16

LAGRANGIAN RELAXATION AND NETWORK OPTIMIZATION

I never missed the opportunity to remove obstacles in the way of unity.
—Mohandas Gandhi

Chapter Outline

16.1 INTRODUCTION

As we have noted throughout our discussion in this book, the basic network flow models that we have been studying—shortest paths, maximum flows, minimum cost flows, minimum spanning trees, matchings, and generalized and convex flows—arise in numerous applications. These core network models are also building blocks for many other models and applications, in the sense that many models met in practice have embedded network structure: that is, the broader models are network problems with additional variables and/or constraints.

In this chapter we consider ways to solve these models using a solution strategy known as *decomposition* which permits us to draw upon the many algorithms that we have developed in previous chapters to exploit the underlying network structure. In a sense this chapter serves a dual purpose. First, it permits us to introduce a broader set of network optimization models than we have been considering in our earlier discussion. As such, this chapter provides a glimpse of how network flow models arise in a wide range of applied problem settings that cannot be modeled as pure network flow problems. Second, the chapter introduces a solution method, known as *Lagrangian relaxation*, that has become one of the very few solution methods in optimization that cuts across the domains of linear and integer programming, combinatorial optimization, and nonlinear programming.

Perhaps the best way to understand the basic idea of Lagrangian relaxation is via an example.

Constrained Shortest Paths

Consider the network shown in Figure 16.1(a) which has two attributes associated with each arc (i, j): a cost c_{ij} and a traversal time t_{ij}. Suppose that we wish to find the shortest path from the source node 1 to the sink node 6, but we wish to restrict our choice of paths to those that require no more than $T = 10$ time units to traverse. This type of constrained shortest path application arises frequently in practice since in many contexts a company (e.g., a package delivery firm) wants to provide its services at the lowest possible cost and yet ensure a certain level of service to its customers (as embodied in the time restriction). In general, the constrained shortest path problem from node 1 to node n can be stated as the following integer programming problem:

$$\text{Minimize} \quad \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{16.1a}$$

subject to

$$\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = \begin{cases} 1 & \text{for } i = 1 \\ 0 & \text{for } i \in N - \{1, n\}, \\ -1 & \text{for } i = n \end{cases} \tag{16.1b}$$

$$\sum_{(i,j)\in A} t_{ij}x_{ij} \le T, \tag{16.1c}$$

$$x_{ij} = 0 \text{ or } 1 \qquad \text{for all } (i, j) \in A. \tag{16.1d}$$

The problem is not a shortest path problem because of the timing restriction. Rather, it is a shortest path problem with an additional side constraint (16.1c). Instead of solving this problem directly, suppose that we adopt an indirect approach by combining time and cost into a single *modified cost*; that is, we place a dollar equivalent on time. So instead of setting a limit on the total time we can take on the chosen path, we set a "toll charge" on each arc proportional to the time that it takes to

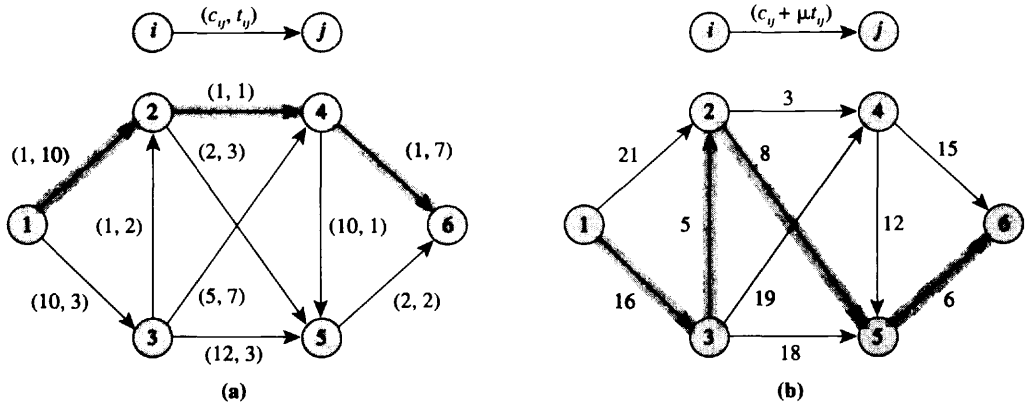

Figure 16.1 Time-constrained shortest path problem: (a) constrained shortest path problem (bold lines denote the shortest path for $\mu = 0$); (b) modified cost $c + \mu t$ with Lagrange multiplier $\mu = 2$ (bold lines denote the shortest path).

traverse that arc. For example, we might charge \$2 for each hour that it takes to traverse any arc. Note that if the toll charge is zero, we are ignoring time altogether and the problem becomes a usual shortest path problem with respect to the given costs. On the other hand, if the toll charge is very large, these charges become the dominant cost and we will be seeking the quickest path from the source to the sink. Can we find a toll charge somewhere in between these values so that by solving the shortest path problem with the combined costs (the toll charges and the original costs), we solve the constrained shortest path problem as a single shortest path problem?

For any choice μ of the toll charge, we solve a shortest path problem with respect to the modified costs $c_{ij} + \mu t_{ij}$. For the sample data shown in Figure 16.1(a), if $\mu = 0$, the modified problem becomes the shortest path problem with respect to the original costs c_{ij} and the shortest path 1–2–4–6 has length 3. This value is an obvious lower bound on the length of the constrained shortest path since it ignores the timing constraint. Now suppose that we set $\mu = 2$ and solve the modified problem. Figure 16.1(b) shows the modified costs $c_{ij} + 2t_{ij}$. The shortest path 1–3–2–5–6 has length 35. In this case, the path 1–3–2–5–6 that solves the modified problem happens to require 10 units to traverse, so it is a feasible constrained shortest path. Is it an optimal constrained shortest path?

To answer this question, let us make an important observation (which we will prove formally in the next section). Let P, with cost $c_P = \sum_{(i,j)\in P} c_{ij}$ and traversal time $t_P = \sum_{(i,j)\in P} t_{ij}$, be *any* feasible path to the constrained shortest path problem, and let $l(\mu)$ denote the optimal length of the shortest path with the modified costs when we impose a toll of μ units. Since the path P is feasible for the constrained shortest path problem, the time t_P required to traverse this path is at most $T = 10$ units. With respect to the modified costs $c_{ij} + \mu t_{ij}$, the cost $c_P + \mu t_P$ of the path P is the path's true cost c_P plus $\mu t_P \leq \mu T$ units. Therefore, if we subtract μT from the modified cost $c_P + \mu t_P$ of this path, we obtain a lower bound $c_P + \mu t_P - \mu T = c_P + \mu(t_P - T) \leq c_P$ on the cost c_P of this path. Since the shortest path with respect to the modified arc costs is less than or equal to the modified cost of any particular path, $l(\mu) \leq c_P + \mu t_P$ and so $l(\mu) - \mu T$ is a *common* lower bound on the length of *any* feasible path P and thus on the length of the constrained shortest path. Because this argument is completely general and applies to any value $\mu \geq 0$ of the toll charges, if we subtract μT from the optimal length of the shortest path of the modified problem, we obtain a lower bound on the optimal cost of the constrained shortest path problem.

Bounding Principle. *For any nonnegative value of the toll μ, the length $l(\mu)$ of the modified shortest path with costs $c_{ij} + \mu t_{ij}$ minus μT is a lower bound on the length of the constrained shortest path.*

Note that for our numerical example, for $\mu = 2$, the cost of the modified shortest path problem is 35 units and so $35 - 2(T) = 35 - 2(10) = 15$ is a lower bound on the length of the optimal constrained shortest path. But since the path 1–3–2–5–6 is a feasible solution to the constrained shortest path problem and its cost equals the lower bound of 15 units, we can be assured that it is an optimal constrained shortest path.

Observe that in this example we have been able to solve a difficult optimization

model (the constrained shortest path problem is an $\mathcal{NP}$-complete problem) by removing one or more problem constraints—in this case the single timing constraint—that makes the problem much more difficult to solve. Rather than solving the difficult optimization problem directly, we combined the complicating timing constraint with the original objective function, via the toll μ, so that we could then solve a resulting embedded shortest path problem. The motivation for adopting this approach was our observation that the original constrained shortest path problem had an attractive substructure, the shortest path problem, that we would like to exploit algorithmically. Whenever we can identify such attractive substructure, we could adopt a similar approach. For reasons that will become clearer in the next section, this general solution approach has become known as *Lagrangian relaxation.*

In our example we have been fortunate to find a constrained shortest path by solving the Lagrangian subproblem for a particular choice of the toll μ. We will not always be so lucky; nevertheless, as we will see, the lower bounding mechanism of Lagrangian relaxation frequently provides valuable information that we can exploit algorithmically.

Lagrangian relaxation is a general solution strategy for solving mathematical programs that permits us to decompose problems to exploit their special structure. As such, this solution approach is perfectly tailored for solving many models with embedded network structure. The Lagrangian solution strategy has a number of significant advantages:

1. Since it is often possible to decompose models in several ways and apply Lagrangian relaxation to each different decomposition, Lagrangian relaxation is a very flexible solution approach. Indeed, because of its flexibility, Lagrangian relaxation is more of a general problem solving strategy and solution framework than any single solution technique.
2. In decomposing problems, Lagrangian relaxation solves core subproblems as stand-alone models. Consequently, the solution approach permits us to exploit any known methodology or algorithm for solving the subproblems. In particular, when the subproblems are network models, the Lagrangian solution approach can take advantage of the various algorithms that we have developed previously in this book.
3. As we have already noted, Lagrangian relaxation permits us to develop bounds on the value of the optimal objective function and, frequently, to quickly generate good, though not necessarily optimal solutions with associated performance guarantees—that is, a bound on how far the solution could possibly be from optimality (in objective function value). In many instances in the context of integer programming, the bounds provided by Lagrangian relaxation methods are much better than those generated by solving the linear programming relaxation of the problems, and as a consequence, Lagrangian relaxation is often an attractive alternative to linear programming as a bounding mechanism in branch-and-bound methods for solving integer programs.
4. In many instances we can use Lagrangian relaxation methods to devise effective heuristic solution methods for solving complex combinatorial optimization problems and integer programs.

In the remainder of this chapter we describe the Lagrangian relaxation solution approach in more detail and demonstrate its use in solving several important network optimization models. Our purpose is not to present a comprehensive treatment of Lagrangian relaxation or of its applications to the field of network optimization, but rather to introduce this general solution strategy and to illustrate its applications in a way that would lay the essential foundations for applying the method in many other problem contexts. As a by-product of this discussion, in the text and in the exercises at the end of this chapter we introduce several noteworthy network optimization models that we do not treat elsewhere in the book.

Since one of the principal uses of Lagrangian relaxation is within implicit enumeration procedures for solving integer programs, before describing Lagrangian relaxation in more detail, we first discuss its use within classical branch-and-bound algorithms for solving integer programs. The reader can skip this section without loss of continuity.

16.2 PROBLEM RELAXATIONS AND BRANCH AND BOUND

In the last section we observed that Lagrangian relaxation permits us to develop a lower bound on the optimal length of a constrained shortest path. In Section 16.3 we develop a generalization of this result, showing that we can obtain a lower bound on the optimal objective function value of any minimization problem. These lower bounds can be of considerable value: for example, for our constrained shortest path example, we were able to use a lower bound to demonstrate that a particular solution that we generated by solving a shortest path subproblem, with modified costs, was optimal for the overall constrained problem. In general, we will not always be as fortunate in being able to use a lower bound to guarantee that the solution to a single subproblem solves the original problem. Nevertheless, as we show briefly in the section, we might still be able to use lower bounds as an algorithmic tool in reducing the number of computations required to solve combinatorial optimization problems formulated as integer programs.

Consider the following integer programming model:

Minimize cx

subject to

$x \in F$.

In this formulation, the set F represents the set of feasible solutions to an integer program, that is, the set of solutions $x = (x_1, x_2, \ldots, x_J)$ to the system

$$\mathcal{A}x = b,$$

$$x_j = 0 \text{ or } 1 \qquad \text{for } j = 1, 2, \ldots, J.$$

In a certain conceptual sense, this integer program is trivial to solve: We simply enumerate every combination of the decision variables, that is, all zero–one vectors $(x_1, x_2, \ldots, x_J)$ obtained by setting each variable x_j to value zero or 1; from among

16.3 LAGRANGIAN RELAXATION TECHNIQUE

To describe the general form of the Lagrangian relaxation procedure, suppose that we consider the following generic optimization model formulated in terms of a vector x of decision variables:

$$z^* = \min cx$$

subject to

$$\mathcal{A}x = b, \tag{P}$$

$$x \in X.$$

This model (P) has a linear objective function cx and a set $\mathcal{A}x = b$ of explicit linear constraints. The decision variables x are also constrained to lie in a given constraint set X which, as we will see, often models embedded network flow structure. For example, the constraint set $X = \{x : \mathcal{N}x = q, 0 \leq x \leq u\}$ might be all the feasible solutions to a network flow problem with a supply/demand vector q. Or, the set X might contain the incidence vectors of all spanning trees or matchings of a given graph. Unless we state otherwise, we assume that the set X is finite (e.g., for network flow problems, we will let it be the finite set of spanning tree solutions).

As its name suggests, the Lagrangian relaxation procedure uses the idea of relaxing the explicit linear constraints by bringing them into the objective function with associated Lagrange multipliers μ (this old idea might be a familiar one from advanced calculus in the context of solving nonlinear optimization problems). We refer to the resulting problem

$$\text{Minimize} \quad cx + \mu(\mathcal{A}x - b)$$

subject to

$$x \in X,$$

as a *Lagrangian relaxation* or *Lagrangian subproblem* of the original problem, and refer to the function

$$L(\mu) = \min\{cx + \mu(\mathcal{A}x - b) : x \in X\},$$

as the *Lagrangian function*. Note that since in forming the Lagrangian relaxation, we have eliminated the constraints $\mathcal{A}x = b$ from the problem formulation, the solution of the Lagrangian subproblem need not be feasible for the original problem (P). Can we obtain any useful information about the original problem even when the solution to the Lagrangian subproblem is not feasible in the original problem (P)? The following elementary observation is a key result that helps to answer this question and that motivates the use of the Lagrangian relaxation technique in general.

Lemma 16.1 (Lagrangian Bounding Principle). *For any vector μ of the Lagrangian multipliers, the value $L(\mu)$ of the Lagrangian function is a lower bound on the optimal objective function value z^* of the original optimization problem* (P).

Proof. Since $\mathcal{A}x = b$ for every feasible solution to (P), for any vector μ of Lagrangian multipliers, $z^* = \min\{cx : \mathcal{A}x = b, x \in X\} = \min\{cx + \mu(\mathcal{A}x - b) : \mathcal{A}x = b, x \in X\}$. Since removing the constraints $\mathcal{A}x = b$ from the second formulation

cannot lead to an increase in the value of the objective function (the value might decrease), $z^* \geq \min\{cx + \mu(\mathcal{A}x - b) : x \in X\} = L(\mu)$. ◆

As we have seen, for any value of the Lagrangian multiplier μ, $L(\mu)$ is a lower bound on the optimal objective function value of the original problem. To obtain the sharpest possible lower bound, we would need to solve the following optimization problem

$$L^* = \max_{\mu} L(\mu)$$

which we refer to as the *Lagrangian multiplier problem* associated with the original optimization problem (P). The Lagrangian bounding principle has the following immediate implication.

Property 16.2 (Weak Duality). *The optimal objective function value L^* of the Lagrangian multiplier problem is always a lower bound on the optimal objective function value of the problem* (P) *(i.e., $L^* \leq z^*$).*

Our preceding discussion provides us with valid bounds for comparing objective function values of the Lagrange multiplier problem and optimization (P) for any choices of the Lagrange multipliers μ and any feasible solution x of (P):

$$L(\mu) \leq L^* \leq z^* \leq cx.$$

These inequalities furnish us with a guarantee when a Lagrange multiplier μ to the Lagrange multiplier problem or a feasible solution x to the original problem (P) are optimal.

Property 16.3 (Optimality Test)

(a) Suppose that μ is a vector of Lagrangian multipliers and x is a feasible solution to the optimization problem (P) *satisfying the condition $L(\mu) = cx$. Then $L(\mu)$ is an optimal solution of the Lagrangian multiplier problem [i.e., $L^* = L(\mu)$] and x is an optimal solution to the optimization problem* (P).

(b) If for some choice of the Lagrangian multiplier vector μ, the solution x^* of the Lagrangian relaxation is feasible in the optimization problem (P), *then x^* is an optimal solution to the optimization problem* (P) *and μ is an optimal solution to the Lagrangian multiplier problem.*

Note that by assumption in part (b) of this property, $L(\mu) = cx^* + \mu(\mathcal{A}x^* - b)$ and $\mathcal{A}x^* = b$. Therefore, $L(\mu) = cx^*$ and part (a) implies that x^* solves problem (P) and μ solves the Lagrangian multiplier problem.

As indicated by Property 16.3, the bounding principle immediately implies one advantage of the Lagrangian relaxation approach—the method can give us a *certificate* [in the form of the equality $L(\mu) = cx$ for some Lagrange multiplier μ] for guaranteeing that a given feasible solution x to the optimization problem (P) is an optimal solution. Even if $L(\mu) < cx$, having the lower bound permits us to state a bound on how far a given solution is from optimality: If $[cx - L(\mu)]/L(\mu) \leq 0.05$, for example, we know that the objective function value of the feasible solution x is no more than 5% from optimality. This type of bound is very useful in practice—it

permits us to assess the degree of suboptimality of given solutions and it permits us to terminate our search for an optimal solution when we have a solution that we know is close enough to optimality (in objective function value) for our purposes.

Lagrangian Relaxation and Inequality Constraints

In the optimization model (P), the constraints $\mathcal{A}x = b$ are all equality constraints. In practice, we often encounter models, such as the constrained shortest path problem, that are formulated more naturally in inequality form $\mathcal{A}x \leq b$. The Lagrangian multiplier problem for these problems is a slight variant of the one we have just introduced: The Lagrangian multiplier problem becomes

$$L^* = \max_{\mu \geq 0} L(\mu).$$

That is, the only change in the Lagrangian multiplier problem is that the Lagrangian multipliers now are restricted to be nonnegative. In Exercise 16.1, by introducing "slack variables" to formulate the inequality problem as an equivalent equality problem, we show how to obtain this optimal multiplier problem from the one we have considered for the equality problem. This development implies that the bounding property, the weak duality property, and the optimality test 16.3(a) are valid when we apply Lagrangian relaxation to any combination of equality and inequality constraints.

There is, however, one substantial difference between relaxing equality constraints and inequality constraints. When we relax inequality constraints $\mathcal{A}x \leq b$, if the solution x^* of the Lagrangian subproblem happens to satisfy these constraints, it need *not* be optimal (see Exercise 16.2). In addition to being feasible, this solution needs to satisfy the *complementary slackness condition* $\mu(\mathcal{A}x^* - b) = 0$, which is familiar to us from much of our previous discussion of network flows in section 9.4.

Property 16.4. *Suppose that we apply Lagrangian relaxation to the optimization problem* ($P^{\leq}$) *defined as minimize* $\{cx : \mathcal{A}x \leq b \text{ and } x \in X\}$ *by relaxing the inequalities* $\mathcal{A}x \leq b$. *Suppose, further, that for some choice of the Lagrangian multiplier vector* μ, *the solution* x^* *of the Lagrangian relaxation* (1) *is feasible in the optimization problem* ($P^{\leq}$), *and* (2) *satisfies the complementary slackness condition* $\mu(\mathcal{A}x^* - b) = 0$. *Then* x^* *is an optimal solution to the optimization problem* ($P^{\leq}$).

Proof. By assumption, $L(\mu) = cx^* + \mu(\mathcal{A}x^* - b)$. Since $\mu(\mathcal{A}x^* - b) = 0$, $L(\mu) = cx^*$. Moreover, since $\mathcal{A}x^* \leq b$, x^* is feasible, and so by Property 16.3(a) x^* solves problem ($P^{\leq}$). ◆

Are solutions to the Lagrangian subproblem of use in solving the original problem? Properties 16.3 and 16.4 show that certain solutions of the Lagrangian subproblem provably solve the original problem. We might distinguish two other cases: (1) when solutions obtained by relaxing inequality constraints are feasible but are not provably optimal for the original problem (since they do not satisfy the complementary slackness condition), and (2) when solutions to the Lagrangian relaxation are not feasible in the original problem.

In the first case, the solutions are candidate optimal solutions (possibly for use

in a branch-and-bound procedure). In the second case, for many applications, researchers have been able to devise methods to modify "modestly" infeasible solutions so that they become feasible with only a slightly degradation in the objective function value. These observations suggest that we might be able to use the solutions obtained from the Lagrangian subproblem as "approximate" solutions to the original problem, even when they are not provably optimal; in these instances, we can use Lagrangian relaxation as a heuristic method for generating provably good solutions in practice (the solutions might be provably good because of the Lagrangian lower bound information). The development of these heuristic methods depends heavily on the problem context we are studying, so we will not attempt to provide any further details.

Solving the Lagrangian Multiplier Problem

How might we solve the Lagrangian multiplier problem? To develop an understanding of possible solution techniques, let us consider the constrained shortest path problem that we defined in Section 16.1. Suppose that now we have a time limitation of $T = 14$ instead of $T = 10$. When we relax the time constraint, the Lagrangian multiplier function $L(\mu)$ for the constrained shortest path problem becomes

$$L(\mu) = \min\{c_P + \mu(t_P - T) : P \in \mathcal{P}\}.$$

In this formulation, $\mathcal{P}$ is the collection of all directed paths from the source node 1 to the sink node n. For convenience, we refer to the quantity $c_P + \mu(t_P - T)$ as the *composite cost* of the path P. For a specific value of the Lagrangian multiplier μ, we can solve $L(\mu)$ by enumerating all the directed paths in $\mathcal{P}$ and choosing the path with the smallest composite cost. Consequently, we can solve the Lagrangian multiplier problem by determining $L(\mu)$ for all nonnegative values of the Lagrangian multiplier μ and choosing the value that achieves $\max_{\mu \geq 0} L(\mu)$.

Let us illustrate this brute force approach geometrically. Figure 16.2 records the cost and time data for every path for our numerical example. Note that the composite cost $c_P + \mu(t_P - T)$ for any path P is a linear function of μ with an intercept of c_P and a slope of $(t_P - T)$. In Figure 16.3 we have plotted each of these path composite cost functions. Note that for any specific value of the Lagrange multiplier μ, we can find $L(\mu)$ by evaluating each composite cost function (line) and identifying the one with the least cost. This observation implies that the Lagrangian multiplier function $L(\mu)$ is the lower envelope of the composite cost lines and that the highest point on this envelope corresponds to the optimal solution of the Lagrangian multiplier problem.

In practice, we would never attempt to solve the problem in this way because the number of directed paths from the source node to the sink node typically grows exponentially in the number of nodes in the underlying network, so any such enumeration procedure would be prohibitively expensive. Nevertheless, this problem geometry helps us to understand the nature of the Lagrangian multiplier problem and suggests methods for solving the problem.

As we noted in the preceding paragraph, to find the optimal multiplier value μ^* of the Lagrangian multiplier problem, we need to find the highest point of the Lagrangian multiplier function $L(\mu)$. Suppose that we consider the polyhedron de-

Path P	Path cost c_P	Path time t_P	Composite cost $c_P + \mu (t_P - T)$
1–2–4–6	3	18	$3 + 4\mu$
1–2–5–6	5	15	$5 + \mu$
1–2–4–5–6	14	14	14
1–3–2–4–6	13	13	$13 - \mu$
1–3–2–5–6	15	10	$15 - 4\mu$
1–3–2–4–5–6	24	9	$24 - 5\mu$
1–3–4–6	16	17	$16 + 3\mu$
1–3–4–5–6	27	13	$27 - \mu$
1–3–5–6	24	8	$24 - 6\mu$

Figure 16.2 Path cost and time data for constrained shortest path example with $T = 14$.

fined by those points that lie on or below the function $L(\mu)$. These are the shaded points in Figure 16.3. Then geometrically, we are finding the highest point in a polyhedron defined by the function $L(\mu)$, which is a linear program.

Even though we have illustrated this property on a specific example, this situation is completely general. Consider the generic optimization model (P), defined

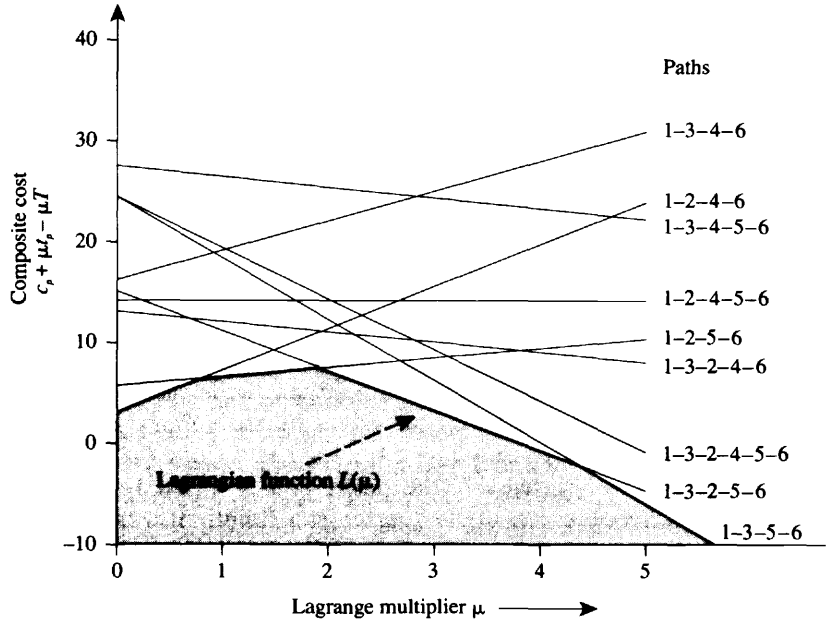

Figure 16.3 Lagrangian function for $T = 14$.

as $\min\{cx : \mathcal{A}x = b, x \in X\}$ and suppose that the set $X = \{x^1, x^2, \ldots, x^K\}$ is finite. By relaxing the constraints $\mathcal{A}x = b$, we obtain the Lagrangian multiplier function $L(\mu) = \min\{cx + \mu(\mathcal{A}x - b) : x \in X\}$. By definition,

$$L(\mu) \leq cx^k + \mu(\mathcal{A}x^k - b) \qquad \text{for all } k = 1, 2, \ldots, K.$$

In the space of composite costs and Lagrange multipliers μ (as in Figure 16.3), each function $cx^k + \mu(\mathcal{A}x^k - b)$ is a multidimensional "line" called a *hyperplane* (if μ is two-dimensional, it is a plane). The Lagrangian multiplier function $L(\mu)$ is the lower envelope of the hyperplanes $cx^k + \mu(\mathcal{A}x^k - b)$ for $k = 1, 2, \ldots, K$. In the Lagrangian multiplier problem, we wish to determine the highest point on this envelope: We can find this point by solving the optimization problem

$$\text{Maximize} \quad w$$

subject to

$$w \leq cx^k + \mu(\mathcal{A}x^k - b) \qquad \text{for all } k = 1, 2, \ldots, K,$$

$$\mu \text{ unrestricted},$$

which is clearly a linear program. We state this result as a theorem.

Theorem 16.5. *The Lagrangian multiplier problem* $L^* = \max_\mu L(\mu)$ *with* $L(\mu) = \min\{cx^k + \mu(\mathcal{A}x - b) : x \in X\}$ *is equivalent to the linear programming problem* $L^* = \max\{w : w \leq cx^k + \mu(\mathcal{A}x^k - b)$ *for* $k = 1, 2, \ldots, K\}$. ◆

Since, as shown by the preceding theorem, the Lagrangian multiplier problem is a linear program, we could solve this problem by applying the linear programming methodology. One resulting algorithm, which is known as *Dantzig–Wolfe decomposition* or *generalized linear programming*, is an important solution methodology that we discuss in some depth in Chapter 17 in the context of solving the multicommodity flow problem. One of the disadvantages of this approach is that it requires the solution of a series of linear programs that are rather expensive computationally. Another approach might be to apply some type of gradient method to the Lagrangian function $L(\mu)$. As shown by the constrained shortest path example, the added complication of this approach is that the Lagrangian function $L(\mu)$ is not differentiable. It is differentiable whenever the optimal solution of the Lagrangian subproblem is unique; but when the subproblem has two or more solutions, the Lagrangian function generally is not differentiable. For example, in Figure 16.4, at $\mu = 0$, the path 1–2–4–6 is the unique shortest path solution to the subproblem and the function $L(\mu)$ is differentiable. At this point, for the path $P = 1$–2–4–6, $L(\mu) = c_P + \mu(t_P - T)$; since $t_P = 18$ and $T = 14$, $L(\mu)$ has a slope $(t_P - T) = (18 - 14) = 4$. At the point $\mu = 2$, however, the paths 1–2–5–6 and 1–3–2–5–6 both solve the Lagrangian subproblem and the Lagrangian function is not differentiable. To accommodate these situations, we next describe a technique, known as the *subgradient optimization technique*, for solving the (nondifferentiable) Lagrangian multiplier problem.

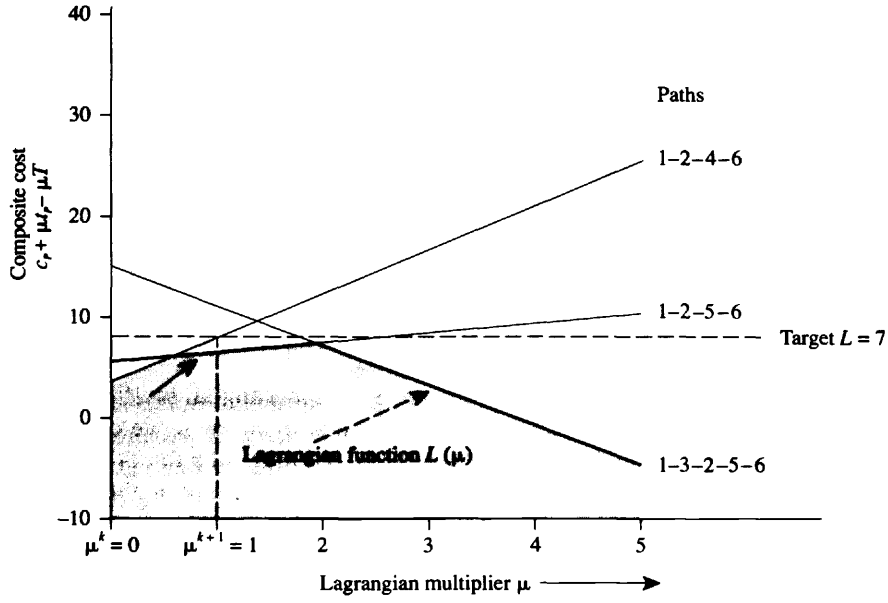

Figure 16.4 Steps of Newton's method for $T = 14$.

Subgradient Optimization Technique

In solving optimization problems with the nonlinear objective function $f(x)$ of an n-dimensional vector x, researchers and practitioners often use variations of the following classical idea: Form the gradient $\nabla f(x)$ of f defined as a row vector with components $(\partial f(x)/dx_1, \partial f(x)/dx_2, \ldots, \partial f(x)/dx_n)$. Recall from advanced calculus that the directional derivative of f in the direction d satisfies the equality

$$\lim_{\theta \to 0} \frac{f(x + \theta d) - f(x)}{\theta} = \nabla f(x) d.$$

So if we choose the direction d so that $\nabla f(x)d > 0$ and move in the direction d with a small enough "step length" θ—that is, change x to $x + \theta d$—we move uphill. This simple observation lies at the core of a considerable literature in nonlinear programming known as *gradient methods*.

Suppose that in solving the Lagrangian multiplier problem, we are at a point where the Lagrangian function $L(\mu) = \min\{cx + \mu(\mathcal{A}x - b) : x \in X\}$ has a unique solution $\bar{x}$, so is differentiable. Since $L(\mu) = c\bar{x} + \mu(\mathcal{A}\bar{x} - b)$ and the solution $\bar{x}$ remains optimal for small changes in the value of μ, the gradient at this point is $\mathcal{A}\bar{x} - b$, so a gradient method would change the value of μ as follows:

$$\mu \leftarrow \mu + \theta(\mathcal{A}\bar{x} - b).$$

In this expression, θ is a step size (a scalar) that specifies how far we move in the gradient direction. Note that this procedure has a nice intuitive interpretation. If $(\mathcal{A}\bar{x} - b)_i = 0$, the solution x uses up exactly the required units of the ith resource, and we hold the Lagrange multiplier (the toll) μ_i of that resource at its current value;

if $(\mathcal{A}\bar{x} - b)_i < 0$, the solution x uses up less than the available units of the ith resource and we decrease the Lagrange multiplier μ_i on that resource; and if $(\mathcal{A}\bar{x} - b)_i > 0$, the solution x uses up more than the available units of the ith resource and we increase the Lagrange multiplier μ_i on that resource.

To solve the Lagrangian multiplier problem, we adopt a rather natural extension of this solution approach. We let μ^0 be any initial choice of the Lagrange multiplier; we determine the subsequent values μ^k for $k = 1, 2, \ldots$, of the Lagrange multipliers as follows:

$$\mu^{k+1} = \mu^k + \theta_k(\mathcal{A}x^k - b).$$

In this expression, x^k is any solution to the Lagrangian subproblem when $\mu = \mu^k$ and θ_k is the step length at the kth iteration.

To ensure that this method solves the Lagrangian multiplier problem, we need to exercise some care in the choice of the step sizes. If we choose them too small, the algorithm would become stuck at the current point and not converge; if we choose the step sizes too large, the iterates μ^k might overshoot the optimal solution and perhaps even oscillate between two nonoptimal solutions (see Exercise 16.4 for an example). The following compromise ensures that the algorithm strikes an appropriate balance between these extremes and does converge:

$$\theta_k \to 0 \quad \text{and} \quad \sum_{j=1}^{k} \theta_j \to \infty.$$

For example, choosing $\theta_k = 1/k$ satisfies these conditions. These conditions ensure that the algorithm always converges to an optimal solution of the multiplier problem, but a proof of this convergence result is beyond the scope of our coverage in this book (the reference notes cite papers and books that examine the convergence of subgradient methods).

One important variant of the subgradient optimization procedure would be an adaptation of "Newton's method" for solving systems of nonlinear equations. Suppose, as before, that $L(\mu^k) = cx^k + \mu^k(\mathcal{A}x^k - b)$; that is, x^k solves the Lagrangian subproblem when $\mu = \mu^k$. Suppose that we assume that x^k continues to solve the Lagrangian subproblem as we vary μ; or, stated in another way, we make a linear approximation $r(\mu) = cx^k + \mu(\mathcal{A}x^k - b)$ to $L(\mu)$. Suppose further that we know the optimal value L^* of the Lagrangian multiplier problem (which we do not). Then we might move in the subgradient direction until the value of the linear approximation exactly equals L^*. Figure 16.4 shows an example of this procedure when applied to our constrained shortest path example, starting with $\mu^k = 0$. At this point, the path $P = 1\text{–}2\text{–}4\text{–}6$ solves the Lagrangian subproblem and $\mathcal{A}x^k - b$ equals $t_P - T = 18 - 14 = 4$. Since $L^* = 7$ and the path P has a cost $c_P = 3$, in accordance with this linear approximation, or Newton's method, we would approximate $L(\mu)$ by $r(\mu) = 3 + 4\mu$, set $3 + 4\mu = 7$, and define the new value of μ as $\mu^{k+1} = (7 - 3)/4 = 1$. In general, we set the step length θ_k so that

$$r(\mu^{k+1}) = cx^k + \mu^{k+1}(\mathcal{A}x^k - b) = L^*,$$

or since, $\mu^{k+1} = \mu^k + \theta_k(\mathcal{A}x^k - b)$,

$$r(\mu^{k+1}) = cx^k + [\mu^k + \theta_k(\mathcal{A}x^k - b)](\mathcal{A}x^k - b) = L^*.$$

Collecting terms, recalling that $L(\mu^k) = cx^k + \mu(\mathcal{A}x^k - b)$, and letting $\| y \| = (\sum_j y_j^2)^{1/2}$ denote the Euclidean norm of the vector y, we can solve for the step length and find that

$$\theta_k = \frac{L^* - L(\mu^k)}{\| \mathcal{A}x^k - b \|^2}.$$

Since we do not know the optimal objective function value L^* of the Lagrangian multiplier problem (after all, that's what we are trying to find), practitioners of Lagrangian relaxation often use the following popular heuristic for selecting the step length:

$$\theta_k = \frac{\lambda_k[\mathrm{UB} - L(\mu^k)]}{\| \mathcal{A}x^k - b \|^2}.$$

In this expression, UB is an upper bound on the optimal objective function value z^* of the problem (P), and so an upper bound on L^* as well, and λ_k is a scalar chosen (strictly) between 0 and 2. Initially, the upper bound is the objective function value of any known feasible solution to the problem (P). As the algorithm proceeds, if it generates a better (i.e., lower cost) feasible solution, it uses the objective function value of this solution in place of the upper bound UB. Usually, practitioners choose the scalars λ_k by starting with $\lambda_k = 2$ and then reducing λ_k by a factor of 2 whenever the best Lagrangian objective function value found so far has failed to increase in a specified number of iterations. Since this version of the algorithm has no convenient stopping criteria, practitioners usually terminate it after it has performed a specified number of iterations.

The rationale for these choices of the step size and the convergence proof of the subgradient method would take us beyond the scope of our coverage. In passing, we might note that the subgradient optimization procedure is not the only way to solve the Lagrangian multiplier problem: practitioners have used a number of other heuristics, including methods known as *multiplier ascent methods* that are tailored for special problems. Since we merely wish to introduce some of the basic concepts of Lagrangian relaxation and to indicate some of the essential methods used to solve the Lagrangian multiplier problem, we will not discuss these alternative methods.

Subgradient Optimization and Inequality Constraints

As we noted earlier in this section, if we apply Lagrangian relaxation to a problem with constraints $\mathcal{A}x \leq b$ stated in inequality form instead of the equality constraints, the Lagrange multipliers μ are constrained to be nonnegative. The update formula $\mu^{k+1} = \mu^k + \theta_k(\mathcal{A}x^k - b)$ might cause one or more of the components μ_i of μ to become negative. To avoid this possibility, we modify the update formula as follows:

$$\mu^{k+1} = [\mu^k + \theta_k(\mathcal{A}x^k - b)]^+.$$

In this expression, the notation $[y]^+$ denotes the "positive part" of the vector y; that is, the ith component of $[y]^+$ equals the maximum of 0 and y_i. Stated in another way, if the update formula $\mu^{k+1} = \mu^k + \theta_k(\mathcal{A}x^k - b)$ would cause the ith component of μ_i to be negative, then we simply set the value of this component to be zero. We then implement all the other steps of the subgradient procedure (i.e., the choice of

the step size θ at each step and the solution of the Lagrangian subproblems) exactly the same as for problems with equality constraints. For problems with both equality and inequality constraints, we use a straightforward mixture of the equality and inequality versions of the algorithm: whenever the update formula for the Lagrange multipliers would cause any component μ_i of μ corresponding to an inequality constraint to become negative, we set the value of that multiplier to be zero.

Let us illustrate the subgradient method for inequality constraints on our constrained shortest path example. Suppose that we start to solve our constrained shortest path problem at $\mu^0 = 0$ with $\lambda^0 = 0.8$ and with UB = 24, the cost corresponding to the shortest path 1–3–5–6 joining nodes 1 and 6. Suppose that we choose to reduce the scalar λ_k by a factor of 2 whenever three successive iterations at a given value of λ_k have not improved on the best Lagrangian objective function value $L(\mu)$. As we have already noted, the solution x^0 to the Lagrangian subproblem with $\mu = 0$ corresponds to the path P = 1–2–4–6, the Lagrangian subproblem has an objective function value of $L(0) = 3$, and the subgradient $\mathcal{A}x^0 - b$ at $\mu = 0$ is $(t_P - 14) = 18 - 14 = 4$. So at the first step, we choose

$$\theta_0 = 0.8(24 - 3)/16 = 1.05,$$

$$\mu^1 = [0 + 1.05(4)]^+ = 4.2.$$

For this value of the Lagrange multiplier, from Figure 16.3, we see that the path P = 1–3–2–5–6 solves the Lagrangian subproblem; therefore, $L(4.2) = 15 + 4.2(10) - 4.2(14) = 15 - 16.8 = -1.8$, and $\mathcal{A}x^1 - b$ equals $(t_P - 14) = 10 - 14 = -4$. Since the path 1–3–2–5–6 is feasible, and its cost of 15 is less than UB, we change UB to value 15. Therefore,

$$\theta_1 = 0.8(15 + 1.8)/16 = 0.84,$$

$$\mu^2 = [4.2 + 0.84(-4)]^+ = 0.84.$$

From iterations 2 through 5, the shortest paths alternate between the paths 1–2–4–6 and 1–3–2–5–6. At the end of the fifth iteration, the algorithm has not improved upon (increased) the best Lagrangian objective function value of 6.36 for three iterations, so we reduce λ_k by a factor of 2. In the next 7 iterations the shortest paths are the paths 1–2–5–6, 1–3–5–6, 1–3–2–5–6, 1–3–2–5–6, 1–2–5–6, 1–3–5–6, and 1–3–2–5–6. Once again for three consecutive iterations, the algorithm has not improved the best Lagrangian objective function value, so we decrease λ_k by a factor of 2 to value 0.2. From this point on, the algorithm chooses either path 1–3–2–5–6 or path 1–2–5–6 as the shortest path at each step. Figure 16.5 shows the first 33 iterations of the subgradient algorithm. As we see, the Lagrangian objective function value is converging to the optimal value $L^* = 7$ and the Lagrange multiplier is converging to its optimal value of $\mu^* = 2$.

Note that for this example, the optimal multiplier objective function value of $L^* = 7$ is strictly less than the length of the shortest constrained path, which has value 13. In these instances, we say that the Lagrangian relaxation has a *duality (relaxation) gap*. To solve problems with a duality gap to completion (i.e., to find an optimal solution and a guarantee that it is optimal), we would apply some form of enumeration procedure, such as branch and bound, using the Lagrangian lower bound to help reduce the amount of concentration required.

k	μ^k	$t_p - T$	$L(\mu^k)$	λ_k	θ_k
0	0.0000	4	3.0000	0.80000	1.0500
1	4.2000	−4	−1.8000	0.80000	0.8400
2	0.8400	4	6.3600	0.80000	0.4320
3	2.5680	−4	4.7280	0.80000	0.5136
4	0.5136	4	5.0544	0.80000	0.4973
5	2.5027	−4	4.9891	0.40000	0.2503
6	1.5016	1	6.5016	0.40000	3.3993
7	4.9010	−6	−5.4059	0.40000	0.2267
8	3.5406	−4	0.8376	0.40000	0.3541
9	2.1244	−4	6.5026	0.40000	0.2124
10	1.2746	1	6.2746	0.40000	3.4902
11	4.7648	−6	−4.5886	0.40000	0.2177
12	3.4589	−4	1.1646	0.20000	0.1729
13	2.7671	−4	3.9316	0.20000	0.1384
14	2.2137	−4	6.1453	0.20000	0.1107
15	1.7709	1	6.7709	0.20000	1.6458
16	3.4167	−4	1.3330	0.20000	0.1708
17	2.7334	−4	4.0664	0.20000	0.1367
18	2.1867	−4	6.2531	0.10000	0.0547
19	1.9680	1	6.9680	0.10000	0.8032
20	2.7712	−4	3.9150	0.10000	0.0693
21	2.4941	−4	5.0235	0.10000	0.0624
22	2.2447	−4	6.0212	0.05000	0.0281
23	2.1325	−4	6.4701	0.05000	0.0267
24	2.0258	−4	6.8966	0.05000	0.0253
25	1.9246	1	6.9246	0.00250	0.0202
26	1.9447	1	6.9447	0.00250	0.0201
27	1.9649	1	6.9649	0.00250	0.0201
28	1.9850	1	6.9850	0.00250	0.0200
29	2.0050	−4	6.9800	0.00250	0.0013
30	2.0000	−4	7.0000	0.00250	0.0012
31	1.9950	1	6.9950	0.00250	0.0200
32	2.0150	−4	6.9400	0.00250	0.0013
33	2.0100	−4	6.9601	0.00125	0.0006

Figure 16.5 Subgradient optimization for a constrained shortest path problem.

16.4 *LAGRANGIAN RELAXATION AND LINEAR PROGRAMMING*

In this section we discuss several theoretical properties of the Lagrangian relaxation technique. As we have noted earlier in Section 16.2, the primary use of the Lagrangian relaxation technique is to obtain lower bounds on the objective function values of (discrete) optimization problems. By relaxing the integrality constraints in the integer programming formulation of a discrete optimization problem, thereby

This side constraint specifies a flow relationship between several of the arcs in the network flow model. Relaxing this constraint and using Lagrangian relaxation provides us with one algorithmic approach for solving this problem. The algorithmic procedure for applying Lagrangian relaxation to the general network flow model with side constraints is essentially the same as the procedure we have discussed for the constrained shortest path problem: we associate nonnegative Lagrange multipliers μ with the side constraints $\mathcal{A}x \leq b$ and bring them into the objective function to produce the network flow subproblem

$$\text{minimize}\{cx + \mu(\mathcal{A}x - b) : \mathcal{N}x = q, l \leq x \leq u\},$$

and then solve a sequence of these problems with different values of the Lagrange multipliers μ which we update using the subgradient optimization technique. For each choice of the Lagrangian multiplier on this constraint, the Lagrangian subproblem is a network flow problem. In Exercise 9.9 we show that we can actually solve this special case of network flows with side constraints much more efficiently by solving a polynomial sequence of network flow problems.

Application 16.2 Traveling Salesman Problem

The traveling salesman problem is perhaps the most famous problem in all of network and combinatorial optimization: Its simplicity and yet its difficulty have made it an alluring problem that has attracted the attention of many noted researchers over a period of several decades. The problem is deceptively easy to state: Starting from his home base, node 1, a salesman wishes to visit each of several cities, represented by nodes 2, . . . , n, exactly once and return home, doing so at the lowest possible travel cost. We will refer to any feasible solution to this problem as a *tour* (of the cities).

The traveling salesman problem is a generic core model that captures the combinatorial essence of most routing problems and, indeed, most other routing problems are extensions of it. For example, in the classical vehicle routing problem, a set of vehicles, each with a fixed capacity, must visit a set of customers (e.g., grocery stores) to deliver (or pick up) a set of goods. We wish to determine the best possible set of delivery routes. Once we have assigned a set of customers to a vehicle, that vehicle should take the minimum cost tour through the set of customers assigned to it; that is, it should visit these customers along an optimal traveling salesman tour.

The traveling salesman problem also arises in problems that on the surface have no connection with routing. For example, suppose that we wish to find a sequence for loading jobs on a machine (e.g., items to be painted), and that whenever the machine processes job i after job j, we must reset the machine (e.g., clear the dies of the colors of the previous job), incurring a setup time c_{ij}. Then in order to find the processing sequence that minimizes the total setup time, we need to solve a traveling salesman problem—the machine, which functions as the "salesman," needs to "visit" the jobs in the most cost-effective manner.

There are many ways to formulate the traveling salesman problem as an optimization model. We present a model with an embedded (directed) network flow structure. Exercises 16.21 and 16.23 consider other modeling approaches. Let c_{ij}

denote the cost of traveling from city i to city j and let y_{ij} be a zero–one variable, indicating whether or not the salesman travels from city i to city j. Moreover, let us define flow variables x_{ij} on each arc (i, j) and assume that the salesman has $n - 1$ units available at node 1, which we arbitrarily select as a "source node," and that he must deliver 1 unit to each of the other nodes. Then the model is

$$\text{Minimize} \quad \sum_{(i,j)\in A} c_{ij}y_{ij} \tag{16.3a}$$

subject to

$$\sum_{1\le j\le n} y_{ij} = 1 \qquad \text{for all } i = 1, 2, \ldots, n, \tag{16.3b}$$

$$\sum_{1\le i\le n} y_{ij} = 1 \qquad \text{for all } j = 1, 2, \ldots, n, \tag{16.3c}$$

$$\mathcal{N}x = b, \tag{16.3d}$$

$$x_{ij} \le (n - 1)y_{ij} \qquad \text{for all } (i, j) \in A, \tag{16.3e}$$

$$x_{ij} \ge 0 \qquad \text{for all } (i, j) \in A, \tag{16.3f}$$

$$y_{ij} = 0 \text{ or } 1 \qquad \text{for all } (i, j) \in A. \tag{16.3g}$$

To interpret this formulation, let $A' = \{(i, j) : y_{ij} = 1\}$ and let $A'' = \{(i, j) : x_{ij} > 0\}$. The constraints (16.3b) and (16.3c) imply that exactly one arc of A' leaves and enters any node i; therefore, A' is the union of node disjoint cycles containing all of the nodes of N. In general, any integer solution satisfying (16.3b) and (16.3c) will be the union of disjoint cycles; if any such solution contains more than one cycle, we refer to each of the cycles as subtours, since they pass through only a subset of the nodes. Figure 16.8 gives an example of a subtour solution to the constraints (16.3b) and (16.3c).

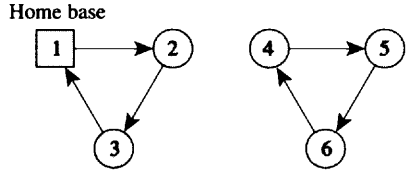

Figure 16.8 Infeasible solution for the traveling salesman problem containing subtours.

Constraint (16.3d) ensures that A'' is connected since we need to send 1 unit of flow from node 1 to every other node via arcs in A''. The "forcing" constraints (16.3e) imply that A'' is a subset of A'. [Notice that since no arc need ever carry more than $(n - 1)$ units of flow, the forcing constraint for arc (i, j) is redundant if $y_{ij} = 1$.] These conditions imply that the arc set A' is connected and so cannot contain any subtours. We conclude that the formulation (16.3) is a valid formulation for the traveling salesman problem.

One of the nice features of this formulation is that we can apply Lagrangian relaxation to it in several ways. For example, suppose that we attach Lagrange multipliers $\mu_{ij} \ge 0$ with the forcing constraints (16.3e) and bring them into the objective function, giving the Lagrangian objective function

$$\text{Minimize} \quad \sum_{(i,j)\in A} [c_{ij} - (n-1)\mu_{ij}]y_{ij} + \sum_{(i,j)\in A} \mu_{ij}x_{ij},$$

and leaving (16.3b)–(16.3d), (16.3f), and (16.3g) as constraints in the Lagrangian subproblem. Note that nothing in this Lagrangian subproblem couples the variables y_{ij} and x_{ij}. Therefore, the subproblem decomposes into two separate subproblems: (1) an assignment problem in the variables y_{ij}, and (2) a minimum cost flow problem in the variables x_{ij}. So for any choice of the Lagrangian multipliers μ, we solve two network flow subproblems; by using subgradient optimization we can find the best lower bound and optimal values of the multipliers. By relaxing other constraints in this model, or by applying Lagrangian relaxation to other formulations of the traveling salesman problem, we could define other network flow subproblems (see Exercise 16.19).

Application 16.3 Vehicle Routing

The vehicle routing problem is a generic model that practitioners encounter in many problem settings including the delivery of consumer products to grocery stores, the collection of money from vending machines and telephone coin boxes, and the delivery of heating oil to households. As we have noted earlier in this section, the vehicle routing problem is a generalization of the traveling salesman problem.

The vehicle routing problem is easy to state: Given (1) a fleet of K capacitated vehicles domiciled at a common depot, say node 1, (2) a set of customer sites $j = 2, 3, \ldots, n$, each with a prescribed demand d_j, and (3) a cost c_{ij} of traveling from location i to location j, what is the minimum cost set of routes for delivering (picking up) the goods to the customer sites? We assume that the vehicle fleet is homogeneous and that each vehicle has a capacity of u units.

There are many different variants on this core vehicle routing problem. For example, the vehicle fleet might be nonhomogeneous, each vehicle route might have a total travel time restriction, or deliveries for each customer might have time window restrictions (earliest and latest delivery times). We illustrate the use of Lagrangian relaxation by considering only the basic model, which we formulate with decision variables x_{ij}^k indicating whether ($x_{ij}^k = 1$) or not ($x_{ij}^k = 0$) we dispatch vehicle k on arc (i, j) and y_{ij} indicating whether some vehicle travels on arc (i, j):

$$\text{Minimize} \quad \sum_{1\le k\le K} \sum_{(i,j)\in A} c_{ij}x_{ij}^k \tag{16.4a}$$

subject to

$$\sum_{1\le k\le K} x_{ij}^k = y_{ij}, \tag{16.4b}$$

$$\sum_{1\le j\le n} y_{ij} = 1 \qquad \text{for } i = 2, 3, \ldots, n, \tag{16.4c}$$

$$\sum_{1\le i\le n} y_{ij} = 1 \qquad \text{for } j = 2, 3, \ldots, n, \tag{16.4d}$$

$$\sum_{1\le j\le n} y_{1j} = K, \tag{16.4e}$$

$$\sum_{1\le i\le n} y_{i1} = K, \tag{16.4f}$$