FUNDAMENTALS OF SUPPLY CHAIN THEORY

Second Edition

Lawrence V. Snyder
Lehigh University

Zuo-Jun Max Shen
University of California, Berkeley

WILEY-INTERSCIENCE

A JOHN WILEY & SONS, INC., PUBLICATION

Library of Congress Cataloging-in-Publication Data:

Fundamentals of Supply Chain Theory / Lawrence V. Snyder and Zuo-Jun Max Shen.
p. cm.—(Wiley series in XXX)
"Wiley-Interscience."
Includes bibliographical references and index.
ISBN XXX (pbk.)
1. Supply Chain Management. 2. Operations Research sciences—Research—Supply Chain Management.

Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

CHAPTER 8

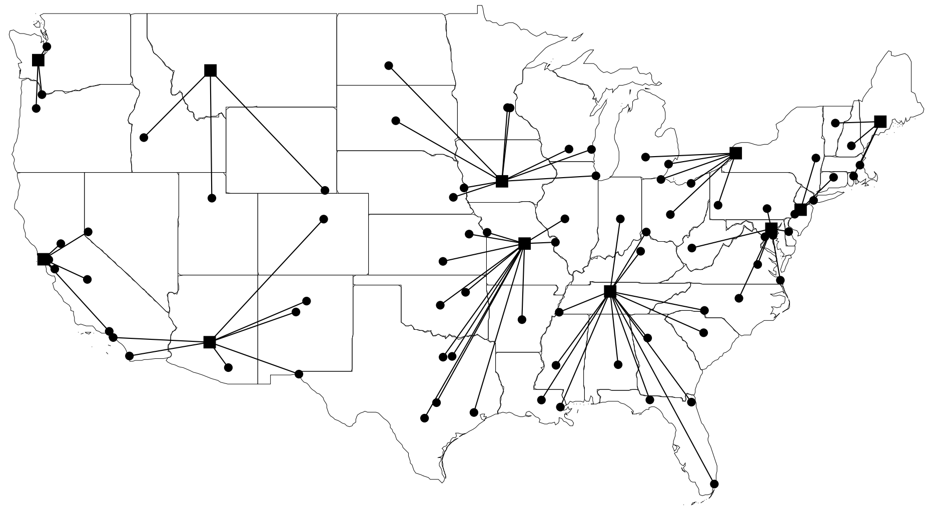
FACILITY LOCATION MODELS

8.1 INTRODUCTION

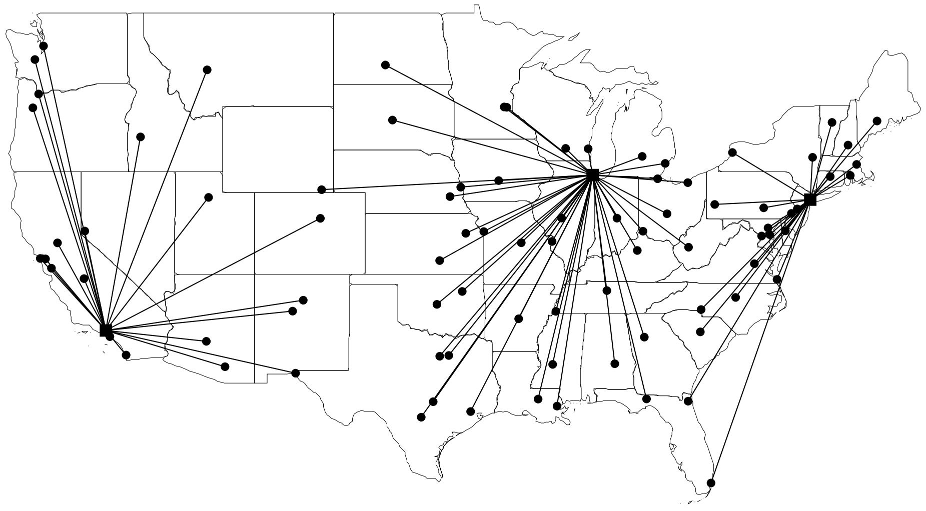
One of the major strategic decisions faced by firms is the number and locations of factories, warehouses, retailers, or other physical facilities. This is the purview of a large class of models known as *facility location problems*. The key trade-off in most facility location problems is between the facility cost and customer service. If we open a lot of facilities (Figure 8.1(a)), we incur high facility costs (to build and maintain them), but we can provide good service since most customers are close to a facility. On the other hand, if we open few facilities (Figure 8.1(b)), we reduce our facility costs but must travel farther to reach our customers (or they to reach us).

Most (but not all) location problems make two related sets of decisions: (1) where to locate, and (2) which customers are assigned or allocated to which facilities. Therefore, facility location problems are also sometimes known as *location–allocation problems*.

A huge range of approaches has been considered for modeling facility location decisions. These differ in terms of how they model facility costs (for example, some include the costs explicitly, while others impose a constraint on the number of facilities to be opened) and how they model customer service (for example, some include a transportation cost, while others require all or most facilities to be *covered*—that is, served by a facility that is within some specified distance). Facility location problems come in a great variety of flavors based on what types of facilities are to be located, whether the facilities are capacitated, which (if any) elements of the problem are stochastic, what topology the facilities may be located



(a) Many facilities open.



(b) Few facilities open.

Figure 8.1 Facility location configurations. Squares represent facilities; circles represent customers.

on (e.g., on the plane, in a network, or at discrete points), how distances or transportation costs are measured, and so on. Several excellent textbooks provide additional material for the interested reader; for example, see Mirchandani and Francis (1990), Drezner (1995a), Drezner and Hamacher (2002), or Daskin (2013). For an annotated bibliography of papers on facility location problems, see ReVelle et al. (2008b). The book by Eiselt and Marianov (2011) contains chapters on a number of seminal papers in facility location, each describing the original contribution as well as later extensions.

In addition to supply chain facilities such as plants and warehouses, location models have been applied to public sector facilities such as bus depots and fire stations, as well as to telecommunications hubs, satellite orbits, bank accounts, and other items that are not really "facilities" at all. In addition, many operations research problems can be formulated as facility location problems or have subproblems that resemble them. Facility location problems are often easy to state and formulate but are difficult to solve; this makes them a popular testing ground for new optimization tools. For all of these reasons, facility location problems are an important topic in operations research, and in supply chain management in particular, in both theoretical and applied work.

In this chapter, we will begin by discussing a classical facility location model, the uncapacitated fixed-charge location problem (UFLP), in Section 8.2. The UFLP and its descendants have been deployed more widely in supply chain management than perhaps any other location model. One reason for this is that the UFLP is very flexible and, although it is NP-hard, lends itself to a variety of effective solution methods. Another reason is that the UFLP includes explicit costs for both key elements of the problem—facilities and customer service—and is therefore well suited to supply chain applications.

In Section 8.3, we discuss other so-called *minisum* models (in particular, the p-median problem and a capacitated version of the UFLP), and in Section 8.4, we discuss *covering* models (including the p-center, set covering, and maximal covering problems). We briefly discuss a variety of other deterministic facility location problems in Section 8.5. In Section 8.6, we introduce stochastic and robust models for facility location under uncertainty. We then discuss models for *network design*—a close cousin of facility location—in Section 8.7.

8.2 THE UNCAPACITATED FIXED-CHARGE LOCATION PROBLEM

8.2.1 Problem Statement

The *uncapacitated fixed-charge location problem* (UFLP) chooses facility locations in order to minimize the total cost of building the facilities and transporting goods from facilities to customers. The UFLP makes location decisions for a single echelon, and the facilities in that echelon are assumed to serve facilities in a downstream echelon, all of whose locations are fixed. We will tend to refer to the facilities in the upstream echelon as *distribution centers* (DCs) or *warehouses* and to those in the downstream echelons as *customers*. However, the model is generic, and the two echelons may instead contain other types of facilities—for example, factories and warehouses, or regional and local DCs, or even fire stations and homes. Sometimes it's also useful to think of an upstream echelon, again with fixed location(s), that serves the DCs.

Each potential DC location has a *fixed cost* that represents building (or leasing) the facility; the fixed cost is independent of the volume that passes through the DC. There is a *transportation cost* per unit of product shipped from a DC to each customer. There is a single product. The DCs have no capacity restrictions—any amount of product can be handled by any DC. (We'll relax this assumption in Section 8.3.1.) The problem is to choose facility locations to minimize the fixed cost of building facilities plus the transportation cost to transport product from DCs to customers, subject to constraints requiring every customer to be served by some open DC.

As noted above, the key trade-off in the UFLP is between fixed and transportation costs. If too few facilities are open, the fixed cost is small, but the transportation cost is large because many customers will be far from their assigned facility. On the other hand, if too many facilities are open, the fixed cost is large, but the transportation cost is small. The UFLP tries to find the right balance, and to optimize not only the number of facilities, but also their locations.

8.2.2 Formulation

Define the following notation:

Sets

I	= set of customers
J	= set of potential facility locations

Parameters

h_i	= annual demand of customer $i \in I$
c_{ij}	= cost to transport one unit of demand from facility $j \in J$ to customer $i \in I$
f_j	= fixed annual cost to open a facility at site $j \in J$

Decision Variables

x_j	= 1 if facility j is opened, 0 otherwise
y_{ij}	= the fraction of customer i's demand that is served by facility j

The transportation costs c_{ij} might be of the form $k \times$ distance for some constant k (if the shipping company charges k per mile per unit) or may be more arbitrary (for example, based on airline ticket prices, which are not linearly related to distance). In the former case, distances may be computed in a number of ways:

- *Euclidean distance*: The distance between (a_1, b_1) and (a_2, b_2) is given by

$$\sqrt{(a_1 - a_2)^2 + (b_1 - b_2)^2}.$$

  The Euclidean distance metric is also known as the ℓ_2 *norm*. This is an intuitive measure of distance but is not usually applicable in supply chain contexts because Cartesian coordinates are not useful for describing real-world locations.

- *Manhattan or rectilinear metric*: The distance is given by

$$|a_1 - a_2| + |b_1 - b_2|.$$

  This metric assumes that travel is only possible parallel to the x- or y-axis, e.g., travel along city streets. It is also known as the ℓ_1 *norm*.

- *Great circle*: This method for calculating distances takes into account the curvature of the earth and, more importantly, takes latitudes and longitudes as inputs and returns distances in miles or kilometers. Great circle distances assume that travel occurs over a great circle, the shortest route over the surface of a sphere. Let (α_1, β_1) and (α_2, β_2) be the latitude and longitude of two points in radians, and let $\Delta\alpha \equiv \alpha_1 - \alpha_2$ and $\Delta\beta \equiv \beta_1 - \beta_2$ be the differences in the latitude and longitude (respectively). Then the great circle distance between the two points is given by

$$2r \arcsin\left(\sqrt{\sin^2\left(\frac{\Delta\alpha}{2}\right) + \cos\alpha_1 \cos\alpha_2 \sin^2\left(\frac{\Delta\beta}{2}\right)}\right), \tag{8.1}$$

where r is the radius of the Earth, approximately 3958.76 miles or 6371.01 km (on average), and the trigonometric functions are assumed to use radians.

A simpler formula, known as the *spherical law of cosines*, sets the distance equal to

$$r \arccos\left(\sin\alpha_1 \sin\alpha_2 + \cos\alpha_1 \cos\alpha_2 \cos\left(\Delta\beta\right)\right) \tag{8.2}$$

and is nearly as accurate as (8.1) except when the distance between the two points is very small. (See Problem 8.44.)

- *Highway/network*: The distance is computed as the shortest path within a network, for example, the US highway network. This is usually the most accurate method for calculating distances in a supply chain context. However, since they require data on the entire road network, they must be obtained from geographic information systems (GIS) or from online services such as Mapquest or Google Maps. (In contrast, the distance measures above can be calculated from simple formulas using only the coordinates of the facilities and customers.)

- *Matrix*: Sometimes a matrix containing the distance between every pair of points is given explicitly. This is the most general measure, since all others can be considered a special case. It is also the only possible measure when the cost structure exhibits no particular pattern—for example, when they are based on airline ticket prices.

In general, we won't be concerned with how transportation costs are computed—we'll assume they are given to us already as the parameters c_{ij}.

The UFLP is formulated as follows:

$$\begin{aligned}
\text{(UFLP)} \quad \text{minimize} \quad & \sum_{j \in J} f_j x_j + \sum_{i \in I} \sum_{j \in J} h_i c_{ij} y_{ij} && && (8.3)\\
\text{subject to} \quad & \sum_{j \in J} y_{ij} = 1 && \forall i \in I && (8.4)\\
& y_{ij} \le x_j && \forall i \in I, \forall j \in J && (8.5)\\
& x_j \in \{0,1\} && \forall j \in J && (8.6)\\
& y_{ij} \ge 0 && \forall i \in I, \forall j \in J && (8.7)
\end{aligned}$$

Formulations very similar to this were originally proposed by Manne (1964) and Balinski (1965). The objective function (8.3) computes the total (fixed plus transportation) cost. In the discussion that follows, we'll use z^* to denote the optimal objective value of (UFLP). Constraints (8.4) require the full amount of every customer's demand to be assigned, to one or more facilities. These are often called *assignment constraints*. Constraints (8.5) prohibit a customer from being assigned to a facility that has not been opened. These are often called *linking constraints*. Constraints (8.6) require the location (x) variables to be binary, and constraints (8.7) require the assignment (y) variables to be nonnegative.

Constraints (8.4) and (8.7) together ensure that $0 \le y_{ij} \le 1$. In fact, it is always optimal to assign each customer solely to its nearest open facility. (Why?) Therefore, there always exists an optimal solution in which $y_{ij} \in \{0, 1\}$ for all $i \in I,\ j \in J$. It is therefore appropriate to think of the y_{ij} as binary variables and to talk about "the facility to which customer i is assigned."

Another way to write constraints (8.5) is

$$\sum_{i \in I} y_{ij} \le |I| x_j \qquad \forall j \in J. \tag{8.8}$$

If $x_j = 1$, then y_{ij} can be 1 for any or all $i \in I$, while if $x_j = 0$, then y_{ij} must be 0 for all i. These constraints are equivalent to (8.5) for the IP. But the LP relaxation is weaker (i.e., it provides a weaker bound) if constraints (8.8) are used instead of (8.5). This is because there are solutions that are feasible for the LP relaxation with (8.8) that are not feasible for the LP relaxation with (8.5). To take a trivial example, suppose there are 2 facilities and 10 customers with equal demand, and suppose each facility serves 5 customers in a given solution. Then it is feasible to set $x_1 = x_2 = \frac{1}{2}$ for the problem with (8.8) but not with (8.5). Since the feasible region for the problem with (8.8) is larger than that for the problem with (8.5), its objective value is no greater. It is important to understand that the IPs have the *same* optimal objective value, but the LPs have different values—one provides a weaker LP bound than the other.

The UFLP is NP-hard (Garey and Johnson 1979). A large number of solution methods have been proposed in the literature over the past several decades, both exact algorithms and heuristics. Some of the earliest exact algorithms involve simply solving the IP using branch-and-bound. Today, this would mean solving (UFLP) as-is using CPLEX, Gurobi, or another off-the-shelf IP solver, although such general-purpose solvers did not exist when the UFLP was first formulated. This approach works quite well using modern solvers, in part because the LP relaxation of (UFLP) is usually extremely tight, and in fact it often results in all-integer solutions "for free" (Morris 1978). (ReVelle and Swain (1970) discuss this property in the context of a related problem, the p-median problem.) Current versions of CPLEX or Gurobi can solve instances of the UFLP with thousands of potential facility sites in a matter of minutes. However, when it was first proposed that branch-and-bound be used to solve the UFLP (by Efroymson and Ray (1966)), IP technology was much less advanced, and this approach could only be used to solve problems of modest size. Therefore, a number of other optimal approaches were developed. Two of these—Lagrangian relaxation and a dual-ascent method called DUALOC—are discussed in Sections 8.2.3 and 8.2.4, respectively. Many other IP techniques, such as Dantzig–Wolfe or Benders decomposition, have also been successfully applied to the UFLP (e.g., Balinski (1965) and Swain (1974)). We discuss heuristic methods for the UFLP in Section 8.2.5.

8.2.3 Lagrangian Relaxation

8.2.3.1 Introduction One of the methods that has proven to be most effective for the UFLP and other location problems is Lagrangian relaxation, a standard technique for integer programming (as well as other types of optimization problems). The basic idea behind Lagrangian relaxation is to remove a set of constraints to create a problem that's easier to solve than the original. But instead of just removing the constraints, we include them in the objective function by adding a term that penalizes solutions for violating the constraints. This process gives a *lower bound* on the optimal objective value of the UFLP, but it does not necessarily give a feasible solution. Feasible solutions must be found using some other method (to be described below); each feasible solution provides an *upper bound* on the optimal objective value. When the upper and lower bounds are close (say, within 1%), we know that the feasible solution we have found is close to optimal.

For more details on Lagrangian relaxation, see Appendix D.1. See also Fisher (1981, 1985) for excellent overviews. Lagrangian relaxation was proposed as a method for solving a UFLP-like problem by Cornuejols et al. (1977).

We want to use Lagrangian relaxation on the UFLP formulation given in Section 8.2.2. The question is, which constraints should we relax? There are only two options: (8.4) and (8.5). (Constraints (8.6) and (8.7) can't be relaxed using Lagrangian relaxation.) Relaxing either (8.4) or (8.5) results in a problem that is quite easy to solve, and both relaxations produce the same bound (for reasons discussed below). But relaxing (8.4) involves relaxing fewer constraints, which is generally preferable (also for reasons that will be discussed below). Therefore, we will relax constraints (8.4), although in Section 8.2.3.8 we will briefly discuss what happens when constraints (8.5) are relaxed.

8.2.3.2 Relaxation We relax constraints (8.4), removing them from the problem and adding a penalty term to the objective function:

$$\sum_{i\in I}\lambda_i\left(1-\sum_{j\in J}y_{ij}\right)$$

The λ_i are called *Lagrange multipliers*. There is one for each relaxed constraint. Their purpose is to ensure that violations in the constraints are penalized by just the right amount—more on this later. We'll use λ to represent the vector of λ_i values.

For now, assume λ is fixed. Relaxing constraints (8.4) gives us the following problem, known as the *Lagrangian subproblem*:

$$\begin{aligned}
(\text{UFLP-LR}_\lambda)\quad \text{minimize}\quad & \sum_{j\in J} f_j x_j + \sum_{i\in I}\sum_{j\in J} h_i c_{ij} y_{ij} + \sum_{i\in I}\lambda_i\left(1-\sum_{j\in J}y_{ij}\right)\\
&= \sum_{j\in J} f_j x_j + \sum_{i\in I}\sum_{j\in J}(h_i c_{ij}-\lambda_i) y_{ij} + \sum_{i\in I}\lambda_i \qquad (8.9)
\end{aligned}$$

$$\begin{aligned}
\text{subject to}\quad & y_{ij}\le x_j && \forall i\in I, \forall j\in J && (8.10)\\
& x_j\in\{0,1\} && \forall j\in J && (8.11)\\
& y_{ij}\ge 0 && \forall i\in I, \forall j\in J && (8.12)
\end{aligned}$$

(The subscript λ on the problem name reminds us that this problem depends on λ as a parameter.) Since the λ_i are all constants, the last term of (8.9) can be ignored during the optimization.

How can we solve this problem? It turns out that the problem is quite easy to solve by inspection—we don't need to use an IP solver or any sort of complicated algorithm.

Suppose that we set $x_j = 1$ for a given facility j. By constraints (8.10), setting $x_j = 1$ allows y_{ij} to be set to 1 for any $i \in I$. For which i would y_{ij} be set to 1 in an optimal solution to the problem? Since this is a minimization problem, y_{ij} would be set to 1 for all i such that $h_i c_{ij} - \lambda_i < 0$. So if x_j were set to 1, the *benefit* (or contribution to the objective function) would be

$$\beta_j = \sum_{i \in I} \min\{0, h_i c_{ij} - \lambda_i\}. \tag{8.13}$$

Now the question is, which x_j should be set to 1? It's optimal to set $x_j = 1$ if and only if $\beta_j + f_j < 0$; that is, if the benefit of opening the facility outweighs its fixed cost. Theorem 8.1 summarizes these conclusions.

Theorem 8.1 *Let*

$$\bar{x}_j = \begin{cases} 1, & \textit{if } \beta_j + f_j < 0 \\ 0, & \textit{otherwise} \end{cases} \tag{8.14}$$

$$\bar{y}_{ij} = \begin{cases} 1, & \textit{if } \bar{x}_j = 1 \textit{ and } h_i c_{ij} - \lambda_i < 0 \\ 0, & \textit{otherwise}. \end{cases} \tag{8.15}$$

Then $(\bar{x}, \bar{y})$ *is an optimal solution for* (UFLP-LR$_\lambda$), *and it has an objective value of*

$$z_{\text{LR}}(\lambda) = \sum_{j \in J} \min\{0, \beta_j + f_j\} + \sum_{i \in I} \lambda_i.$$

Notice that in optimal solutions to (UFLP-LR$_\lambda$), customers may be assigned to 0 or more than 1 facility since the constraints requiring exactly one facility per customer have been relaxed.

Why is this problem so much easier to solve than the original problem? The answer is that (UFLP-LR$_\lambda$) decomposes by j, in the sense that we can focus on each $j \in J$ individually since there are no constraints tying them together. In the original problem, constraints (8.4) tied the js together—we could not make a decision about y_{ij} without also making a decision about y_{ik} since i had to be assigned to exactly one facility.

The method for solving (UFLP-LR$_\lambda$) is summarized in Algorithm 8.1.

8.2.3.3 Lower Bound We've now solved (UFLP-LR$_\lambda$) for given λ_i. How does this help us? Well, from Theorem D.1, we know that, for any λ, the optimal objective value of (UFLP-LR$_\lambda$) is a lower bound on the optimal objective value for the original problem:

$$z_{\text{LR}}(\lambda) \le z^*. \tag{8.16}$$

The point of Lagrangian relaxation is not to generate feasible solutions, since the solutions to (UFLP-LR$_\lambda$) will generally be infeasible for (UFLP). Instead, the point is to generate good (i.e., high) lower bounds in order to prove that a feasible solution we've found some other way is good. For example, if we've found a feasible solution for the UFLP (using any method at all) whose objective value is 1005 and we've also found a λ so that $z_{\text{LR}}(\lambda) = 1000$, then we know our solution is no more than $(1005 - 1000)/1000$ = 0.5% away from optimal. (It may in fact be *exactly* optimal, but given these two bounds, we can only say it's within 0.5%.)

Algorithm 8.1 Solve (UFLP-LR_λ)

```
1: input Lagrange multipliers $\lambda$
2: for all $j \in J$ do                                              ▷ Main loop
3:     $\beta_j \leftarrow \sum_{i \in I} \min\{0, h_i c_{ij} - \lambda_i\}$                 ▷ Calculate benefit
4:     if $\beta_j + f_j < 0$ then                                  ▷ Check benefit vs. fixed cost
5:         $\bar{x}_j \leftarrow 1$                                              ▷ Open $j$
6:         for all $i \in I$ do
7:             if $h_i c_{ij} - \lambda_i < 0$ then $\bar{y}_{ij} \leftarrow 1$ else $\bar{y}_{ij} \leftarrow 0$ end if
8:         end for
9:     else
10:        $\bar{x}_j \leftarrow 0$                                              ▷ Do not open $j$
11:        for all $i \in I$ do
12:            $\bar{y}_{ij} \leftarrow 0$
13:        end for
14:    end if
15: end for
16: $z_{\text{LR}}(\lambda) \leftarrow \sum_{j \in J} \min\{0, \beta_j + f_j\} + \sum_{i \in I} \lambda_i$       ▷ Calculate objective function
17: return $\bar{x}$, $\bar{y}$, $z_{\text{LR}}(\lambda)$
```

Now, if we pick λ at random, we're not likely to get a particularly good bound—that is, $z_{\text{LR}}(\lambda)$ won't be close to z^*. We have to choose λ cleverly so that we get the best possible bound—so that $z_{\text{LR}}(\lambda)$ is as large as possible. That is, we want to solve problem (LR) given in (D.8), which, for the UFLP, can be written as follows:

$$\text{(LR)} \qquad \max_{\lambda} \left\{ \begin{array}{ll} \min_{x,y} & \sum_{j \in J} f_j x_j + \sum_{i \in I} (h_i c_{ij} - \lambda_i) y_{ij} + \sum_{i \in I} \lambda_i \\ \text{s.t.} & y_{ij} \le x_j \quad \forall i \in I, \forall j \in J \\ & x_j \in \{0,1\} \quad \forall j \in J \\ & y_{ij} \ge 0 \quad \forall i \in I, \forall j \in J \end{array} \right\} \tag{8.17}$$

We'll talk more later about how to solve this problem. For now, let's assume we know the optimal λ^* and that the optimal objective value is $z_{\text{LR}} \equiv z_{\text{LR}}(\lambda^*)$. How large can z_{LR} be? Theorem D.1 tells us it cannot be larger than z^*, but how close can it get? The answer turns out to be related to the LP relaxation of the problem. From Theorem D.2, we have

$$z_{\text{LP}} \le z_{\text{LR}}, \tag{8.18}$$

where z_{LP} is the optimal objective value of the LP relaxation of (UFLP) and z_{LR} is the optimal objective value of (LR).

Combining (8.16) and (8.18), we now know that

$$z_{\text{LP}} \le z_{\text{LR}} \le z^*. \tag{8.19}$$

For most problems, $z_{\text{LP}} \lneqq z^*$, so where in the gap does z_{LR} fall? An IP is said to have the *integrality property* if its LP relaxation naturally has an all-integer optimal solution. You should be able to convince yourself that (UFLP-LR_λ) has the integrality property for all λ since it is never better to set x and y to fractional values. Therefore, the following is a corollary to Lemma D.3:

Corollary 8.2 *For the UFLP,* $z_{\mathrm{LP}} = z_{\mathrm{LR}}$.

Combining (8.19) and Corollary 8.2, we have

$$z_{\mathrm{LR}} = z_{\mathrm{LP}} \leq z^*.$$

This means that if the LP relaxation bound from the UFLP is not very tight, the Lagrangian relaxation bound won't be very tight either. Fortunately, as noted in Section 8.2.2, the UFLP tends to have very tight LP relaxation bounds. This raises the question of why we'd want to use Lagrangian relaxation at all since the LP bound is just as tight.

There are several possible answers to this question. The first is that when Lagrangian relaxation was first applied to the UFLP, computer implementations of the simplex method were quite inefficient, and even the LP relaxation of the UFLP could take a long time to solve, whereas the Lagrangian subproblem could be solved quite quickly. Recent implementations of the simplex method, however (for example, recent versions of CPLEX), are much more efficient and are able to solve reasonably large instances of the UFLP—LP or IP—pretty quickly. Nevertheless, Lagrangian relaxation is still an important tool for solving the UFLP. One advantage of this method is that it can often be modified to solve extensions of the UFLP that IP solvers can't solve—for example, nonlinear, nonconvex problems like the location model with risk pooling (LMRP), which we discuss in Section 12.2.

It is important to distinguish between z_{LR} (the best possible lower bound achievable by Lagrangian relaxation) and $z_{\mathrm{LR}}(\lambda)$ (the lower bound achieved at a given iteration of the procedure). At any given iteration, we have

$$z_{\mathrm{LR}}(\lambda) \leq z_{\mathrm{LR}} = z_{\mathrm{LP}} \leq z^* \leq z(x, y), \tag{8.20}$$

where $z_{\mathrm{LR}}(\lambda)$ is the objective value of the Lagrangian subproblem for the particular λ at the current iteration, and $z(x, y)$ is the objective value of the particular feasible solution (x, y) found at the current iteration.

8.2.3.4 Upper Bound

Now that we've obtained a lower bound on the optimal objective of (UFLP) using (UFLP-LR$_\lambda$), we need to find an upper bound. Upper bounds come from feasible solutions to (UFLP). How can we build good feasible solutions? One way would be using construction and/or improvement heuristics like those described in Section 8.2.5. But we'd like to take advantage of the information contained in the solutions to (UFLP-LR$_\lambda$); that is, we'd like to convert a solution to (UFLP-LR$_\lambda$) into one for (UFLP). Remember that solutions to (UFLP-LR$_\lambda$) consist of a set of facility locations (identified by the x variables) and a set of assignments (identified by the y variables). It is the y variables that make the solution infeasible for (UFLP), since customers might be assigned to 0 or more than 1 facility. (If every customer happens to be assigned to exactly 1 facility, the solution is also feasible for (UFLP). In fact, it is *optimal* for (UFLP) since it has the same objective value for both (UFLP-LR$_\lambda$), which provides a lower bound, and (UFLP), which provides an upper bound. But we can't expect this to happen in general.)

Generating a feasible solution for (UFLP) is easy: We just open the facilities that are open in the solution to (UFLP-LR$_\lambda$) and then assign each customer to its nearest open facility. (See Algorithm 8.2.) The resulting solution is feasible and provides an upper bound on the optimal objective value of (UFLP). Sometimes an improvement heuristic (like the swap or neighborhood search heuristics discussed in Section 8.3.2.3) is applied to each feasible solution found, but this is optional.

Algorithm 8.2 Get feasible solution for UFLP from solution to (UFLP-LR$_\lambda$)

```
1:  input location vector $\bar{x}$ for (UFLP-LR$_\lambda$)
2:  $x \leftarrow \bar{x}$                                   ▷ Open facilities in lower bound solution
3:  for all $i \in I$ do                                      ▷ Main loop
4:      $j^* \leftarrow \text{argmin}_{\bar{x}_j=1}\{c_{ij}\}$          ▷ Find nearest open facility to $i$
5:      $y_{ij^*} \leftarrow 1$                                ▷ Assign $i$ to $j^*$
6:      for all $j \in J, j \neq j^*$ do
7:          $y_{ij} \leftarrow 0$
8:      end for
9:  end for
10: $z(x,y) \leftarrow \sum_{j\in J} f_j x_j + \sum_{i\in I}\sum_{j\in J} h_i c_{ij} y_{ij}$   ▷ Calculate obj. function
11: return $x$, $y$, $z(x,y)$
```

8.2.3.5 Updating the Multipliers

Each λ gives a single lower bound and (using the method in Section 8.2.3.4) a single upper bound. The Lagrangian relaxation process involves many iterations, each using a different value of λ, in the hopes of tightening the bounds. It would be impractical to try every possible value of λ; we want to choose λ cleverly.

Using the logic of Section D.1.3, if λ_i is too small, there's no real incentive to set the y_{ij} variables to 1 since the penalty will be small. On the other hand, if λ_i is too large, there will be an incentive to set *lots* of y_{ij} variables to 1, making the term inside the parentheses negative and the overall penalty large and negative. (Remember that (UFLP-LR$_\lambda$) is a minimization problem.) By changing λ_i, we'll encourage fewer or more y_{ij} variables to be 1.

So:

- If $\sum_{j\in J} y_{ij} = 0$, then λ_i is too small; it should be increased.
- If $\sum_{j\in J} y_{ij} > 1$, then λ_i is too large; it should be decreased.
- If $\sum_{j\in J} y_{ij} = 1$, then λ_i is just right; it should not be changed.

Here's another way to see the effect of changing λ_i. Remember that if $x_j = 1$ in the solution to (UFLP-LR$_\lambda$), y_{ij} will be set to 1 if

$$h_i c_{ij} - \lambda_i < 0.$$

Increasing λ_i makes this hold for more facilities j, while decreasing it makes it hold for fewer.

There are several ways to make these adjustments to λ. Perhaps the most common is *subgradient optimization*, discussed in Section D.1.3. For the UFLP, the step size at iteration t (denoted Δ^t) is given by

$$\Delta^t = \frac{\alpha^t(\text{UB} - z_{\text{LR}}(\lambda^t))}{\sum_{i\in I}\left(1 - \sum_{j\in J} y_{ij}\right)^2}, \tag{8.21}$$

where $z_{\text{LR}}(\lambda^t)$ is the lower bound found at iteration t, UB is the best upper bound found (i.e., the objective value of the best feasible solution found so far), and α^t is a constant that is generally set to 2 at iteration 1 and divided by 2 after a given number (say 20) of consecutive iterations have passed during which the best known lower bound has not improved. The step direction for iteration i is simply given by

$$1 - \sum_{j \in J} y_{ij}$$

(the violation in the constraint).

To obtain the new multipliers (call them λ^{t+1}) from the old ones (λ^t), we set

$$\lambda_i^{t+1} = \lambda_i^t + \Delta^t \left(1 - \sum_{j \in J} y_{ij}\right). \tag{8.22}$$

Note that since $\Delta^t > 0$, this follows the rules given above: If $\sum_{j \in J} y_{ij} = 0$, then λ_i increases; if $\sum_{j \in J} y_{ij} > 1$, then λ_i decreases; and if $\sum_{j \in J} y_{ij} = 1$, then λ_i stays the same.

The process of solving (UFLP-LR$_\lambda$), finding a feasible solution, and updating λ is continued until some of criteria are met. (See Section D.1.4.)

At the first iteration, λ can be initialized using a variety of ways: For example, set $\lambda_i = 0$ for all i, set it to some random number, or set it according to some other ad-hoc rule.

If the Lagrangian procedure stops before the upper and lower bounds are sufficiently close to each other, we can use branch-and-bound to close the optimality gap; see Section D.1.6. The Lagrangian procedure is summarized in Section D.1.7.

8.2.3.6 Summary

The Lagrangian relaxation method for the UFLP is summarized in the pseudocode in Algorithm 8.3. In the pseudocode, $(\bar{x}, \bar{y})$ represents an optimal solution to (UFLP-LR$_\lambda$) , (x, y) represents a feasible solution to (UFLP), and $(x^{\text{UB}}, y^{\text{UB}})$ represents the current best solution for (UFLP). Note that in step 29, other termination criteria can be used, instead or in addition.

☐ **EXAMPLE 8.1**

The instance pictured in Figure 8.1 is the 88-node instance from Daskin (1995). It consists of the capitals of the 48 continental United States, plus Washington, DC, plus the 50 largest cities in the 1990 US census, minus duplicates. In this instance, $I = J$: Every node is both a customer and a potential facility site. Demands h_i are set equal to the city populations divided by 1000; fixed costs f_j are set equal to the median home value; and transportation costs c_{ij} are set equal to 0.5 times the great circle distance between i and j. (The full data set, along with other related data sets, are available on the book's companion web site.)

The optimal solution locates five facilities, in Houston, TX; Philadelphia, PA; Detroit, MI; Fresno, CA; and Topeka, KS. The total cost of this solution is \$783,813, with fixed and transportation costs of \$521,713 and \$262,100, respectively. We obtained this solution using the Lagrangian relaxation algorithm discussed in this section, implemented in MATLAB, with a total CPU time of less than 2 seconds on a laptop computer.

In case you're curious: The 9-facility solution shown in Figure 8.1(a) has a total cost of \$1,480,059 (\$954,600 fixed cost plus \$525,459 transportation cost), while the

Algorithm 8.3 Lagrangian relaxation algorithm for UFLP

```
1:  input initial multipliers $\lambda^1$, initial constant $\alpha^0$, $\alpha$-halving constant $\gamma$, optimality toler-
    ance $\kappa$, iteration limit $\zeta$
2:  $t \leftarrow 1$, LB $\leftarrow -\infty$, UB $\leftarrow \infty$, NonImprCtr $\leftarrow 0$          ▷ Initialization
3:  repeat                                                          ▷ Main loop
4:      solve (UFLP-LR$_\lambda$) using Algorithm 8.1 with input $\lambda^t$          ▷ Lower bound
5:      $(\bar{x}, \bar{y})$, $z_{\text{LR}}(\lambda^t) \leftarrow$ output of Algorithm 8.1
6:      if $z_{\text{LR}}(\lambda^t) > \text{LB}$ then                  ▷ Compare to best-known lower bound
7:          LB $\leftarrow z_{\text{LR}}(\lambda^t)$
8:          NonImprCtr $\leftarrow 0$                          ▷ Reset non-improvement counter
9:      else
10:         NonImprCtr $\leftarrow$ NonImprCtr $+ 1$               ▷ Increment non-impr. counter
11:         if NonImprCtr $= \gamma$ then                        ▷ Check whether to halve $\alpha$
12:             $\alpha^t \leftarrow \alpha^{t-1}/2$
13:             NonImprCtr $\leftarrow 0$
14:         else
15:             $\alpha^t \leftarrow \alpha^{t-1}$
16:         end if
17:     end if
18:     get feasible solution from Algorithm 8.2 with input $\bar{x}$          ▷ Upper bound
19:     $x$, $y$, $z(x,y) \leftarrow$ output of Algorithm 8.2
20:     if $z(x,y) < \text{UB}$ then                        ▷ Compare to best-known upper bound
21:         UB $\leftarrow z(x,y)$
22:         $(x^{\text{UB}}, y^{\text{UB}}) \leftarrow (x,y)$
23:     end if
24:     $\Delta^t \leftarrow \alpha^t(\text{UB} - z_{\text{LR}}(\lambda^t)) / \sum_{i\in I} \left(1 - \sum_{j\in J} \bar{y}_{ij}\right)^2$     ▷ Update multipliers
25:     for all $i \in I$ do
26:         $\lambda_i^{t+1} \leftarrow \lambda_i^t + \Delta^t \left(1 - \sum_{j\in J} \bar{y}_{ij}\right)$
27:     end for
28:     $t \leftarrow t + 1$                                              ▷ Increment $t$
29: until UB $- z_{\text{LR}}(\lambda^t) \le \kappa$ or $t > \zeta$                 ▷ Check for termination
30: return $x^{\text{UB}}$, $y^{\text{UB}}$, UB
```

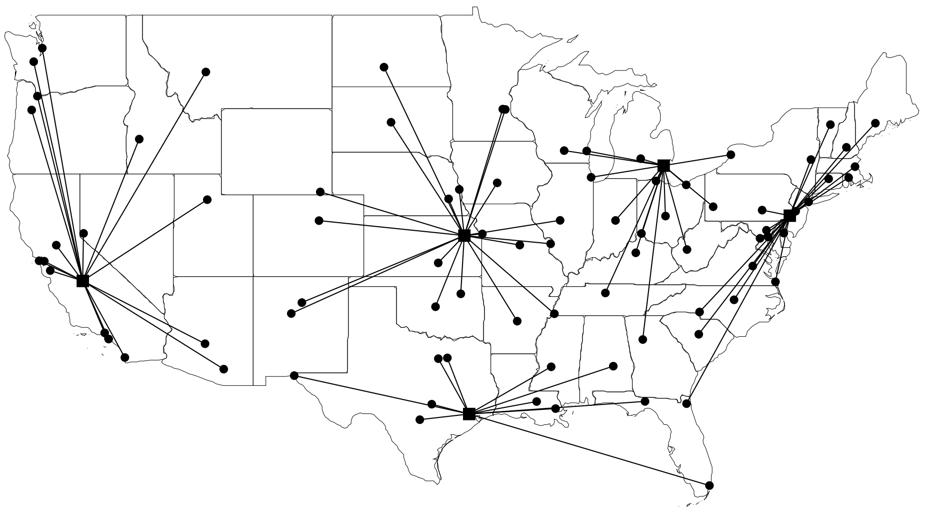


Figure 8.2 Optimal solution to 88-node UFLP instance. Total cost = \$783,813.

3-facility solution in Figure 8.1(b) has a total cost of \$1,238,911 (\$512,800 fixed cost plus \$726,111 transportation cost).

□

8.2.3.7 Variable Fixing

Sometimes the Lagrangian relaxation procedure terminates with the lower and upper bounds farther apart than we'd like. Before executing branch-and-bound to close the gap, we may be able to fix some of the x_j variables to 0 or 1 based on the facility benefits and the current bounds. The variables can be fixed permanently, throughout the entire branch-and-bound tree. The more variables we can fix, the faster the branch-and-bound procedure is likely to run. Essentially, the method works by "peeking" down a branch of the tree and running a quick check to determine whether the next node down the branch would be fathomed.

Theorem 8.3 *Let UB be the best upper bound found during the Lagrangian procedure, let λ be a given set of Lagrange multipliers that were used during the procedure, let β_j be the facility benefits* (8.13) *under λ, and let $z_{\text{LR}}(\lambda)$ be the lower bound (the optimal objective value of* (UFLP-LR$_\lambda$)*) under λ. If $x_j = 0$ in the solution to* (UFLP-LR$_\lambda$) *and*

$$z_{\text{LR}}(\lambda) + \beta_j + f_j > \text{UB}, \tag{8.23}$$

then $x_j = 0$ in every optimal solution to (UFLP). *If $x_j = 1$ in the solution to* (UFLP-LR$_\lambda$) *and*

$$z_{\text{LR}}(\lambda) - (\beta_j + f_j) > \text{UB}, \tag{8.24}$$

then $x_j = 1$ in every optimal solution to (UFLP).

Proof. Suppose we were to branch on x_j, setting $x_j = 0$ for one child node and $x_j = 1$ for the other, and suppose we use λ as the initial multipliers for the Lagrangian procedure at each child node.

At the "$x_j = 1$" node, the same facilities would be open as in the root-node solution, except that now facility j is also open. The cost of this solution for (UFLP-LR$_\lambda$) is the cost of the original solution, $z_{\text{LR}}(\lambda)$, plus $\beta_j + f_j$. Therefore, we would obtain $z_{\text{LR}}(\lambda) + \beta_j + f_j$ as a lower bound at this node. Since this lower bound is greater than the best-known upper bound, we would fathom the tree at this node, and the optimal solution would be contained in the other half of the tree—the "$x_j = 0$" half.

A similar argument applies to the second case. At the "$x_j = 0$" node, we obtain a lower bound of $z_{\text{LR}}(\lambda) - (\beta_j + f_j)$, and if this is greater than UB, we fathom the tree at this node. ■

Note that, in the second part of the theorem, if $x_j = 1$ then, by (8.14), $\beta_j + f_j < 0$, which is why the left-hand side of (8.24) might be greater than UB.

This trick has been applied successfully to a variety of facility location problems; see, e.g., Daskin et al. (2002) and Snyder and Daskin (2005). Typically, the conditions in Theorem 8.3 are checked twice after processing has terminated at the root node, once using the most recent multipliers λ and once using the multipliers that produced the best-known lower bound. The time required to check these conditions for every j is negligible.

8.2.3.8 Alternate Relaxation As stated above, we could have chosen instead to relax constraints (8.5). In this case, the Lagrangian subproblem becomes

$$\begin{aligned}
(\text{UFLP-LR}_\lambda) \quad \text{minimize} \quad & \sum_{j\in J} f_j x_j + \sum_{i\in I}\sum_{j\in J} h_i c_{ij} y_{ij} + \sum_{i\in I}\sum_{j\in J} \lambda_{ij}(x_j - y_{ij}) \\
= & \sum_{j\in J}\left(\sum_{i\in I}\lambda_{ij} + f_j\right) x_j + \sum_{i\in I}\sum_{j\in J}(h_i c_{ij} - \lambda_{ij}) y_{ij} \qquad (8.25) \\
\text{subject to} \quad & \sum_{j\in J} y_{ij} = 1 \qquad \forall i \in I \qquad (8.26) \\
& x_j \in \{0,1\} \qquad \forall j \in J \qquad (8.27) \\
& y_{ij} \ge 0 \qquad \forall i \in I, \forall j \in J \qquad (8.28)
\end{aligned}$$

Now every customer must be assigned to a single facility, but that facility need not be open. There are no constraints linking the x and y variables, so the problem can be written as two separate problems:

$$\begin{aligned}
(x\text{-problem}) \quad \text{minimize} \quad & \sum_{j\in J}\left(\sum_{i\in I}\lambda_{ij} + f_j\right) x_j \qquad (8.29) \\
\text{subject to} \quad & x_j \in \{0,1\} \qquad \forall j \in J \qquad (8.30)
\end{aligned}$$

$$\begin{aligned}
(y\text{-problem}) \quad \text{minimize} \quad & \sum_{i\in I}\sum_{j\in J}(h_i c_{ij} - \lambda_{ij}) y_{ij} \qquad (8.31) \\
\text{subject to} \quad & \sum_{j\in J} y_{ij} = 1 \qquad \forall i \in I \qquad (8.32) \\
& y_{ij} \ge 0 \qquad \forall i \in I, \forall j \in J \qquad (8.33)
\end{aligned}$$

To solve the x-problem, we simply set $x_j = 1$ for all j such that $\sum_{i\in I} \lambda_{ij} + f_j < 0$. (Note that since the constraints relaxed are $\le$ constraints, $\lambda \le 0$; see Section D.1.5.1.) To solve the y-problem, for each i, we set $y_{ij} = 1$ for the j that minimizes $h_i c_{ij} - \lambda_{ij}$. The rest of the procedure is similar, except that the step-size calculation becomes

$$\Delta^t = \frac{\alpha^t(\text{UB} - z_{\text{LR}}(\lambda^t))}{\sum_{i\in I}\sum_{j\in J}(x_j - y_{ij})^2} \tag{8.34}$$

and the multiplier-updating formula becomes

$$\lambda_{ij}^{t+1} = \lambda_{ij}^t + \Delta^t(x_j - y_{ij}). \tag{8.35}$$

In practice, relaxing the assignment constraints (8.4) tends to work better than relaxing the linking constraints (8.5). One reason for this is that the former relaxation involves relaxing fewer constraints, which generally makes it easier to find good multipliers using subgradient optimization. Another reason is that since y_{ij} will be 0 for many j that are open, there will be many constraints such that $y_{ij} < x_j$. It is often difficult to get good results when relaxing inequality constraints if many of them have slack.

8.2.4 The DUALOC Algorithm

The DUALOC algorithm was proposed by Erlenkotter (1978). It is a *dual-ascent* or *primal–dual* algorithm that constructs good feasible solutions for the dual of the LP relaxation of (UFLP) and then uses these to develop good (often optimal) integer solutions for the primal, i.e., for (UFLP) itself.

We form the LP relaxation of (UFLP), denoted (UFLP-P), by replacing constraints (8.6) with

$$x_j \ge 0 \qquad \forall j \in J. \tag{8.36}$$

Let v_i and w_{ij} be the dual variables for constraints (8.4) and (8.5), respectively. In addition, for notational convenience, let $\hat{c}_{ij} \equiv h_i c_{ij}$. Then the LP dual is given by

$$\text{(UFLP-D)} \quad \text{maximize} \quad \sum_{i\in I} v_i \tag{8.37}$$

$$\text{subject to} \quad \sum_{i\in I} w_{ij} \le f_j \qquad \forall j \in J \tag{8.38}$$

$$v_i - w_{ij} \le \hat{c}_{ij} \qquad \forall i \in I, \forall j \in J \tag{8.39}$$

$$w_{ij} \ge 0 \qquad \forall i \in I, \forall j \in J \tag{8.40}$$

The complementary slackness conditions for (UFLP-P) and (UFLP-D) are given by

$$x_j^* \left(f_j - \sum_{i\in I} w_{ij}^* \right) = 0 \tag{8.41}$$

$$y_{ij}^* \left[\hat{c}_{ij} - (v_i^* - w_{ij}^*) \right] = 0 \tag{8.42}$$

$$v_i^* \left(1 - \sum_{j\in J} y_{ij}^* \right) = 0 \tag{8.43}$$

$$w_{ij}^*\left(y_{ij}^* - x_j^*\right) = 0 \tag{8.44}$$

Suppose we are given arbitrary values of the variables v_i. Then either it is feasible to set w_{ij} as small as possible, i.e.,

$$w_{ij} = \max\{0, v_i - \hat{c}_{ij}\} \tag{8.45}$$

for all i and j, or there are *no* feasible w_{ij} values (for the given v_i values). Moreover, since w_{ij} does not appear in the objective function, any feasible w_{ij} (for fixed v_i) is acceptable. Thus, we assume that (8.45) holds and substitute this relationship into (UFLP-D). Constraints (8.39) and (8.40) are automatically satisfied when (8.45) holds; therefore, we obtain the following *condensed dual*, which uses only v_i and not w_{ij}:

$$\text{(UFLP-CD)} \quad \text{maximize} \quad \sum_{i\in I} v_i \tag{8.46}$$

$$\text{subject to} \quad \sum_{i\in I} \max\{0, v_i - \hat{c}_{ij}\} \le f_j \qquad \forall j \in J \tag{8.47}$$

Substituting (8.45) into the complementary slackness conditions (8.41)–(8.44), we obtain

$$x_j^*\left(f_j - \sum_{i\in I}\max\{0, v_i^* - \hat{c}_{ij}\}\right) = 0 \tag{8.48}$$

$$y_{ij}^*\left[\hat{c}_{ij} - (v_i^* - \max\{0, v_i^* - \hat{c}_{ij}\})\right] = 0 \tag{8.49}$$

$$v_i^*\left(1 - \sum_{j\in J} y_{ij}^*\right) = 0 \tag{8.50}$$

$$\max\{0, v_i^* - \hat{c}_{ij}\}\left(y_{ij}^* - x_j^*\right) = 0 \tag{8.51}$$

Note that (UFLP-CD) is not an LP, since the $\max\{\cdot\}$ function is nonlinear. One could develop a customized simplex-type algorithm to solve it—an approach like this is proposed by Schrage (1978) for the p-median problem, among others—but instead, the DUALOC approach exploits the structure of (UFLP-CD) to find near-optimal solutions directly.

The DUALOC algorithm consists of two main procedures. The first is a *dual-ascent procedure* that generates feasible dual solutions for (UFLP-CD) and corresponding primal integer solutions for (UFLP). The second is a *dual-adjustment procedure* that attempts to reduce complementary slackness violations (thereby improving the primal or dual solutions, or both) by adjusting the dual solution iteratively and calling the dual-ascent procedure as a subroutine. If these procedures terminate without an optimal integer solution to (UFLP), branch-and-bound is used to close the optimality gap.

8.2.4.1 Primal–Dual Relationships

The dual-ascent procedure generates both a dual solution v^+ for (UFLP-CD) and a set $J^+ \subseteq J$ of facility locations such that the following properties hold:

- *Primal–Dual Property 1* (PDP1): $\sum_{i\in I}\max\{0, v_i^+ - \hat{c}_{ij}\} = f_j$ for all $j \in J^+$
- *Primal–Dual Property 2* (PDP2): For each $i \in I$, there exists at least one $j \in J^+$ such that $\hat{c}_{ij} \le v_i^+$

Such a solution can easily be converted to an integer primal solution: The set J^+ provides the x variables for (UFLP), and, as in the Lagrangian relaxation procedure (Section 8.2.3.4), the y variables can be set by assigning each customer to its nearest open facility. That is, an integer primal solution for (UFLP) can be obtained from J^+ as follows:

$$x_j^+ = \begin{cases} 1, & \text{if } j \in J^+ \\ 0, & \text{otherwise} \end{cases} \tag{8.52}$$

$$y_{ij}^+ = \begin{cases} 1, & \text{if } j = j^+(i) \\ 0, & \text{otherwise,} \end{cases} \tag{8.53}$$

where $j^+(i) \equiv \text{argmin}_{k \in J^+}\{\hat{c}_{ik}\}$.

The primal–dual solution (x^+, y^+, v^+) satisfies three of the four complementary slackness conditions: (8.48) is satisfied because of PDP1, and (8.50) is satisfied because each i is assigned to exactly one j in (8.53). To see why (8.49) is satisfied, suppose $y_{ij}^+ = 1$, i.e., $j = j^+(i)$. By PDP2, $\hat{c}_{ij} \le v_i^+$ for some $j \in J^+$ and $\hat{c}_{i,j^+(i)} \le \hat{c}_{ij}$ by the definition of $j^+(i)$, so

$$\begin{aligned} & y_{i,j^+(i)}^+ \left[\hat{c}_{i,j^+(i)} - \left(v_i^+ - \max\{0, v_i^+ - \hat{c}_{i,j^+(i)}\}\right)\right] \\ =& \hat{c}_{i,j^+(i)} - \left(v_i^+ - (v_i^+ - \hat{c}_{i,j^+(i)})\right) \\ =& 0. \end{aligned}$$

Thus, (x^+, y^+) and v^+ are optimal for (UFLP-P) and (UFLP-CD), respectively, if and only if (8.51) holds. Moreover, since (x^+, y^+) is integer, if it is optimal for (UFLP-P), then it is also optimal for (UFLP). (It may seem strange to hope that the integer solution (x^+, y^+) is optimal for the LP relaxation (UFLP-P). But remember that (UFLP-P) often has all-integer solutions "for free" (see page 272), and is usually very tight when it is not all-integer so that good integer solutions to (UFLP-P) are likely to be good also for (UFLP).)

Condition (8.51) may be violated when $\hat{c}_{ij} < v_i^+$ for some $j \ne j^+(i)$, since in that case $y_{ij}^+ = 0$ but $x_j^+ = 1$. This suggests that complementary slackness violations can be reduced by focusing on the $v_i^+ - \hat{c}_{ij}$ terms for $j \ne j^+(i)$, and indeed those terms directly affect the duality gap, as the next lemma attests.

Lemma 8.4 *Let z_P^+ be the objective function value of (UFLP-P) under the solution (x^+, y^+), and let z_D^+ be the objective function value of (UFLP-CD) under the solution v^+. Then*

$$z_P^+ - z_D^+ = \sum_{i \in I} \sum_{\substack{j \in J^+ \\ j \ne j^+(i)}} \max\{0, v_i^+ - \hat{c}_{ij}\}.$$

Proof. Omitted; see Problem 8.40. ■

The dual-ascent procedure (Section 8.2.4.2) generates v^+ and J^+. The dual-adjustment procedure (Section 8.2.4.3) then attempts to improve the solutions by reducing $v_i^+ - \hat{c}_{ij}$ terms for $j \ne j^+(i)$.

8.2.4.2 The Dual-Ascent Procedure

The dual-ascent procedure constructs a dual solution v^+ and a facility set J^+ such that properties PDP1 and PDP2 hold for v^+ and J^+. The procedure begins by constructing an easy feasible solution in which the v_i variables are small (in order to ensure feasibility with respect to (8.47)) and then increasing the v_i one by one (in order to improve the objective (8.46)).

For each $i \in I$, sort the costs $\hat{c}_{ij}$ in nondecreasing order and let $\hat{c}_i^k$ be the kth of these costs, for $k = 1, \ldots, |J|$. Define $\hat{c}_i^{|J|+1} \equiv \infty$. Then an initial solution can be generated by setting $v_i = \hat{c}_i^1$ for all $i \in I$; this solution is feasible for (UFLP-CD). (Why?) Actually, any initial feasible solution will work, but this one is easy to obtain.

The dual-ascent procedure is given in Algorithm 8.4. In line 1, we initialize v_i to $\hat{c}_i^1$ and initialize the index k_i to 2. Throughout the algorithm, k_i equals the smallest k such that $v_i \lneqq \hat{c}_i^k$; as the v_i increase in the algorithm, so do the k_i. In line 2, s_j represents the slack in constraint (8.47) for facility j; since v_i equals the smallest $\hat{c}_{ij}$, $s_j = f_j - \sum_{i \in I} \max\{0, v_i - \hat{c}_{ij}\} = f_j$. The algorithm loops through the customers; for each customer i, we would like to set v_i to the next larger value of $\hat{c}_{ij}$, i.e., to $\hat{c}_i^{k_i}$. However, increasing v_i increases the left-hand side of (8.47) for all j such that $v_i - \hat{c}_{ij} \geq 0$. (These j are the facilities whose costs are $\hat{c}_i^1, \ldots, \hat{c}_i^{k_i-1}$.) Therefore, line 6 calculates the largest allowable increase in v_i without violating (8.47) for any j. Note that we only consider j such that $v_i - \hat{c}_{ij} \geq 0$; for the other j, the left-hand sides of (8.47) will not increase because we will not increase v_i past $\hat{c}_i^{k_i}$, as enforced by lines 7–8. Lines (9)–(10) update the `IMPROVED` flag and the index k_i. (The `IMPROVED` flag is only set to `TRUE` if we were able to increase v_i all the way to $\hat{c}_i^{k_i}$ for some i, not for smaller increases.) Lines 12–14 adjust the slack for all facilities whose left-hand sides of (8.47) will change, and line 15 performs the update to v_i. The process repeats until v_i cannot be increased to $\hat{c}_i^{k_i}$ for any customer. Line 18 sets v^+ equal to the final value of v and builds the set J^+, and the algorithm returns both these values.

Algorithm 8.4 Dual-ascent procedure for DUALOC algorithm

```
1:  $v_i \leftarrow \hat{c}_i^1$, $k_i \leftarrow 2 \; \forall i \in I$                     ▷ Initialization
2:  $s_j \leftarrow f_j \; \forall j \in J$
3:  repeat                                                   ▷ Improvement
4:      IMPROVED ← FALSE
5:      for all $i \in I$ do
6:          $\Delta_i \leftarrow \min_{j \in J: v_i - \hat{c}_{ij} \geq 0}\{s_j\}$      ▷ Calculate allowable increase in $v_i$
7:          if $\Delta_i \geq \hat{c}_i^{k_i} - v_i$ then                ▷ Did we get all the way to $\hat{c}_i^{k_i}$?
8:              $\Delta_i \leftarrow \hat{c}_i^{k_i} - v_i$
9:              IMPROVED ← TRUE
10:             $k_i \leftarrow k_i + 1$
11:         end if
12:         for all $j \in J$ s.t. $v_i - \hat{c}_{ij} \geq 0$ do
13:             $s_j \leftarrow s_j - \Delta_i$                            ▷ Adjust slack
14:         end for
15:         $v_i \leftarrow v_i + \Delta_i$                                ▷ Adjust $v_i$
16:     end for
17: until not IMPROVED                                       ▷ Stop when no improvement
18: $v^+ \leftarrow v$, $J^+ \leftarrow \{j \in J | s_j = 0\}$                  ▷ Build solutions to return
19: return $v^+$, $J^+$
```

Proposition 8.5 *The v^+ and J^+ returned by Algorithm 8.4 satisfy PDP1 and PDP2.*

Proof. PDP1: It suffices to show that, throughout the progression of the algorithm, $s_j = f_j - \sum_{i \in I} \max\{0, v_i - \hat{c}_{ij}\}$. (We use v to refer to the values set during the course of the algorithm, and v^+ to refer to the final values returned by the algorithm.) Clearly, this holds after line 2. In the main loop, each time v_i increases by Δ_i for any i, then either $v_i - \hat{c}_{ij} \geq 0$, in which case we reduce s_j by Δ_i; or $v_i - \hat{c}_{ij} < 0$, in which case we increase v_i to at most $\hat{c}_{ij}$ (in line 8), so $f_j - \max\{0, v_i - \hat{c}_{ij}\}$ does not change, and neither does s_j. In other words, at the end of each iteration through the main loop, $s_j = f_j - \sum_{i \in I} \max\{0, v_i - \hat{c}_{ij}\}$.

PDP2: Suppose, for a contradiction, that there exists an $i \in I$ such that $\hat{c}_{ij} > v_i^+$ for all $j \in J^+$. This means that $s_j > 0$ for all j such that $v_i^+ - \hat{c}_{ij} \geq 0$. Then at line 6, Δ_i would have been set to a positive number, and at line 15, v_i would have been increased by Δ_i. This contradicts our assumption that v^+ is the solution returned by the algorithm. ■

If there is a strict subset of J^+ that still satisfies PDP1 and PDP2, it is better to use that subset. To see why, suppose there is a facility j' with $s_{j'} = 0$ that is not included in J^+. PDP1 does not prohibit this situation; it prohibits the converse. Would it be better to add j' to J^+? Lemma 8.4 suggests the answer is no: For each $i \in I$, either $\hat{c}_{ij'} < \hat{c}_{i,j^+(i)}$ (so j' becomes the new closest facility to i), in which case z_P^+ increases by $v_i^+ - \hat{c}_{i,j^+(i)} > 0$; or $\hat{c}_{ij'} \geq \hat{c}_{i,j^+(i)}$, in which case z_P^+ increases by $\max\{0, v_i - \hat{c}_{ij'}\} \geq 0$. Therefore, we want J^+ to be *minimal* in the sense that no facility can be removed from it without violating PDP2. Of course, finding a minimal J^+ is itself a combinatorial problem. Erlenkotter (1978) suggests a simple heuristic for finding such a set, but to keep things simple, we'll just assume that J^+ contains *all* j for which $s_j = 0$.

You might be wondering why we limit Δ_i to $\hat{c}_i^{k_i} - v_i$ in line 8, since we want v_i to be as large as possible, and we can leave Δ_i at the value set in line 6 while maintaining feasibility. Recall that the complementary slackness condition (8.51) is violated when $\hat{c}_{ij} < v_i$ and j is open but i is not assigned to j. There tend to be fewer of these violations when we spread the $\hat{c}_{ij} < v_i$ among the customers i rather than having a few customers with very large v_i values.

Once we have J^+, we can generate an integer primal solution (x^+, y^+) using (8.52) and (8.53). If (x^+, y^+, v^+) satisfies (8.51) for all i and j, then the complementary slackness conditions are all satisfied and (x^+, y^+) is optimal. If, instead, (8.51) is violated for some i and j, then we attempt to reduce these violations using the dual-adjustment procedure, described in the next section.

☐ **EXAMPLE 8.2**

Figure 8.3 depicts an instance of the UFLP with four customers (marked as circles) and three potential facility sites (marked as squares). Fixed costs f_j are marked next to each facility. Each customer has a demand of $h_i = 1$, and transportation costs are equal to the Manhattan-metric distance between the facility and customer. We will use DUALOC's dual-ascent procedure (Algorithm 8.4) to find a feasible solution for this instance.

First, we sort the transportation costs for each customer. In Figure 8.4, for each customer, each facility is positioned based on its distance from the customer. The

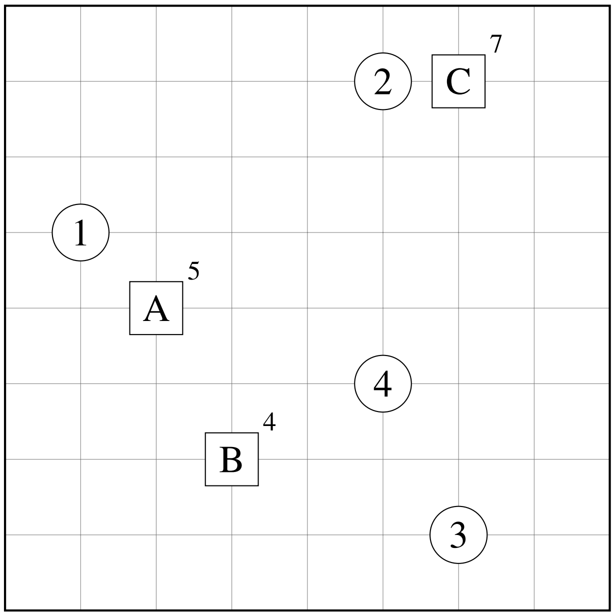


Figure 8.3 Customer and facility layout for Example 8.2.

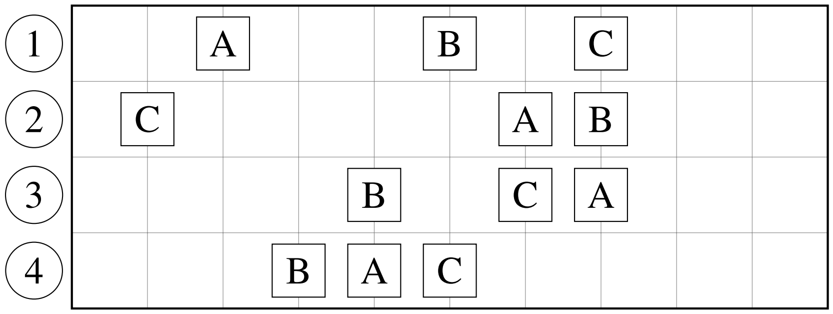


Figure 8.4 Sorted facility positions for Example 8.2.

algorithm begins by setting $v_i \leftarrow \hat{c}_i^1$ and $k_i = 2$ for all i, and $s_j \leftarrow f_j$ for all j:

$$\begin{array}{lll} v_1 = \hat{c}_{1\text{A}} = 2 & k_1 = 2 & s_\text{A} = 5 \\ v_2 = \hat{c}_{2\text{C}} = 1 & k_2 = 2 & s_\text{B} = 4 \\ v_3 = \hat{c}_{3\text{B}} = 4 & k_3 = 2 & s_\text{C} = 7 \\ v_4 = \hat{c}_{4\text{B}} = 3 & k_4 = 2 & \end{array}$$

(You can imagine v_i as being positioned at the relevant spot in Figure 8.4.) Next the algorithm searches for v_i values to increase. One could choose to loop through the customers in any order; we'll go in increasing order to keep things simple. For $i = 1$, facility A is the only j for which $v_1 - \hat{c}_{1j} \geq 0$, so we have $\Delta_1 \leftarrow s_\text{A} = 5$. Since $\Delta_1 > \hat{c}_{1\text{B}} - v_1 = 3$, we set $\Delta_1 \leftarrow 3$, $k_1 \leftarrow 3$, $s_\text{A} \leftarrow 2$, and $v_1 \leftarrow 5$. Similarly, for $i = 2$, $\Delta_2 \leftarrow s_\text{C} = 7$; since $\Delta_2 > \hat{c}_{2\text{C}} - v_2 = 5$, we set $\Delta_2 \leftarrow 5$, $k_2 \leftarrow 3$, $s_\text{C} \leftarrow 2$, and $v_2 \leftarrow 6$. For $i = 3$: $\Delta_3 \leftarrow 2$, $k_3 \leftarrow 3$, $s_\text{B} \leftarrow 2$, and $v_3 \leftarrow 6$. Finally, for $i = 4$: $\Delta_4 \leftarrow 1$, $k_4 \leftarrow 3$, $s_\text{B} \leftarrow 1$, and $v_4 \leftarrow 4$. At the end of this iteration, we have:

$$\begin{array}{lll} v_1 = 5 & k_1 = 3 & s_\text{A} = 2 \\ v_2 = 6 & k_2 = 3 & s_\text{B} = 1 \\ v_3 = 6 & k_3 = 3 & s_\text{C} = 2 \\ v_4 = 4 & k_4 = 3 & \end{array}$$

Since Δ_i reached $\hat{c}_i^{k_i} - v_i$ for at least one i (actually, for all of them), we repeat for another iteration. For $i = 1$: $\Delta_1 \leftarrow \min\{s_\text{A}, s_\text{B}\} = 1$ since $v_1 - \hat{c}_{1j} \geq 0$ for both $j =$ A and B. Since $\Delta_1 < \hat{c}_1^{k_1} - v_1$, we leave Δ_1 and k_1 where they are, and then set $s_\text{A} \leftarrow 1$, $s_\text{B} \leftarrow 0$, and $v_1 \leftarrow 6$. For $i = 2$: $\Delta_2 \leftarrow \min\{s_\text{A}, s_\text{C}\} = 1$ since $v_2 - \hat{c}_{2j}$ for both $j =$ A and C. Since $\Delta_2 = \hat{c}_2^{k_2} - v_2$, we set $k_2 \leftarrow 4$, $s_\text{A} \leftarrow 0$, $s_\text{C} \leftarrow 1$, and $v_2 \leftarrow 7$. For $i = 3$ and 4, we have $\Delta_3,\ \Delta_4 \leftarrow 0$ (since $s_\text{B} = 0$), so there is nothing to do. At the end of this iteration, we have:

$$\begin{array}{lll} v_1 = 6 & k_1 = 3 & s_\text{A} = 0 \\ v_2 = 7 & k_2 = 4 & s_\text{B} = 0 \\ v_3 = 6 & k_3 = 3 & s_\text{C} = 1 \\ v_4 = 4 & k_4 = 3 & \end{array}$$

Since Δ_2 reached $\hat{c}_2^{k_2} - v_2$, we repeat for another iteration. However, at this iteration, we cannot increase v_i for any i since $s_\text{A} = s_\text{B} = 0$, so the **repeat** $\cdots$ **until** loop terminates.

The algorithm returns $v^+ = (6, 7, 6, 4)$ and $J^+ = \{\text{A}, \text{B}\}$. The feasible primal solution obtained from (8.52)–(8.53) is $x^+ = (1, 1, 0)$ and $y_{1\text{A}}^+ = y_{2\text{A}}^+ = y_{3\text{B}}^+ = y_{4\text{B}}^+ = 1$. The dual solution v^+ has objective value $z_D^+ = 23$ and the primal solution has objective value $z_P^+ = 24$. The fact that there is a duality gap indicates that *either* we have not found an optimal solution to the dual LP, *or* we have not found an optimal solution to the primal IP, *or* there is an integrality gap, i.e., the LP relaxation has a fractional optimum.

We can't tell which—yet. To resolve this question, we would run the dual-adjustment procedure. For this instance, the dual-adjustment procedure would yield no improvement to the solution above. We would then use branch-and-bound to close the duality gap, and we would find that there is an optimal integer solution in which we locate only at facility B and to assign all customers to it, for a total cost of 23.

Therefore, the dual solution found by the dual-ascent procedure was optimal, but the corresponding primal solution was not. □

8.2.4.3 The Dual-Adjustment Procedure The dual-adjustment procedure identifies customers and facilities that violate the complementary slackness condition (8.51) and reduces these violations by decreasing the dual variable v_i for some $i \in I$. Doing so frees up slack on some of the binding constraints (8.47), which allows us to increase $v_{i'}$ for other $i' \in I$. Each unit of decrease in v_i allows one unit of increase in $v_{i'}$ (since the coefficients in (8.51) equal 1). The dual objective value will increase if more than one $v_{i'}$ can be increased in this way and will stay the same if only one can be increased. In either case, we may obtain a new (potentially better) primal solution since the set J^+ might change.

We face three questions: (1) Which dual variables v_i are candidates for reduction? (2) Once we reduce v_i, adding slack to some of the constraints, which $v_{i'}$ are candidates for increase? (3) How much should we increase each of the candidate $v_{i'}$? We'll answer each of these questions in turn.

Which v_i to decrease? A customer i is a candidate for reduction in v_i if it violates the complementary slackness condition (8.51). The next lemma characterizes those customers in terms of v and $\hat{c}$.

Lemma 8.6 *Let v be a dual solution and J^+ be a facility set that satisfy PDP1 and PDP2, and let (x^+, y^+) be the corresponding feasible solution calculated from* (8.52)–(8.53). *Then $i \in I$ violates* (8.51) *if and only if $v_i > \hat{c}_{ij}$ for at least two $j \in J^+$.*

Proof. $i \in I$ violates (8.51) if and only if there is some $j' \in J^+$ such that $v_i > \hat{c}_{ij'}$ but $y^+_{ij'} = 0$. This happens if and only if i is assigned to a different $j'' \in J^+$, i.e., if and only if there is a $j'' \in J^+$ such that $\hat{c}_{ij''} \leq \hat{c}_{ij'}$. This happens if and only if $v_i \geq \hat{c}_{ij}$ for at least two $j \in J^+$. ■

Therefore, a customer i is a candidate for reduction in v_i if $v_i > \hat{c}_{ij}$ for at least two $j \in J^+$. The algorithm reduces it only as far as the next-smaller $\hat{c}_{ij}$; that is, it reduces it to $\hat{c}_i^-$, where $\hat{c}_i^-$ is the largest $\hat{c}_{ij}$ (among all $j \in J$) that is strictly less than v_i:

$$\hat{c}_i^- = \max_{j \in J}\{\hat{c}_{ij} | v_i > c_{ij}\}.$$

Which $v_{i'}$ to increase? Suppose $v_i > \hat{c}_{ij}$ for at least two $j \in J^+$ and so we reduce v_i. This adds slack to (8.47) for all $j \in J$ such that $v_i > \hat{c}_{ij}$. Lemma 8.6 implies that two of these constraints correspond to $j^+(i)$ and $j^{++}(i)$, where $j^{++}(i)$ is the second-closest facility in J^+ to i. (Recall that $j^+(i)$ is the closest.) Suppose there is some $i' \in I$ for which there is only one $j \in J^+$ such that $v_{i'} \geq \hat{c}_{i'j}$. We'll say that i' is *solely constrained* by j in this case, because j is the only facility preventing an increase in $v_{i'}$. If i' is solely constrained by $j^+(i)$ or $j^{++}(i)$, then a decrease in v_i can be matched by an increase $v_{i'}$. The algorithm therefore focuses on such i', attempting to increase their $v_{i'}$ values first. It also uses the "solely constrained" test to identify candidates for reduction in v_i: If there are no i' that are solely constrained by $j^+(i)$ or $j^{++}(i)$, the algorithm does not consider reducing v_i, even if i is a candidate for a decrease in v_i as described above.

How much to increase $v_{i'}$? Deciding which $v_{i'}$ to increase, and by how much, is precisely the intent of the dual-ascent procedure! Therefore, the dual-adjustment procedure uses the dual-ascent procedure as a subroutine—first with the set of customers restricted to the candidates for increases in $v_{i'}$, then with i added, and then with the full customer set I.

The dual-adjustment procedure is described in pseudocode in Algorithm 8.5. The algorithm loops through the customers to identify candidates for reducing v_i. A customer is a candidate if (1) it violates the complementary slackness condition (8.51) (this check occurs in line 3, making use of Lemma 8.6), and (2) there are at least two customers that are solely constrained by either $j^+(i)$ or $j^{++}(i)$ (this check occurs in lines 4–5). Assuming customer i passes both checks, lines 6–8 increase the slack for all j for which $v_i > \hat{c}_{ij}$, and line 9 reduces v_i to the next smallest $\hat{c}_{ij}$ value.

Next, the algorithm calls the dual-ascent procedure to decide how to use up the newly created slack. Line 10 restricts I to the customers that are solely constrained by $j^+(i)$ or $j^{++}(i)$; line 11 adds i itself to this set; and line 12 runs the dual-ascent algorithm on the entire set I in order to ensure a valid solution v^+. If v_i has increased, the adjustment procedure repeats (for the same i), and this continues until there is no improvement or v_i reaches its original value. At that point, we move on to the next customer. The algorithm terminates when all customers have been considered.

Algorithm 8.5 Dual-adjustment procedure for DUALOC algorithm

```
1:  for all $i \in I$ do                                           ▷ Loop through customers
2:      repeat
3:          if $v_i > \hat{c}_{ij}$ for at least two $j \in J^+$ then          ▷ Check for CSC violation
4:              $I^+ \leftarrow \{i' \in I | i'$ is solely constrained by $j^+(i)$ or $j^{++}(i)\}$
5:              if $I^+ \neq \emptyset$ then                 ▷ Are there other $v_{i'}$ we can increase?
6:                  for all $j \in J$ s.t. $v_i > \hat{c}_{ij}$ do
7:                      $s_j \leftarrow s_j + v_i - \hat{c}_i^-$                      ▷ Increase slack
8:                  end for
9:                  $v_i \leftarrow \hat{c}_i^-$                                       ▷ Reduce $v_i$
10:                 Run Alg. 8.4 with $I$ restricted to $I^+$             ▷ Increase other $v_{i'}$
11:                 Run Alg. 8.4 with $I$ restricted to $I^+ \cup \{i\}$
12:                 Run Alg. 8.4 for full $I$
13:             end if
14:         end if
15:     until no improvement in dual objective or $v_i$ has resumed its original value
16: end for
```

If the dual-ascent and dual-adjustment procedures result in primal and dual solutions (v^+, x^+, y^+) whose objective values are equal, then v^+ is optimal for the dual LP and (x^+, y^+) is optimal for the primal LP *and* IP. If the objectives are unequal, then a straightforward implementation of branch-and-bound can be used to close the optimality gap. Erlenkotter (1978) reports excellent computational results for this method on several test problems, typically with little or no branching required.

Körkel (1989) proposes computational improvements that speed the DUALOC algorithm up considerably. DUALOC has been adapted to solve many other problems, such as the p-median problem discussed in Section 8.3.2 (Galvão 1980, Nauss and Markland 1981), the stochastic UFLP discussed in Section 8.6.2 (Mirchandani et al. 1985), the Steiner tree problem (Wong 1984), and general supply chain network design problems (Balakrishnan et al. 1989). Goemans and Williamson (1997) discuss DUALOC and other primal–dual algorithms.

8.2.5 Heuristics for the UFLP

Heuristics for combinatorial problems such as the UFLP fall into two categories: *construction heuristics* and *improvement heuristics*. Construction heuristics build a feasible solution from scratch, whereas improvement heuristics start with a feasible solution and attempt to improve it.

The most basic construction heuristics for the UFLP are *greedy* heuristics such as the "greedy-add" procedure (Kuehn and Hamburger 1963): Start with all facilities closed and open the single facility that can serve all customers with the smallest objective function value; then at each iteration open the facility that gives the largest decrease in the objective, stopping when no facility can be opened that will decrease the objective. (See Algorithm 8.6. In the algorithm, x^k, y^k, and z^k refer to the solution when facility k is (temporarily) opened.)

Algorithm 8.6 Greedy-add heuristic for UFLP

```
1:  $x_j \leftarrow 0 \, \forall \, j \in J; z \leftarrow \infty$                          ▷ Initialization
2:  repeat
3:      IMPROVED ← FALSE
4:      for all $k \in J$ s.t. $x_k = 0$ do                                    ▷ Main loop
5:          $x^k \leftarrow x, y^k \leftarrow y$                               ▷ Make copy of current solution
6:          $x^k_k \leftarrow 1$                                               ▷ Open facility $k$
7:          for all $i \in I$ do
8:              $j(i) \leftarrow \text{argmin}_{j \in J : x^k_j = 1}\{c_{ij}\}$   ▷ Assign $i$ to nearest open $j$
9:              $y^k_{i,j(i)} \leftarrow 1$
10:         end for
11:         $z^k \leftarrow \sum_{j \in J} f_j x^k_j + \sum_{i \in I} \sum_{j \in J} h_i c_{ij} y^k_{ij}$   ▷ Calculate cost if open $j$
12:     end for
13:     if $\min_{k \in J}\{z^k\} < z$ then                                    ▷ Compare to current cost
14:         $k^* \leftarrow \text{argmin}_{k \in J}\{z^k\}$                    ▷ Open best facility
15:         $x_{k^*} \leftarrow 1; y \leftarrow y^{k^*}; z \leftarrow \min_{k \in J}\{z^k\}$   ▷ Update current solution
16:         IMPROVED ← TRUE
17:     end if
18: until not IMPROVED
19: return $x$, $y$, $z$
```

☐ **EXAMPLE 8.3**

Let us apply the greedy-add heuristic to the UFLP instance in Example 8.1. We begin with all facilities closed and, one by one, calculate the cost of opening each facility and assigning all customers to it. For example, if we open facility 1 (New York, NY), the single-facility solution costs \$2,591,762 (\$189,600 in fixed cost and \$2,402,162 in transportation cost). If we open facility 2 (Los Angeles, CA), the cost is \$3,638,252, and so on. The best and worst facilities to open, given that we only open one facility, sorted by cost, are listed in Table 8.1, and a few of the corresponding

Table 8.1 Greedy algorithm costs: Iteration 1.

Rank	Facility #	City, State	Cost
1	34	St. Louis, MO	1,935,714
2	69	Springfield, IL	1,941,395
3	13	Indianapolis, IN	1,950,055
4	80	Jefferson City, MO	1,970,644
5	44	Cincinnati, OH	1,983,877
		⋮	
84	68	Salem, OR	3,911,294
85	11	San Jose, CA	3,913,554
86	81	Olympia, WA	3,991,424
87	14	San Francisco, CA	4,019,984
88	21	Seattle, WA	4,030,326

Table 8.2 Greedy algorithm costs: Iteration 2.

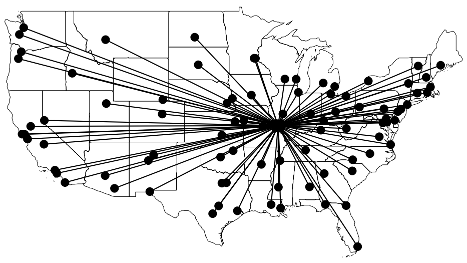
Rank	Facility #	City, State	Cost
1	46	Fresno, CA	1,338,962
2	9	Phoenix, AZ	1,391,766
3	78	Carson City, NV	1,398,997
4	41	Sacramento, CA	1,414,305
5	33	Tucson, AZ	1,424,947
		⋮	
83	42	Minneapolis, MN	1,930,728
84	51	St. Paul, MN	1,933,291
85	69	Springfield, IL	1,936,401
86	66	Tallahassee, FL	1,945,300
87	45	Miami, FL	1,980,578

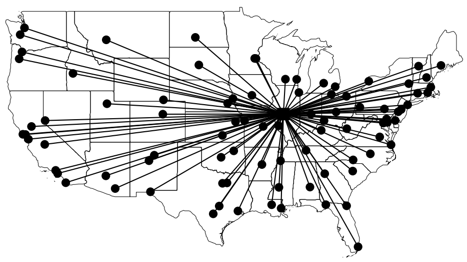
solutions are depicted in Figure 8.5. Since St. Louis, MO is the best city to open, we open it and leave it open for the duration of the heuristic.

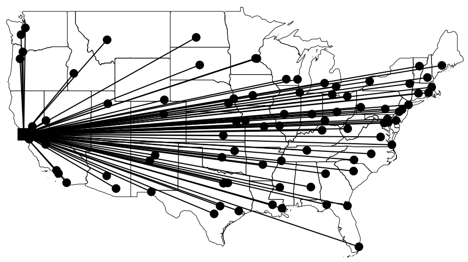
Next we determine the best second facility to open, given that St. Louis is also open. The best and worst facilities are given in Table 8.2 and Figure 8.6.

So, we fix open the facilities in St. Louis, MO, and Fresno, CA. Proceeding in this manner, in iteration 3, we open facility 5, in Philadelphia, PA, to obtain the solution shown in Figure 8.7(a), which has a cost of 904,055. In iteration 4, we open facility 28 in Fort Worth, TX, for a solution with a cost of 821,501 (Figure 8.7(b)). In iteration 5, we open facility 7 in Detroit, MI, for a solution with a cost of 793,443 (Figure 8.7(c)). In iteration 6, the best facility to open is facility 15 (Jacksonville, FL), but the resulting solution has a cost of 807,938, which is greater than the cost of the previous solution. Therefore, the heuristic terminates, returning the 5-facility solution in Figure 8.7(c).

□


(a) St. Louis, MO (cost = 1,935,714).


(b) Springfield, IL (cost = 1,941,395).


(c) San Francisco, CA (cost = 4,019,984).

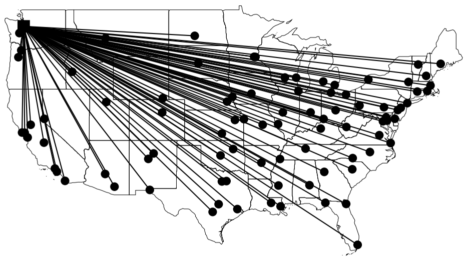
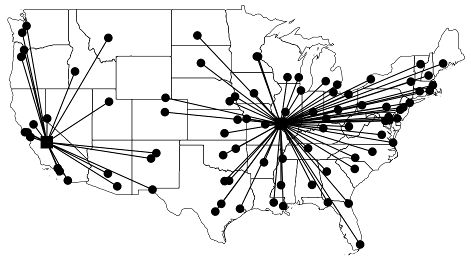
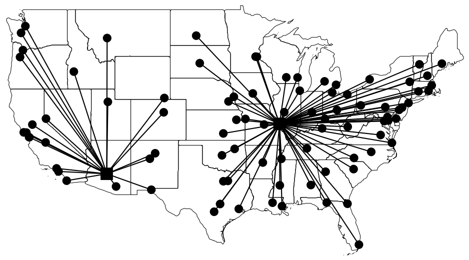

(d) Seattle, WA (cost = 4,030,326).

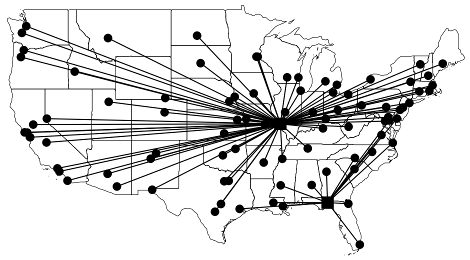
Figure 8.5 Considering each facility for iteration 1 of greedy algorithm for UFLP.



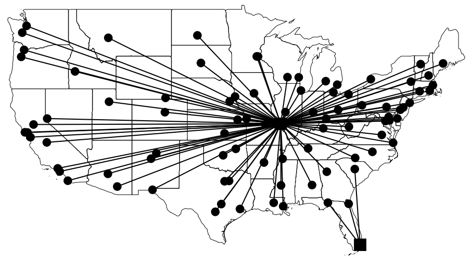
(a) Fresno, CA (cost = 1,338,962).



(b) Phoenix, AZ (cost = 1,391,766).

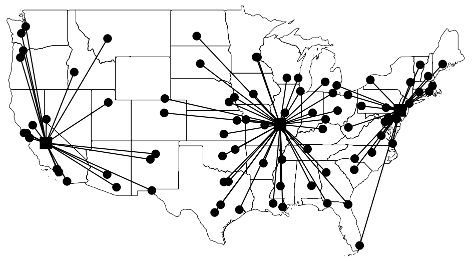


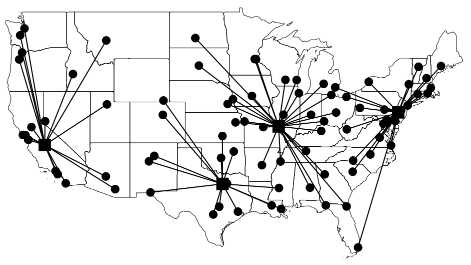
(c) Tallahassee, FL (cost = 1,945,300).



(d) Miami, FL (cost = 1,980,578).

Figure 8.6 Considering each facility for iteration 2 of greedy algorithm for UFLP, with facility in St. Louis, MO, fixed open.


(a) Iter. 3: Philadelphia, PA (cost = 904,055).


(b) Iter. 4: Fort Worth, TX (cost = 821,501).

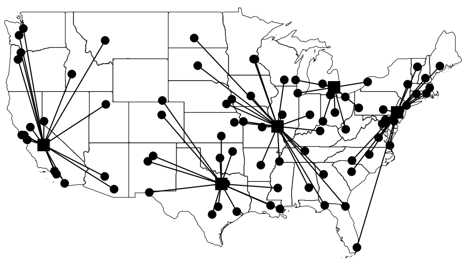

(c) Iter. 5: Detroit, MI (cost = 793,443).

Figure 8.7 Solutions from iterations 3, 4, and 5 of greedy algorithm for UFLP.

By assuming that the next facility to open will be the last, the greedy-add heuristic can easily fall into a trap. For example, if it is optimal to open two facilities, the greedy-add heuristic may first open a facility in the center of the geographical region, which then must stay open for the second iteration, when in fact it is optimal to open one facility on each side of the region.

A reverse approach is called the "greedy-drop" heuristic, which starts with all facilities open and sequentially closes the facility that decreases the objective the most. It has similar advantages and disadvantages as greedy-add.

One important improvement heuristic is the *swap* or *exchange* heuristic (Teitz and Bart 1968), which attempts to find a facility to open and a facility to close such that the new solution has a smaller objective function value. For more on the swap heuristic, see Section 8.3.2.3. Other procedures attempt to find closed facilities that can be opened to reduce the objective function, or open facilities that can be closed.

The heuristics mentioned here have proven to perform well in practice, which means they return good solutions *and* execute quickly. Metaheuristics have also been widely applied to the UFLP. These include genetic algorithms (Jaramillo et al. 2002), tabu search (Al-Sultan and Al-Fawzan 1999), and simulated annealing (Arostegui et al. 2006).

8.3 OTHER MINISUM MODELS

The UFLP is an example of a *minisum* location problem. Minisum models are so called because their objective is to minimize a sum of the costs or distances between customers and their assigned facilities (as well as possibly other terms, such as fixed costs). In contrast, *covering* location problems are more concerned with the maximum distance, with the goal of ensuring that most or all customers are located close to their assigned facilities.

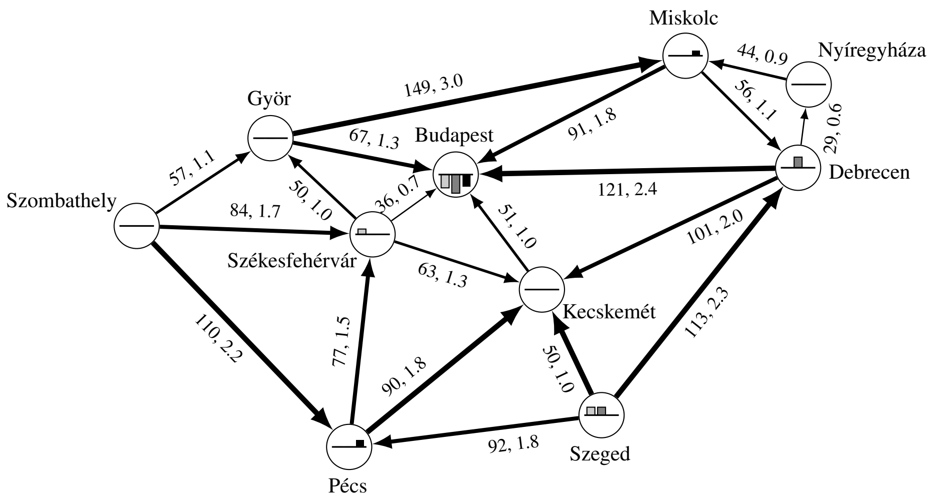


Figure 8.20 Hungary cities arc design problem instance. Grey and black bars inside nodes indicate supply (> 0) or demand (< 0) for three products.

to their capacities, and the fixed and variable costs are listed along the arcs. Variable costs are the same for all products. (For the complete specification of the instance, see the file `hungary.xlsx`.)

The optimal solution to this instance of the arc design problem is drawn in Figure 8.21(a). This solution has a total cost of 647.6. The solution opens 8 arcs. CPLEX solved this instance in less than 1 second.

The optimal solution is different from the solution that would result from solving 3 separate single-product problems, one for each product, and then combining the results. That solution is plotted in Figure 8.21(b). In fact, that solution is not even feasible for the original problem, since it sends 6 units of flow along the arc from Szeged to Kecskemét, but that arc only has a capacity of 3. Solving for each product individually ignores the shared capacity and results in this infeasible solution. This highlights the fact that it can be difficult even to find a feasible solution for the capacitated arc design problem, let alone an optimal one.

□

CASE STUDY 8.1 Locating Fire Stations in Istanbul

Istanbul, Turkey, is one of the world's largest cities, with a population of over 13 million and growing. By 2008, the population growth had rendered the existing set of fire stations insufficient to meet the current needs, prompting the Istanbul Metropolitan Municipality (IMM) to sponsor a project by researchers from Dogus University and Istanbul Technical University to determine locations for new fire stations in the city. Their project is described by Aktaş et al. (2013) and summarized here.

The city of Istanbul is divided into 40 districts and 790 subdistricts. The researchers treated these subdistricts as both the demand nodes and potential facility locations in their location models. The IMM aims to respond to fire incidents within 5 minutes, leading the researchers to use coverage models with a coverage radius based on a five-minute travel time. The researchers used the set covering location problem (SCLP) to identify the cheapest solution that would achieve 100% service, as well as the maximal covering location problem (MCLP) to find solutions that maximize coverage subject to a budget constraint. Both models required all existing fire stations to remain open; the goal was to choose locations for new fire stations.

Their SCLP model differs from the formulation given in Section 8.4.1 in two ways. First, it allows for multiple types of fire stations, each with its own fixed cost and capacity. Second, it requires a given subdistrict to be covered by sufficiently many, or sufficiently large, fire stations to meet the annual number of fire incidents in that subdistrict. In particular, it replaces constraints (8.81) with

$$\sum_{j\in J}\sum_{k\in K} r_k a_{ij} x_{jk} \geq h_i \qquad \forall i \in I, \tag{8.149}$$

where K is the set of fire station types, r_k is the capacity of a type-k station (number of incidents it can handle per year), h_i is the number of incidents in subdistrict i per year, and $x_{jk} = 1$ if we open a fire station of type k in subdistrict j. The model also imposes a constraint requiring at most one type of station to be opened in each subdistrict:

$$\sum_{k\in K} x_{jk} \leq 1 \qquad \forall j \in J. \tag{8.150}$$

Finally, the objective function (8.83) is modified to sum over k in addition to j.

Their MCLP model is similarly modified, replacing constraints (8.86) with

$$h_i z_i \leq \sum_{j\in J}\sum_{k\in K} r_k a_{ij} x_{jk} \qquad \forall i \in I. \tag{8.151}$$

In other words, i only counts as covered if the opened facilities that cover i have sufficient combined capacity to respond to the number of incidents at i. In some versions of their model, they also modify h_i to reflect the number of cultural heritage sites in the subdistrict and to give more weight to those subdistricts that have more such sites. (The city's history goes back more than 2500 years. A group of sites called the Historical Areas of Istanbul was placed on the UNESCO World Heritage List in 1985.)

The research team used a commercial geographic information system (GIS) to assemble the data for the study. The GIS calculated the geographical center of each subdistrict and the average travel times between subdistricts, taking into account the road network and the typical speed on each road link. These travel times were then used to determine the coverage parameters a_{ij} for the SCLP and MCLP. Fixed location costs were assumed to be the same at every location, but different for different fire station types. Demands h_i were estimated from 12 years of historical incident data from IMM.

The status quo solution, consisting of Istanbul's existing 60 fire stations, was shown to cover only 56.6% of the demands in the model (as measured by historical incidents) within a 5-minute service time, and only 18.2% of demands from subdistricts that contain cultural heritage sites. This poor coverage was the result of the city's expansion or changes in the road system and is what prompted this study in the first place.

The SCLP solution, which covers 100% of all demands, required 149 new stations. This exceeded the IMM's budget for opening new stations, which allowed for the equivalent of 64 new stations. Therefore, the researchers imposed this budget constraint in the MCLP and found a solution that covers 93.9% of the demand, including 71.1% of demands from heritage subdistricts. It also double-covers 35.6% of the subdistricts, more than twice the number that are double-covered in the status quo solution. The problems were solved in the modeling language GAMS using the MIP solver CPLEX, with run times of less than 1 second.

As of their 2013 paper, Aktaş, et al. report that IMM had opened 25 new fire stations in subdistricts proposed by the model, with a subsequent slowdown due to economic conditions. Their solution provides a roadmap for future expansion of the fire station network that can be implemented as budgets allow.

PROBLEMS

8.1 (Locating DCs for Toy Stores) A toy store chain operates 100 retail stores throughout the United States. The company currently ships all products from a central distribution center (DC) to the stores, but it is considering closing the central DC and instead operating multiple regional DCs that serve the retail stores. It will use the UFLP to determine where to locate DCs. Planners at the company have identified 24 potential cities in which regional DCs may be located. The file `toy-stores.xlsx` lists the longitude and latitude for all of the locations (stores and DCs), as well as the annual demand (measured in pallets) at each store and the fixed annual location cost at each potential DC location. Using optimization software of your choice, implement the UFLP model from Section 8.2.2 and solve it using the data provided. Assume that transportation from DCs to stores costs \$1 per mile, as measured by the great circle distance between the two locations. Report the optimal cities to locate DCs in and the optimal total annual cost.

8.2 (10-Node UFLP Instance: Exact) The file `10node.xlsx` contains data for a 10-node instance of the UFLP, with nodes located on the unit square and $I = J$, pictured in Figure 8.22. The file lists the x- and y-coordinates, demands h_i, and fixed costs f_j for each node, as well as the transportation cost c_{ij} between each pair of nodes i and j. Transportation costs equal 10 times the Euclidean distance between the nodes. All fixed costs equal 200.

Solve this instance of the UFLP exactly by implementing the UFLP in the modeling language of your choice and solving it with a MIP solver. Report the optimal locations, optimal assignments, and optimal cost.

8.3 (10-Node UFLP Instance: Greedy-Add) Use the greedy-add heuristic to solve the 10-node UFLP instance described in Problem 8.2. Report the facility that is opened at each iteration, as well as the final locations, assignments, and cost.

8.4 (10-Node UFLP Instance: Swap) Suppose we have a solution to the 10-node UFLP instance described in Problem 8.2 in which $x_2 = x_3 = 1$ and $x_j = 0$ for all other j. Use the swap heuristic to improve this solution. Use a best-improving strategy (that is, search through the facilities in order of index, and at each iteration, implement the first swap

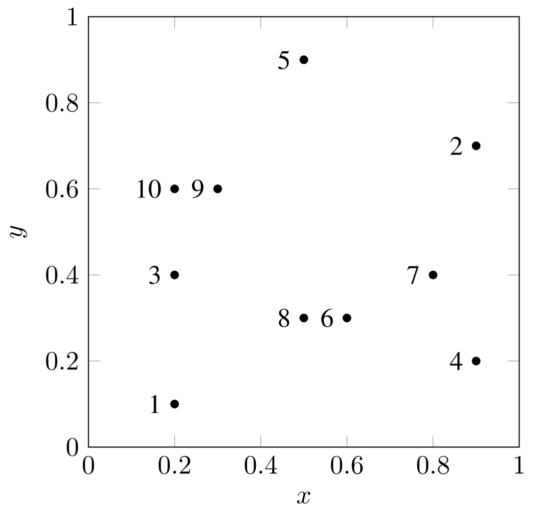


Figure 8.22 10-node facility location instance for Problems 8.2–8.11.

found that improves the cost.) Report the swaps made at each iteration, as well as the final locations, assignments, and cost.

8.5 (10-node pMP Instance: Exact) Using the file `10node.xlsx` (see Problem 8.2), solve the pMP exactly by implementing it in the modeling language of your choice and solving it with a MIP solver. Ignore the fixed costs in the data set and use $p = 4$. Report the optimal locations, optimal assignments, and optimal cost.

8.6 (10-Node pMP Instance: Swap) Suppose we have a solution to the 10-node pMP instance described in Problem 8.5 in which $x_2 = x_3 = x_5 = x_8 = 1$ and $x_j = 0$ for all other j. Use the swap heuristic to improve this solution. Use a best-improving strategy (that is, search through the facilities in order of index, and at each iteration implement the first swap found that improves the cost.) Report the swaps made at each iteration, as well as the final locations, assignments, and cost.

8.7 (10-Node pMP Instance: Neighborhood Search) Suppose we have a solution to the 10-node pMP instance described in Problem 8.5 in which $x_4 = x_5 = x_6 = x_{10} = 1$ and $x_j = 0$ for all other j. Use the neighborhood search heuristic to improve this solution. Report the swaps made at each iteration, as well as the final locations, assignments, and cost.

8.8 (10-node SCLP Instance) Using the file `10node.xlsx` (see Problem 8.2), solve the SCLP exactly by implementing it in the modeling language of your choice and solving it with a MIP solver. Set the fixed cost of every facility equal to 1. Assume that facility j covers customer i if $c_{ij} \leq 2.5$. Report the optimal locations.

8.9 (10-node MCLP Instance) Using the file `10node.xlsx` (see Problem 8.2), solve the MCLP exactly by implementing it in the modeling language of your choice and solving it with a MIP solver. Set $p = 4$. Assume that facility j covers customer i if $c_{ij} \leq 2.5$. Report the optimal locations and the total number of demands covered.

8.10 (10-node MCLP Instance: Coverage vs. p) Using the file `10node.xlsx` (see Problem 8.2), solve the MCLP exactly for $p = 1, 2, \ldots, 10$ using the modeling language and solver of your choice. Assume that facility j covers customer i if $c_{ij} \le 2.5$. Construct a plot similar to Figure 8.14.

8.11 (10-node pCP Instance) Use Algorithm 8.9 to solve the 10-node instance of the pCP specified in the file `10node.xlsx` (see Problem 8.2). Set $p = 3$. Use $r^L = 0$, $r^U = \max_{i\in I, j\in J}\{c_{ij}\}$, and $\epsilon = 0.1$. Report the value of r at each iteration, as well as the optimal locations, assignments, and objective function value.

8.12 (Locating Homework Centers for Chicago Schools) Suppose the City of Chicago wishes to establish homework-help centers at 12 of its public libraries. It wants the homework center locations to be as close as possible to Chicago public schools. In particular, it wants the homework centers to cover as many schools as possible, where a school is "covered" if there is a homework center located within 2 miles of it.

- **a)** Using the files `chicago-schools.csv` and `chicago-libraries.csv` and determining coverage using great circle distances, find the 12 libraries at which homework centers should be established. Report the indices of the libraries selected, as well as the total number of schools covered. (Chicago school and library data are adapted from Chicago Data Portal (2017a,b).)
- **b)** Suppose now that the city wishes to ensure that *all* schools are covered. What is the minimum number of homework centers that must be established to accomplish this?

8.13 (Easy or Hard Modifications?) Which of the following costs can be implemented in the UFLP by modifying the parameters only, without requiring structural changes to the model; that is, without requiring modifications to the variables, objective function, or constraints? Explain your answers briefly.

- **a)** A per-unit cost to ship items from a supplier to facility j. (The cost may be different for each j.)
- **b)** A per-unit processing cost at facility j. (The cost may be different for each j.)
- **c)** A fixed cost to ship items from facility j to customer i. (The cost is independent of the quantity shipped but may be different for each i and j.)
- **d)** A transportation cost from facility j to customer i that is a nonlinear function of the quantity shipped (for example, one of the quantity discount structures discussed in Section 3.4).
- **e)** A fixed capacity-expansion cost that is incurred if the demand served by facility j exceeds a certain threshold.
- **f)** Some facilities are already open; an open facility j can be closed at a cost of $\hat{f}_j$. (In addition, we can open new facilities, as in the UFLP.)

8.14 (LP Relaxation of UFLP) Develop a simple instance of the UFLP for which the optimal solution to the LP relaxation has fractional values of the x_j variables. This solution must be strictly optimal—that is, you can't submit an instance for which the LP relaxation has an optimal solution with all integer values, even if there's another optimal solution, that ties the integer one, with fractional values. Your instance must have $I = J$, that is, all customer nodes are also potential facility sites. Your instance must have at most four nodes.