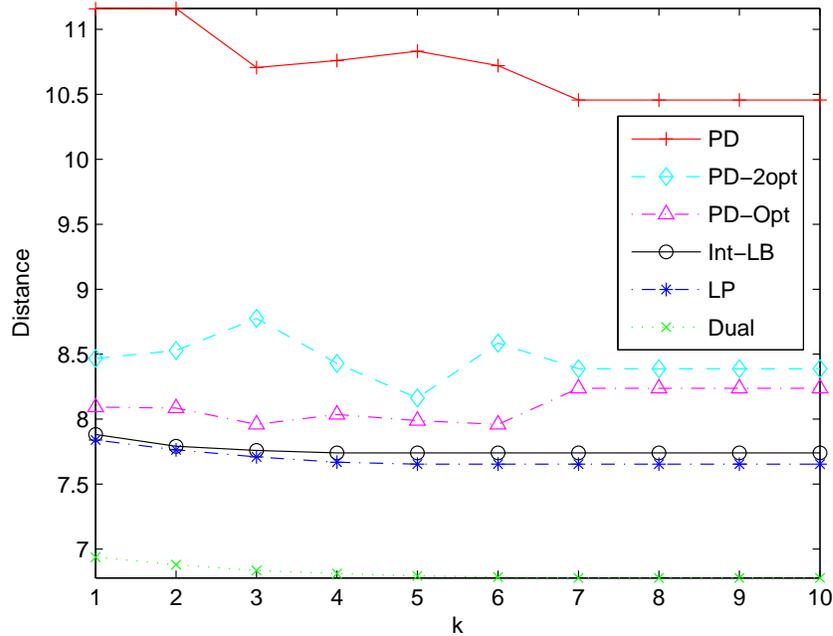


Figure 3.4: Results of running the primal-dual method on uniform, random data.

partition the data into smaller clusters, which are then feasible to solve to optimality. For comparison we solve an LP relaxation to optimality (LP), which provides a lower bound on the optimal solution value. For the random, clustered data we are able to solve the problem to optimality (OPT), making use of SCIP 1.2 [2] to perform a branch-and-cut method with SoPlex 1.4.2 [35] as the LP solver. For the other datasets that could not be solve to optimality, we provide another lower bound. Since the LP relaxation was solved with the cutting plane method, once we have found an optimal fractional solution we also have a formulation comprised of the generated cuts, which we can solve as an IP (Int-LB). This by no means is an optimal solution as it may not be feasible for the original problem, but it is an integer solution and does form an additional lower bound for comparison.

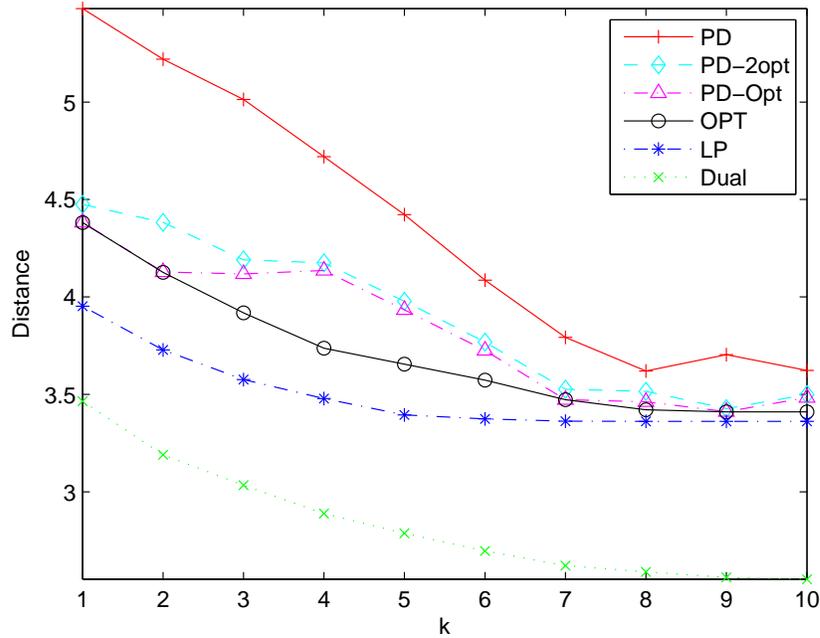


Figure 3.5: Results of running the primal-dual method on clustered, random data.

One aspect of the figures that may seem surprising at first is that the values for the primal-dual solutions are not monotonically decreasing with k . It is true that the value of the primal-dual solutions to the k -location routing problem where we are routing with trees does decrease monotonically with k . When we then shortcut these trees to form tours, however, there is no need for the monotonicity to be preserved. In other words, finding a cheaper tree does not mean we will end up with a cheaper tour after shortcutting. Because the primal-dual method runs so quickly, when solving for a particular k we can instead solve setting the base limit to each integer value that is at most k , and return the cheapest one. Making use of this technique would then, of course, cause the solution values to monotonically decrease with k .

In all of the figures, we can see a dramatic improvement in the quality of

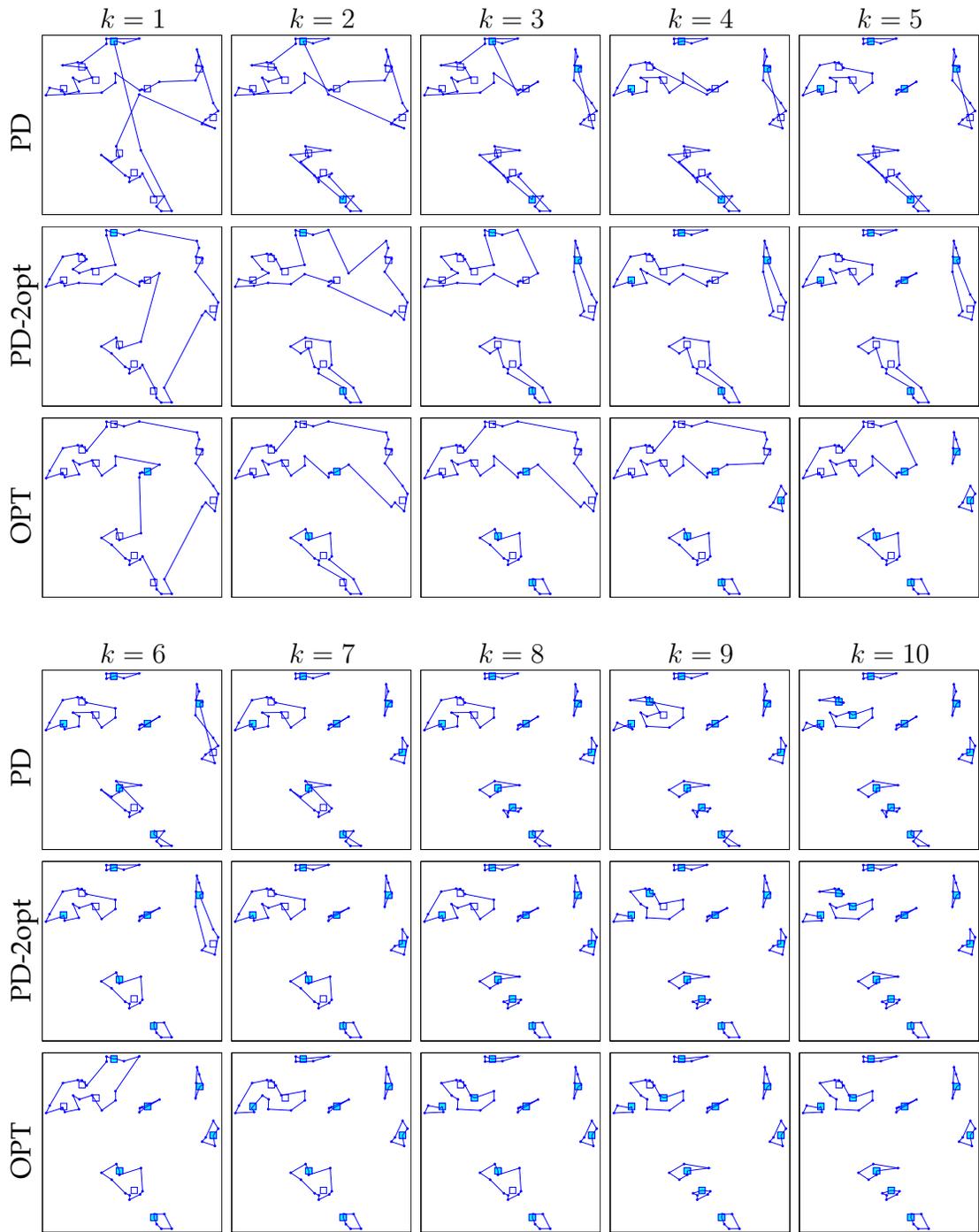


Figure 3.6: A visual depiction of tours found by the primal-dual method, both before and after local improvement, and by the optimal solution. The shaded squares correspond to the bases being used.

the primal-dual solution value when performing the 2-opt local improvement. For the random, clustered data, Figure 3.6 shows a visual depiction of the tours produced by the primal-dual method before and after applying this local improvement, as well as the tours produced by the optimal solution. We can also observe that this improvement gets rather close to the optimal tours for these clusters resulting from the primal-dual solution.

In the results based on data from Ornge and the uniformly, randomly generated data, we see that all of the curves quickly plateau and no longer change with increasing k . This can be attributed to the uncapacitated nature of the problem we are considering. After a certain number of bases are allowed, it can be cheaper to keep requests in a longer tour than to connect them to an additional base. Even though we do not have to pay an additional cost to add a base, each base that is part of the solution results in an additional edge that must be included. When examining the results from the clustered data, we see that we continue to get a significant decrease in cost as we increase k , especially for the primal-dual solution values. For larger k , we observe that the primal-dual is quite close to optimality. Even though we demonstrated a theoretical guarantee of 2, the algorithm may achieve values much closer to optimality in practice.

CHAPTER 4
GENERALIZED ASSIGNMENT PROBLEM

4.1 Introduction

The generalized assignment problem can be framed as the following scheduling problem. We are given a set of machines, \mathcal{M} , and a set of jobs, \mathcal{J} , where scheduling job j on machine i takes up p_{ij} units of machine i 's processing time, and costs c_{ij} . Our goal is to schedule all the jobs on machines, so as to both minimize the cost as well as to minimize the makespan, which is the maximum time any machine spends processing jobs. This can also be framed as an optimization problem with a single objective, where given a value T we want to find the minimum-cost assignment that has makespan at most T . A standard integer programming formulation of this problem is given below.

$$\text{opt}_{GA} := \min \sum_{i=1}^m \sum_{j=1}^n c_{ij} x_{ij} \quad (\text{GA-IP})$$

$$\text{s.t.} \sum_{j=1}^n p_{ij} x_{ij} \leq T \quad \forall i \quad (4.1)$$

$$\sum_{i=1}^m x_{ij} = 1 \quad \forall j \quad (4.2)$$

$$x_{ij} \in \{0, 1\} \quad \forall i, j$$

Shmoys and Tardos [30] use a formulation that is the decision version of (GA-IP) to produce an LP-rounding result that delivers an assignment with cost at most opt_{GA} and makespan at most $2T$. Singh [31] is able to produce an equivalent result using a method of *iterative rounding*. Here, a series of LP relaxations are solved, and each time at least some of the variables get fixed to integer val-

ues, and some of the constraints get removed.

This problem has an important special case where each machine may only serve a subset of jobs, but the processing time incurred for each allowed pairing depends only on the job. In this setting, we are able to produce a simple dual-ascent method that matches the performance guarantee of the LP-rounding result of Shmoys and Tardos. While we are able to show that this method terminates in a finite number of steps, we have not yet proved a polynomial bound on the running time. We are able to prove polynomial running time, however, in another important case where the processing time for allowed pairings is independent of both jobs and machines.

The remainder of this chapter is structured in the following way. In Section 4.2 we provide an alternative formulation to be used as an LP relaxation, which is based on viewing the assignment as a flow. In Section 4.3 we give the details of the primal-dual method and the proof of its performance guarantee. Finally, in Section 4.4 we show the running time of the algorithm is polynomial for a special case.

4.2 A New Flow-Based Formulation for the $\{p_j, \infty\}$ Case

We will now concentrate on a special case of this problem, where we require that for each processing time $p_{ij} \in \{p_j, \infty\}$. In other words, machines may only serve some subset of the jobs, but any machine that can serve a particular job will spend the same amount of processing time on it. We will introduce new valid inequalities based on viewing the assignment of jobs to machines as a network flow. Let us construct a bipartite graph, G , with one set of nodes

corresponding to machines, and the other set of nodes corresponding to jobs. For any machine i and job j with $p_{ij} = p_j$, create the edge (i, j) with capacity p_{ij} . Now add a source node s and make an edge (s, i) with capacity T for each machine $i \in \mathcal{M}$. Finally add a sink t and create an edge (j, t) with capacity p_j for each job j . Let us call this graph, G . If $E \subseteq \{(i, j) : p_{ij} = p_j\}$, then let $G(E)$ be the graph G with all the edges between machines and jobs removed, except for those in E . Now take a subset of machines, $M \subseteq \mathcal{M}$, and a subset of jobs, $J \subseteq \mathcal{J}$. Let $M^C := \mathcal{M} \setminus M$ and $J^C := \mathcal{J} \setminus J$, and consider the cut in G with source-side nodes $S := \{s\} \cup M \cup J^C$. Such a cut is shown in Figure 4.1. We can

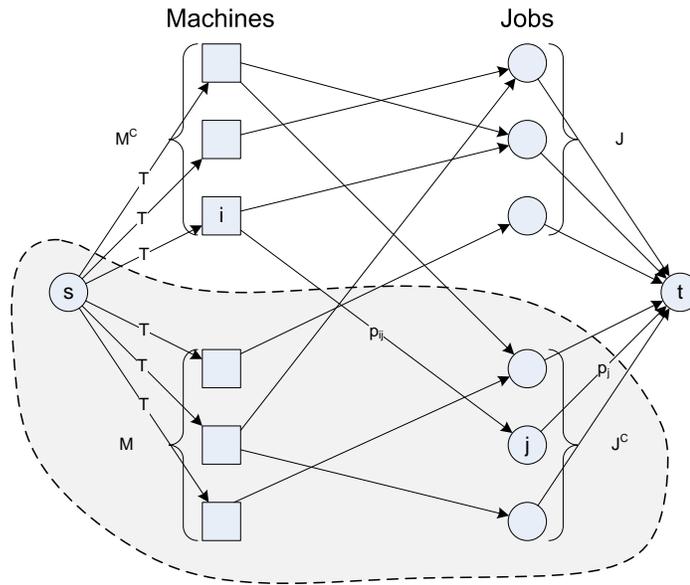


Figure 4.1: The network flow view of the assignment problem with a cut demonstrating the validity of the inequalities (4.4) with respect to a particular partition of machines and jobs.

view any assignment as a flow on this graph with value $\sum_j p_j$. That is, given values of x_{ij} that corresponds to a feasible assignment, we can set the flow on edge (i, j) to be $p_j x_{ij}$. The edges (s, i) will have flow equal to the total processing

time machine i spends, i.e.

$$\sum_{j=1}^n p_j x_{ij},$$

and the edges (j, t) will all have flow equal to p_j . We know that the sum of the flow on edges leading out of S minus the sum of flow on edges leading into S must be equal to the total flow. This implies the equality

$$\sum_{\substack{i \in M, j \in J: \\ p_{ij} = p_j}} p_j x_{ij} + \sum_{i \in M^C} \sum_{j \in \mathcal{J}} p_j x_{ij} + \sum_{j \in J^C} p_j - \sum_{\substack{i \in M^C, j \in J^C: \\ p_{ij} = p_j}} p_j x_{ij} = \sum_{j \in \mathcal{J}} p_j. \quad (4.3)$$

For each $i \in M^C$, we can upper bound its total time spent processing jobs by T , and then substituting this relation into the above equation and rearranging yields constraints that provide the basis for the following LP relaxation.

$$\text{opt}_{GAP} := \min \sum_{i=1}^m \sum_{j=1}^n c_{ij} x_{ij} \quad (\text{GAP-P})$$

$$\text{s.t.} \quad \sum_{\substack{i \in M, j \in J: \\ p_{ij} = p_j}} p_j x_{ij} + T|M^C| \geq \sum_{j \in J} p_j + \sum_{\substack{i \in M^C, j \in J^C: \\ p_{ij} = p_j}} p_j x_{ij} \quad \forall M \subseteq \mathcal{M}, J \subseteq \mathcal{J} \quad (4.4)$$

$$x_{ij} \in \{0, 1\} \quad \forall i, j$$

The dual of this LP is as follows:

$$\text{opt}_{GAD} := \max \sum_{\substack{M \subseteq \mathcal{M}, \\ J \subseteq \mathcal{J}}} \left(\sum_{j \in J} p_j - T|M^C| \right) v(M, J) \quad (\text{GAP-D})$$

$$\text{s.t.} \quad \sum_{\substack{M \subseteq \mathcal{M}, J \subseteq \mathcal{J}: \\ M \ni i, J \ni j}} p_j v(M, J) \leq c_{ij} + \sum_{\substack{M \subseteq \mathcal{M}, J \subseteq \mathcal{J}: \\ M^C \ni i, J^C \ni j}} p_j v(M, J) \quad \forall i, j : p_{ij} = p_j \quad (4.5)$$

$$v(M, J) \geq 0 \quad \forall M \subseteq \mathcal{M}, J \subseteq \mathcal{J}$$

4.3 A Primal-Dual Approach for the $\{p_j, \infty\}$ Case

We now present a primal-dual method based on the derived formulation. We begin with none of the jobs assigned to any machines, and with all dual variables equal to zero. At any given time during the running of the method, some dual variable will be raised until raising it any further would violate a dual constraint. We say a machine is *full* if the sum of the processing times of the jobs assigned to it is at least T . We say a job is *critical* if the machine it is assigned to is full, and if removing this job from this machine would cause it to no longer be full.

At the beginning we set $M := \mathcal{M}$ and $J := \mathcal{J}$, and raise the variable $v(M, J)$. We will maintain E to be the set of edges (i, j) such that the dual constraint corresponding to (i, j) is tight. Initially E will be empty. At any point in time we can think of the current assignment of jobs to machines as a flow on the graph $G(E)$, and we can also form the residual graph of this flow. We say a machine is *critical* if there is a critical job assigned to it that is reachable from the source in the residual graph.

We will maintain M to be the set of all machines that are either not full, or else critical, and J will be the set of all jobs that are not assigned to machines in M . When a dual constraint (i, j) becomes tight, we add (i, j) to E , and assign job j to machine i , unless j is critical, in which case we simply move the machine it is assigned to into M , and move all other jobs assigned to that machine out of J .

There are several things to notice about this method. First, the only machines that ever leave the set M are full, and once a machine becomes full, it stays

Algorithm 5: Primal-Dual Method

```
 $x, v \leftarrow 0$   
 $M \leftarrow \mathcal{M}, J \leftarrow \mathcal{J}$   
 $E \leftarrow \emptyset$   
while  $\sum_{j \in J} p_j > T|M^C|$  do  
  Increase  $v(M, J)$  until dual constraint corresponding to  $(i, j)$  becomes  
  tight  
   $E \leftarrow E \cup \{(i, j)\}$   
   $E \leftarrow E \setminus \{(i, j) : i \in M^C, j \in J^C\}$   
  if  $j$  is not critical then  
    Assign  $j$  to  $i$   
   $M \leftarrow$  all machines either critical or not full  
   $J \leftarrow$  all jobs not assigned to machines in  $M$ 
```

full. This is because for a machine to no longer be full, a job would have to be reassigned from it that would drop its load below T , but by definition this job would be critical. Since the method does not allow for critical jobs to be reassigned, no full machine will ever have their load drop to less than T .

Lemma 12. *In the solution produced by Algorithm 5, the makespan is at most $2T$.*

Proof. Each machine begins with no jobs assigned to it. For a job to be assigned to a machine, that machine must either have load less than T , or else be a critical machine. If the load is less than T , then adding a job cannot cause the load to increase up to $2T$, since by assumption $p_j \leq T$ for all j . If the machine is critical, then it has a critical job j assigned to it, and hence its load is within p_j of T . If the job j remains critical after the assignment, then certainly the load of the machine is still less than $2T$. If the job j is no longer critical, then it will get reassigned to another machine, and since removing j would leave the machine less than full, then the net resulting load of the machine must still be less than $2T$. □

Lemma 13. *Upon completion of Algorithm 5, for any M, J such that $v(M, J) > 0$, we have*

$$\sum_{\substack{i \in M, j \in J: \\ p_{ij} = p_j}} p_j x_{ij} + T|M^C| - \sum_{\substack{i \in M^C, j \in J^C: \\ p_{ij} = p_j}} p_j x_{ij} \leq \sum_{j \in J} p_j.$$

Proof. For any dual variable $v(M, J)$ that was increased, we have already argued above that in the final assignment, any $i \in M^C$ is full. We argued above that the load of any $i \in M^C$ is at least T , so

$$\sum_{i \in M^C} \sum_{j \in \mathcal{J}} p_j x_{ij} \geq T|M^C|.$$

Thus, substituting the above inequality into equation (4.3) yields the desired result. \square

We only assign a job j to a machine i if the dual constraint corresponding to (i, j) is tight. This fact, along with Lemmas 12, 13, and 1, yields the following theorem.

Theorem 5. *Algorithm 5 produces a solution with cost no more than $\text{opt}_{GA'}$ and makespan at most $2T$.*

4.4 A Running Time Bound for the $\{p, \infty\}$ Case

The method provides for a solution that meets the previous results upon termination, but no word has yet been said about its running time, or even if it is polynomial. In order to bound the running time, we will actually be considering an even more specialized case, where the p_j values are not job-dependent, that is $p_j = p$ for all j . In this special case, every time a job is reassigned, at

least one job is given to a machine that is not yet full. This is because the only machines that are full which have jobs reassigned to them are critical machines. But if a critical machine gets a job, then since we now know the newly assigned job has the same processing time as all critical jobs, this means this machine will no longer have any critical jobs. Thus it will pass on one of its previously critical jobs to either another critical machine, or else a machine that is not full. If it passes on to another critical machine, we can repeat this argument, and see that eventually a job must be passed to a machine that is not full.

The only activity that happens during the method which does not involve a machine becoming full, is when a machine becomes critical. But at worst, we have that each machine becomes critical once before starting another chain of job reassignments. This reasoning provides an upper bound of $O(m^2n)$ on the total running time for this special case.

CHAPTER 5

CONCLUSION

This thesis extends the application of the primal-dual schema, and further demonstrates the power and versatility of the simple dual-ascent method. In all of the results presented, we are able to identify strengthened inequalities that comprise the LP relaxations we use, which are not the direct result of aggregating standard inequalities for that problem. In this way we can view constraints as specifying exactly how we want to modify a given solution, and framing a suitable primal-dual method based around them. In Chapter 4 we show a dual-ascent procedure where edges are removed as well as added to the solution as it is modified, and not just in the reverse delete step. This addresses one of the perceived limitations of the dual-ascent method, and shows that it can be more versatile than a sophisticated greedy approach.

One long-standing open question concerns whether there exists an LP-based approximation algorithm for the capacitated facility location problem with a constant performance guarantee. Here one is given a set of facilities and a set of clients that exist in a metric space. The goal is to connect each client to a facility, while minimizing the cost of that assignment. To connect a client to a facility incurs the cost of the metric distance between them, and any facility that is used to connect a client incurs a specified facility opening cost. The current best known approximation algorithm is due to Zhang, Chen & Ye [36] and achieves a performance guarantee of 5.83, but is based on local search techniques. We believe that our results provide a potential foothold for obtaining a primal-dual schema for this problem. In particular, inequalities similar to those used in Chapter 4 can be derived for the capacitated facility location problem, which imply a nat-

ural algorithm. It remains to be seen whether a relaxed form of complementary slackness can be shown for such a method, or if new ideas can be used to prove a constant performance guarantee.

BIBLIOGRAPHY

- [1] K. Aardal, Y. Pochet, and L. A. Wolsey. Capacitated facility location: valid inequalities and facets. *Math. Oper. Res.*, 20(3):562–582, 1995.
- [2] T. Achterberg. *Constraint Integer Programming*. PhD thesis, Technische Universität Berlin, 2007.
- [3] A. Agrawal, P. Klein, and R. Ravi. When trees collide: an approximation algorithm for the generalized Steiner problem on networks. *SIAM J. Comput.*, 24(3):440–456, 1995.
- [4] R. Bar-Yehuda and S. Even. A linear-time approximation algorithm for the weighted vertex cover problem. *J. Algorithms*, 2(2):198–203, 1981.
- [5] R. T. Berger, C. R. Coullard, and M. S. Daskin. Location-routing problems with distance constraints. *Transportation Science*, 41(1):29–43, 2007.
- [6] D. Bertsimas and C. P. Teo. From valid inequalities to heuristics: a unified view of primal-dual approximation algorithms in covering problems. *Operations Research*, 46(4):503–514, 2003.
- [7] R. D. Carr, L. Fleischer, V. J. Leung, and C. A. Phillips. Strengthening integrality gaps for capacitated network design and covering problems. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 106–115, 2000.
- [8] J. Chuzhoy and J. Naor. Covering problems with hard capacities. *SIAM J. Comput.*, 36(2):498–515, 2006.
- [9] V. Chvátal. A greedy heuristic for the set-covering problem. *Math. Oper. Res.*, 4(3):233–235, 1979.
- [10] G. B. Dantzig, L. R. Ford, and D. R. Fulkerson. A primal-dual algorithm for linear programs. In H. W. Kuhn and A. W. Tucker, editors, *Linear Inequalities and Related Systems*, pages 171–181. Princeton University Press, Princeton, NJ, 1956.
- [11] G. Even, R. Levi, D. Rawitz, B. Schieber, S. Shahar, and M. Sviridenko. Algorithms for capacitated rectangle stabbing and lot sizing with joint set-up costs. Submitted to *ACM Trans. on Algorithms*, 2007.

- [12] R. Gandhi, E. Halperin, S. Khuller, G. Kortsarz, and A. Srinivasan. An improved approximation algorithm for vertex cover with hard capacities. *Journal of Computer and System Sciences*, 72(1):16–33, 2006.
- [13] M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24(2):296–317, 1995.
- [14] M. X. Goemans and D. P. Williamson. The primal-dual method for approximation algorithms and its application to network design problems. In D. S. Hochbaum, editor, *Approximation algorithms for NP-hard problems*, pages 144–191. PWS Publishing Company, 1997.
- [15] K. Jain and V. V. Vazirani. Approximation algorithms for metric facility location and k -median problems using the primal-dual schema and Lagrangian relaxation. *J. ACM*, 48(2):274–296, 2001.
- [16] D. S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. System Sci.*, 9(3):256–278, 1974.
- [17] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.
- [18] G. Laporte. Generalized subtour elimination constraints and connectivity constraints. *J. Oper. Res. Soc.*, 37(5):509–514, 1986.
- [19] G. Laporte. Location-routing problems. In B. L. Golden and A. A. Assad, editors, *Vehicle Routing: Methods and Studies*, pages 163–198. North Holland, Amsterdam, The Netherlands, 1986.
- [20] G. Laporte and Y. Nobert. An exact algorithm for minimizing routing and operating costs in depot location. *Eur. J. Oper. Res.*, 6(2):224–226, 1981.
- [21] G. Laporte, Y. Nobert, and D. Arpin. An exact algorithm for solving a capacitated location-routing problem. *Ann. Oper. Res.*, 6(2):293–310, 1986.
- [22] G. Laporte, Y. Nobert, and P. Pelletier. Hamiltonian location problems. *Eur. J. Oper. Res.*, 12(1):82–89, 1983.
- [23] G. Laporte, Y. Nobert, and S. Taillefer. Solving a family of multi-depot vehicle routing and location-routing problems. *Transportation Sci.*, 22(3):161–172, 1988.

- [24] R. Levi, A. Lodi, and M. Sviridenko. Approximation algorithms for the multi-item capacitated lot-sizing problem via flow-cover inequalities. In *Proceedings of the 12th International IPCO Conference*, Lecture Notes in Computer Science, pages 454–468. Springer, 2007.
- [25] R. Levi, R. Roundy, and D. B. Shmoys. Primal-dual algorithms for deterministic inventory problems. *Math. Oper. Res.*, 31(2):267–284, 2006.
- [26] H. Min, V. Jayaraman, and R. Srivastava. Combined location-routing problems: A synthesis and future research directions. *Eur. J. Oper. Res.*, 108(1):1–15, 1998.
- [27] M. W. Padberg, T. J. van Roy, and L. A. Wolsey. Valid linear inequalities for fixed charge problems. *Operations Research*, 33(4):842–861, 1985.
- [28] Y. Pochet and L. A. Wolsey. Single item lot-sizing with non-decreasing capacities. *Math. Program.*, 121(1):123–143, 2009.
- [29] Y. Sharma and D. Williamson. Personal communication, 2007.
- [30] D. B. Shmoys and É. Tardos. An approximation algorithm for the generalized assignment problem. *Math. Program.*, 62:461–474, 1993.
- [31] M. Singh. *Iterative Rounding and Relaxation*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 2008.
- [32] C. P. M. van Hoesel and A. P. M. Wagelmans. Fully polynomial approximation schemes for single-item capacitated economic lot-sizing problems. *Math. Oper. Res.*, 26(2):339–357, 2001.
- [33] V. V. Vazirani. Primal-dual schema based approximation algorithms. In G. B. Khosrovshahi, A. Shokoufandeh, and M. A. Shokrollahi, editors, *Theoretical Aspects of Computer Science*, volume 2292 of *Lecture Notes in Computer Science*, pages 198–207. Springer, 2000.
- [34] V. G. Vizing. On an estimate of the chromatic class of a p -graph. *Diskret. Analiz.*, 3:23–30, 1964.
- [35] R. Wunderling. *Paralleler und objektorientierter Simplex-Algorithmus*. PhD thesis, Technische Universität Berlin, 1996.

- [36] J. Zhang, B. Chen, and Y. Ye. A multi-exchange local search algorithm for the capacitated facility location problem. In *Math. Oper. Res.*, pages 389–403, 2004.