Chapter 2
Examples

This chapter shows how an integrated approach to optimization plays itself out in some concrete cases. It presents several small examples that cover a wide range of application areas. It each case, it formulates a model that is appropriate for integrate1d solution, and in most cases it carries the solution procedure to completion.

The first example—a simple freight transfer problem—demonstrates how inference methods from CP and relaxation methods from MILP can work together to accelerate branching search. A production planning example then shows how discrete choices can be formulated in an integrated modeling environment to result in much faster solution. An employee scheduling example demonstrates the modeling power of metaconstraints.

A fourth example shows how inference and relaxation can be profitably combined in continuous global optimization. A product configuration problem illustrates how high-level modeling can tell the solver how to combine inference and relaxation. A machine scheduling problem shows how integer programming and constraint programming can be integrated in a framework of logic-based Benders decomposition. A final and more complex example deals with communications network routing and frequency assignment. It combines elements of network flow programming and constraint programming using another form of decomposition.

The chapter begins with some basic definitions, followed a general description of the search-infer-and-relax solution process that is illustrated by the examples.

J.N. Hooker, *Integrated Methods for Optimization*, International Series in Operations Research & Management Science 170, DOI 10.1007/978-1-4614-1900-6_2,

2.1 Basic Definitions

For the purposes of this book, an optimization problem can be written

$$\begin{array}{l} \min\,(\text{or } \max)\ f(x) \\ \mathcal{C}(x) \\ x \in D \end{array} \tag{2.1}$$

where $f(x)$ is a real-valued function of variable x and D is the *domain* of x. The function $f(x)$ is to be minimized (or maximized) subject to a set $\mathcal{C}$ of constraints, each of which is either satisfied or violated by any given $x \in D$. Generally, x is a tuple $(x_1, \ldots, x_n)$ and D is a Cartesian product $D_1 \times \cdots \times D_n$, where each $x_j \in D_j$. The notation $\mathcal{C}(x)$ means that x satisfies all the constraints in $\mathcal{C}$.

Adopting terminology from mathematical programming, any $x \in D$ is a *solution* of (2.1). Solution x is a *feasible* solution if $\mathcal{C}(x)$, and the *feasible set* of (2.1) is the set of feasible solutions. A feasible solution x^* is *optimal* if $f(x^*) \leq f(x)$ for all feasible x. An *infeasible* problem is one with no feasible solution. If (2.1) is infeasible, it is convenient to say that it has optimal value ∞ (or $-\infty$ in the case of a maximization problem). The problem is *unbounded* if there is no lower bound on $f(x)$ for feasible values of x, in which case the optimal value is $-\infty$ (or ∞ for maximization).

It is assumed throughout this book that (2.1) is either infeasible, unbounded, or has a finite optimal value. Thus, such problems as minimizing x subject to $x > 0$ are not considered. An optimization problem is considered to be *solved* when an optimal solution is found, or when the problem is shown to be unbounded or infeasible. In incomplete search methods that do not guarantee an optimal solution, the problem is solved when a solution is found that is acceptable in some sense, or when the problem is shown to be unbounded or infeasible.

A constraint G can be *inferred* from $\mathcal{C}$ if any $x \in D$ that satisfies $\mathcal{C}(x)$ also satisfies G. Equivalently, $\mathcal{C}$ *implies* G, or G is *valid* for $\mathcal{C}$. A *relaxation* R of the minimization problem (2.1) is obtained by dropping constraints and/or replacing the objective function $f(x)$ with a lower bounding function $f'(x)$. That is, any $x \in D$ that is feasible in (2.1) is feasible in R and satisfies $f'(x) \leq f(x)$. When one is minimizing, the optimal value of a relaxation is always a lower bound on the optimal value of the original problem.

2.2 The Solution Process

Search, inference, and relaxation interact to provide a general scheme for solving (2.1). The search procedure solves a series of *restrictions* or special cases of the problem. The best solution of a restriction is accepted as a solution of the original problem. Inference and relaxation provide opportunities to exploit problem structure, a key element of any successful approach to solving combinatorial problems. These ideas are more formally developed in Chapters 5, 6, and 7. Only a brief overview is given here.

Restrictions are obtained by adding constraints to the problem. The rationale for searching over restrictions is that they may be easier to solve than the original problem, even when there are many of them. Branching search methods, for example, divide the feasible set into smaller and smaller subsets by branching on alternatives. Each subset corresponds to a restriction of the original problem. Well-known examples of branching search include the branch-and-cut algorithms of MILP solvers and the branch-and-infer methods of CP solvers. Benders decomposition also enumerates restrictions by solving a sequence of subproblems in which certain variables are fixed. Local search methods examine a sequence of neighborhoods, each of which defines a restriction of the problem.

Search can often be accelerated by *inference*—that is, by inferring new constraints from the constraint set. The new constraints are then added to the problem, which may ease solution by describing the feasible set more explicitly. Common forms of inference are domain filtering in CP, cutting-plane generation in MILP, and Benders cuts in Benders decomposition. Inferred constraints may also result in a stronger relaxation, as in the case of cutting planes and Benders cuts.

Inference is most effective when it exploits problem structure. When a group of constraints has special characteristics, the model can indicate this by combining the constraints into a single *metaconstraint*, known as a *global constraint* in constraint programming. This allows inference algorithms to exploit the global structure of the group when filtering domains or generating valid cuts. Developing special-purpose inference procedures has been a major theme of research in CP.

Relaxation is the third element of the solution scheme. Like a restriction, a relaxation of the problem may be easier to solve than the original. For instance, an MILP problem becomes much easier to

solve if one relaxes it by allowing the integer-valued variables to take any real value.

Solving a relaxation can be useful in several ways. Its solution may happen to be a solution of the original problem. Even if not, its solution may provide a clue to where one might find a solution of the original problem and therefore help guide the search. In addition, the optimal value of the relaxation provides a lower bound on the optimal value of the original problem (when one is minimizing). This is particularly useful in branching search, where bounds can often help prune the search tree.

Relaxation also provides a valuable opportunity to exploit special structure in individual constraints or groups of constraints, and this has been a perennial theme of the optimization literature. For example, strong cutting planes can be inferred from sets of inequalities that define certain types of polyhedra. One can use metaconstraints to indicate which inequalities have special structure.

2.3 Freight Transfer

A simple freight transfer problem illustrates, at an elementary level, how inference and relaxation can interact with branching search. Forty-two tons of freight must be conveyed overland. The shipper has a fleet of trucks in four sizes, with three vehicles of each size (Table 2.1). The eight available loading docks must accommodate all the trucks used, because the trucks must be loaded simultaneously. Due to the shape of the loading area, three loading docks must be allocated to the largest trucks even if only one or two of them are loaded. The problem is to select trucks to carry the freight at minimum cost.

Table 2.1 Data for a small instance of a freight transfer problem.

Truck type	*Number available*	*Capacity (tons)*	*Cost per truck*
1	3	7	90
2	3	5	60
3	3	4	50
4	3	3	40

2.3.1 Formulating the Problem

If variable x_i indicates the number of trucks of type i loaded, the requirement that 42 tons be transported can be written

$$7x_1 + 5x_2 + 4x_3 + 3x_4 \geq 42 \tag{2.2}$$

The loading dock constraints can be written

$$\begin{aligned} &x_1 + x_2 + x_3 + x_4 \leq 8 \\ &(1 \leq x_1 \leq 2) \Rightarrow (x_2 + x_3 + x_4 \leq 5) \end{aligned} \tag{2.3}$$

where $\Rightarrow$ means "implies." The problem can therefore be formulated

$$\begin{aligned} &\text{integerLinear: } \begin{cases} \min\ 90x_1 + 60x_2 + 50x_3 + 40x_4 \\ 7x_1 + 5x_2 + 4x_3 + 3x_4 \geq 42 \qquad (a) \\ x_1 + x_2 + x_3 + x_4 \leq 8 \qquad (b) \end{cases} \\ &\text{conditional: } (1 \leq x_1 \leq 2) \Rightarrow (x_2 + x_3 + x_4 \leq 5) \\ &\text{domains: } x_i \in \{0, 1, 2, 3\}, \quad i = 1, \ldots, 4 \end{aligned} \tag{2.4}$$

In accord with the spirit of revealing problem structure, the constraints are grouped by type to provide the solver guidance as to how to process them. Since the objective function becomes an integer linear inequality whenever it is bounded, it is grouped with the integer linear constraints to form an integer linear metaconstraint. An optimal solution is $(x_1, \ldots, x_4) = (3, 2, 2, 1)$, with a minimum cost of 530.

The problem may be solved by a branching algorithm that uses inference and relaxation. The following sections first show how to carry out the inference and relaxation steps and then how to conduct the search.

2.3.2 Inference: Bounds Propagation

Domain filtering removes values from a variable domain when the variable cannot take those values in any feasible solution. The reduced domains inferred from one constraint can be used as a starting point for domain filtering in another constraint (a form of *constraint propagation*). One can, in principle, cycle through the constraints in this fashion until no further filtering is possible. This allows one to draw

some inferences that do not follow from any single constraint, even though the constraints are processed individually.

A type of domain filtering known as *bounds propagation* is useful for the freight transport problem. Focusing first on constraint (2.2), one can "solve" it for variable x_1, for example, to obtain

$$x_1 \geq \frac{42 - 5x_2 - 4x_3 - 3x_4}{7} \geq \frac{42 - 5 \cdot 3 - 4 \cdot 3 - 3 \cdot 3}{7} = \frac{6}{7}$$

where the second inequality is due to the fact that each $x_j \in \{0, 1, 2, 3\}$. Because x_1 must be integral, one can round up the bound to obtain $x_1 \geq 1$. The domain of x_1 is therefore reduced from $\{0, 1, 2, 3\}$ to $\{1, 2, 3\}$. The same procedure can be applied to the other variables, but as it happens, no further domain reductions are possible. The smaller domain for x_1 can now be propagated to the second constraint (2.3), which by similar reasoning implies that $x_1 \leq 8$ and $x_j \leq 7$ for $j = 2, 3, 4$. Unfortunately, this does not further reduce any of the domains. If it did, one could cycle back to the first constraint and repeat the procedure.

If the reduced domain of x_1 were a subset of $\{1, 2\}$, one could infer from the implication constraint that $x_2 + x_3 + x_4 \leq 5$ and perhaps reduce domains further. This condition is not initially satisfied, but it may become satisfied during the search.

2.3.3 Inference: Valid Inequalities

In addition to deducing smaller domains, one can deduce *valid inequalities* or *cutting planes* from the knapsack constraints. The inferred inequalities are added to the original constraint set in order to produce a stronger continuous relaxation.

One type of cutting plane, known as a *general integer knapsack cut*, can be inferred from constraints like (2.2). Note that the first two terms $7x_1, 5x_2$ of (2.2) cannot by themselves satisfy the inequality, even if x_1 and x_2 are set to their largest possible value of 3. To satisfy the inequality, one must have

$$4x_3 + 3x_4 \geq 42 - 7 \cdot 3 - 5 \cdot 3$$

which implies

$$x_3 + x_4 \geq \left\lceil \frac{42 - 7 \cdot 3 - 5 \cdot 3}{4} \right\rceil = \left\lceil \frac{6}{4} \right\rceil = 2$$

The inequality $x_3 + x_4 \geq 2$ is an integer knapsack cut.

Because the first two terms of (2.2) cannot satisfy the inequality by themselves, the index set $\{1, 2\}$ is a *packing*. In fact, it is a *maximal* packing, because no proper superset is a packing. There are four maximal packings for (2.2), each of which gives rise to an integer knapsack cut:

$$\begin{aligned} \{1,2\}: &\quad x_3 + x_4 \geq 2 \\ \{1,3\}: &\quad x_2 + x_4 \geq 2 \\ \{1,4\}: &\quad x_2 + x_3 \geq 3 \\ \{2,3,4\}: &\quad x_1 \geq 1 \end{aligned} \tag{2.5}$$

These cuts are implied by (2.2) because any integral $x = (x_1, \ldots, x_4)$ that satisfies (2.2) must satisfy (2.5). Nonmaximal packings also give rise to integer knapsack cuts, and they may be nonredundant. For example, the packing $\{2, 3\}$ produces the cut $x_1 + x_4 \geq 3$, which is not redundant of (2.5) or any other knapsack cut.

Knapsack cuts can sometimes be strengethened using domains. For example, the packing $\{2\}$ gives rise to the cut $x_1 + x_3 + x_4 \geq 4$, which can be strengthened to $x_1 + x_3 + x_4 \geq 5$. This is because $x_1 \leq 3$, and as a result the largest possible left-hand side of (2.2) when $x_1 + x_3 + x_4 = 4$ is 40. It is therefore neessary to have $x_1 + x_3 + x_4 \geq 5$.

The fourth cut in (2.5) duplicates the bound $x_1 \geq 1$ already obtained from bounds propagation. In fact, it is easy to see that any propagated bound $x_j \geq L$ can be obtained from the knapsack cut corresponding to the maximal packing that consists of the remaining variables. Knapsack cuts corresponding to maximal packings therefore dominate propagated bounds. However, it is much more costly to generate all maximal packings than to propagate bounds.

2.3.4 Relaxation: Linear Programming

A continuous relaxation of the problem instance (2.4) can be obtained from the inequality constraints by allowing each variable x_i to take any real value in the range between its lowest and highest value. One can also add the knapsack cuts (2.5) before relaxing the problem. This yields the relaxation below:

$$\text{linear: } \begin{cases} \min\ 90x_1 + 60x_2 + 50x_3 + 40x_4 & \\ 7x_1 + 5x_2 + 4x_3 + 3x_4 \geq 42 & (a) \\ x_1 + x_2 + x_3 + x_4 \leq 8 & (b) \\ x_3 + x_4 \geq 2 & (c) \\ x_2 + x_4 \geq 2 & (d) \\ x_2 + x_3 \geq 3 & (e) \\ L_i \leq x_i \leq U_i, \quad i = 1, \ldots, 4 & \end{cases} \tag{2.6}$$

$$\text{domains: } x_i \in \mathbb{R},\ i = 1, \ldots, 4$$

where initially $(L_1, U_1) = (1, 3)$ and $(L_i, U_i) = (0, 3)$ for $i = 2, 3, 4$. This relaxed problem can be easily solved by linear programming. An optimal solution is $x = (x_1, \ldots, x_4) = (2\frac{1}{3}, 3, 2\frac{2}{3}, 0)$, which has cost $523\frac{1}{3}$. This solution is not feasible in the original problem because it is not integral, but it provides a lower bound on the optimal cost. The freight cannot be transported for less than $523\frac{1}{3}$.

The knapsack cuts (c)–(e) in (2.6) make the relaxation tighter because they "cut off" solutions that satisfy the other constraints. For example, the solution $x = (3, 3, 1.5, 0)$ satisfies the other constraints but violates (c). As it happens, these particular knapsack cuts do not improve the lower bound, because the optimal value of the relaxation is $523\frac{1}{3}$ without them. Yet if the knapsack cut $x_1 + x_4 \geq 3$ had been included in the relaxation, it would have cut off the solution $x = (2\frac{1}{3}, 3, 2\frac{2}{3}, 0)$ and provided the tighter bound of 525.

2.3.5 Branching Search

A search tree for problem instance (2.4) appears in Figure 2.1. Each node of the tree below the root corresponds to a restriction of the original problem. A restriction is processed by applying inference methods (bounds propagation and cut generation), then solving a continuous relaxation, and finally branching if necessary.

Maximal packings are generated only at the root node, due to the cost of identifying them at every node. Thus the same three knapsack cuts appear in the relaxation at every node of the tree.

Bounds propagation applied to the original problem at the root node reduces the domain of x_1 to $\{1, 2, 3\}$, as described earlier. Figure 2.1 shows the resulting domains in braces. Next, three knapsack cuts (2.5) are generated (note that the knapsack cut $x_1 \geq 1$ is already implicit

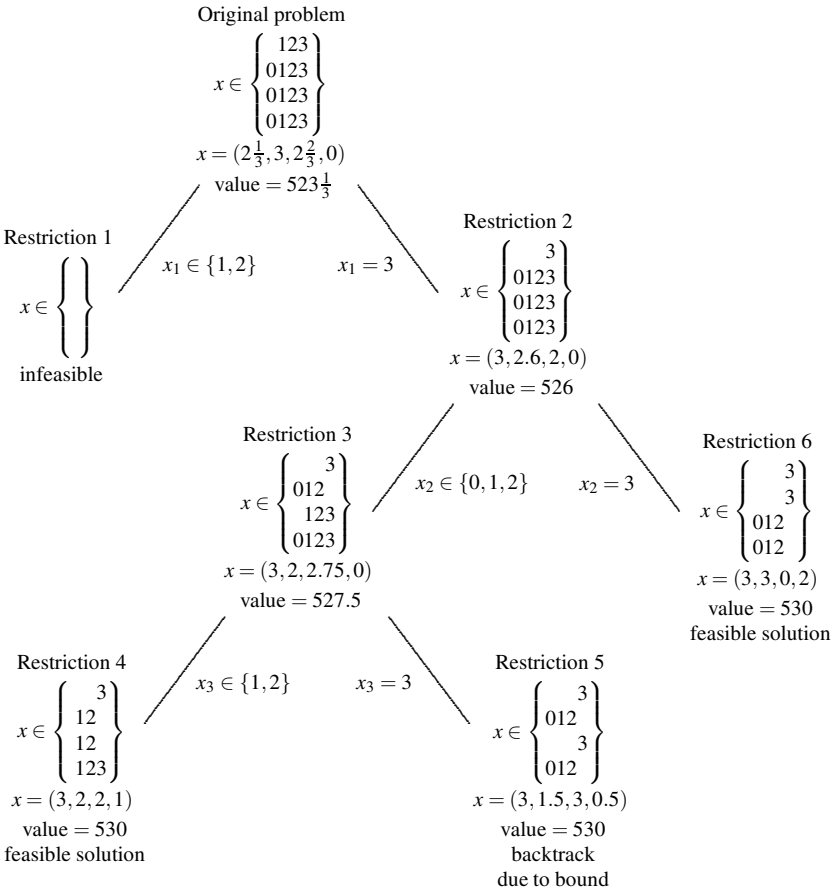

Fig. 2.1 Branch-and-relax tree for the freight transfer problem. Each node of the tree shows, in braces, the filtered domains after domain reduction. The rows inside the braces correspond to $x_1, \ldots, x_4$. The solution of the continuous relaxation appears immediately below the domains.

in x_1's domain). They are added to the constraint set in order to obtain the continuous relaxation (2.6), whose optimal solution $x = (2\frac{1}{3}, 3, 2\frac{2}{3}, 0)$ is shown at the root node.

This solution is infeasible in the original problem because x_1 and x_3 do not belong to their respective domains (they are nonintegral). It is therefore necessary to branch on one of the domain constraints $x_1 \in \{1,2,3\}$, $x_3 \in \{0,1,2,3\}$. Branching on the first splits the domain into $\{1,2\}$ and $\{3\}$, as the solution value $2\frac{1}{3}$ of x_1 lies between 2 and 3. This generates restrictions 1 and 2.

The tree can be traversed in a depth-first manner. Moving first to restriction 1, where x_1's domain is $\{1, 2\}$, bounds propagation applied to the original inequality constraint yields domain $\{2, 3\}$ for x_2 and $\{1, 2, 3\}$ for x_3 and x_4. Because the domain of x_1 activates the conditional constraint, one can infer $x_2 + x_3 + x_4 \leq 5$ and further reduce the domains of x_3 and x_4 to $\{1, 2\}$. One more round of propagation on these two inequalities reduces all domains to the empty set. Restriction 1 is therefore infeasible, and there is no need to solve the relaxation or to branch.

Moving now to restriction 2, no further domain filtering is possible, and solution of the relaxation (2.6) yields $x = (3, 2.6, 2, 0)$. Branching on x_2 creates restrictions 3 and 6.

Continuing in a depth-first manner, restriction 3 is processed next. Inference yields the domains shown, and branching on x_3 produces restrictions 4 and 5. The continuous relaxation of restriction 4 has the integral solution $x = (3, 2, 2, 1)$, which is feasible in the restriction. It becomes the *incumbent* solution (the best feasible solution so far), and no branching is necessary.

Restriction 5 is processed next. Here, the relaxation has a nonintegral solution, but its optimal value 530 is no better than the value of the incumbent solution. Since 530 is a lower bound on the optimal value of any further restriction of restriction 5, there is no need to branch. The tree is therefore "pruned" at restriction 5, and the search proceeds to restriction 6. A branching search in which the tree is pruned in this manner is called *branch and bound*.

The continuous relaxation of restriction 6 has an integral solution, and there is no need to branch, thus completing the search. Because the solution is no better than the incumbent (in fact it is equally good), it and the incumbent solution $x = (3, 2, 2, 1)$ are optimal. The minimum cost of transporting the freight is 530.

Inference and relaxation work together in this example to reduce the solution time. Solutions of relaxations help to guide the branching. At restriction 1, inference alone proves infeasibility, and there is no need to branch. As it happens, solving the relaxation at this node would also prove infeasibility, but due to inference there is no need to incur the greater overhead of solving a relaxation. At restrictions 4 and 6, the relaxation (with the help of prior domain reduction) obtains feasible solutions, and no branching is necessary. If inference alone were used, one would be able to find feasible solutions only by branching until all

the domains are singletons. At restriction 5, the relaxation provides a bound that again obviates the necessity of further branching.

Exercises

2.1. Apply domain propagation to the inequalities

$$5x_1 + 4x_2 + 3x_3 \geq 18$$
$$2x_1 + 3x_2 + 4x_3 \leq 10$$

with initial domains $D_i = \{0, 1, 2\}$ for $i = 1, 2, 3$. Cycle through the inequalities until no further domain reduction is possible.

2.2. Identify all packings for inequality (2.2) using domains $x_j \in \{0, 1, 2, 3\}$, and write the corresponding knapsack cuts. *Hints.* Cuts corresponding to maximal packings appear in (2.5), but there are also nonmaximal packings. In addition, some of the cuts can be strengthened using domains.

2.3. Solve the problem of minimizing $3x_1 + 4x_2$ subject to $2x_1 + 3x_2 \geq 10$ and $x_1, x_2 \in \{0, 1, 2, 3\}$ using branch and relax without bounds propagation. Now, solve it using branch and infer without relaxation. Finally, solve it using branch and relax with propagation. Which results in the smallest search tree? (When using branch and infer, follow the common constraint programming practice of branching on the variable with the smallest domain.)

2.4. Write two integer linear inequalities (with initial domains specified) for which bounds propagation reduces domains more than minimizing and maximizing each variable subject to a continuous relaxation of the constraint set.

2.5. Write two integer linear inequalities (with domains specified) for which minimizing and maximizing each variable subject to a continuous relaxation of the constraint set reduces domains more than bounds propagation.

2.4 Production Planning

A very simple production planning problem illustrates how logical and continuous variables can interact. A manufacturing plant has three operating modes, each of which imposes different constraints. The objective is to decide in which mode to run the plant, and how much of each of two products to make, so as to maximize net income.

2.4.1 Formulating the Problem

Let x_A, x_B be the production levels of products A and B, respectively. In mode 0 the plant is shut down, and $x_A = x_B = 0$. In mode 1, it incurs a fixed cost of 35, and the production levels must satisfy the constraint $2x_A + x_B \leq 10$. Mode 2 incurs a fixed cost of 45 and the constraint $x_A + 2x_B \leq 10$ is imposed. The company earns a net income of 5 for each unit of product A manufactured, and 3 for each unit of product B.

A natural modeling approach is to let a Boolean variable δ_k be true when the plant runs in mode k. Then, the model is immediate:

$$\begin{aligned}
&\text{linear: } \max\ 5x_A + 3x_B - f \\
&\text{logic: } \delta_0 \vee \delta_1 \vee \delta_2 \\
&\text{conditional: } \begin{cases} \delta_0 \Rightarrow (x_A = x_B = f = 0) \\ \delta_1 \Rightarrow (2x_A + x_B \leq 10,\ f = 35) \\ \delta_2 \Rightarrow (x_A + 2x_B \leq 10,\ f = 45) \end{cases} \\
&\text{domains: } x_A, x_B \geq 0,\ \ \delta_k \in \{\text{true}, \text{false}\},\ k = 0, 1, 2
\end{aligned} \tag{2.7}$$

where variable f represents the fixed cost. The logic constraint means that at least one of the three Boolean variables must be true.

2.4.2 Relaxation

The model has an interesting relaxation because each production mode k enforces constraints that define a polyhedron in the space of the continuous variables x_A, x_B, f. So the projection of the feasible set onto this space is the union of the three polyhedra, illustrated in Figure 2.2. The best possible continuous relaxation of this set is its *convex hull*, which is itself a polyhedron and is also shown in the figure. The convex hull of a set is union of all line segments connecting any two points of the set.

There is a general procedure, given in Section 7.4.1, for writing a linear constraint set that describes the convex hull of a disjunction of linear systems. In this case, the convex hull description is

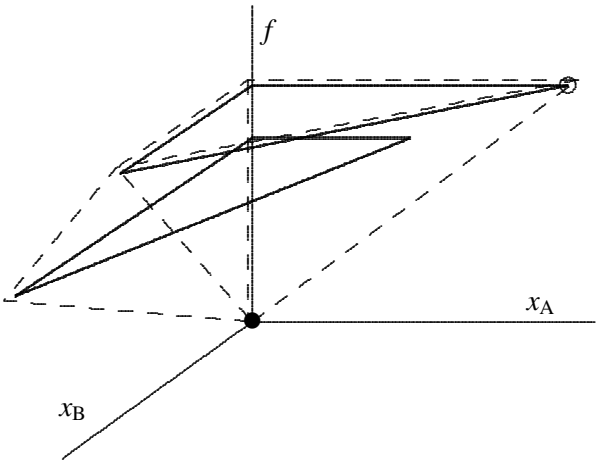

Fig. 2.2 Feasible set (two triangular areas and black circle) of a production planning problem, projected onto the space of the continuous variables. The convex hull is the volume inside the dashed polyhedron. The open circle marks the optimal solution.

$$
\begin{aligned}
& 2x_{A1} + x_{B1} \leq 10y_1 \\
& x_{A2} + 2x_{B2} \leq 10y_2 \\
& f \geq 35y_1 + 45y_2 \\
& x_A = x_{A1} + x_{A2}, \quad x_B = x_{B1} + x_{B2} \\
& y_0 + y_1 + y_2 = 1, \quad y_k \geq 0, \; k = 0, 1, 2 \\
& x_{Ak} \geq 0, \; x_{Bk} \geq 0, \quad k = 1, 2
\end{aligned}
\tag{2.8}
$$

where the variables y_k correspond to the Boolean variables δ_k, and fixing δ_k = true is equivalent to setting $y_k = 1$. Since there are new variables x_{Ak}, x_{Bk}, y_k in the relaxation, one should, strictly speaking, say that it describes a set whose projection onto x_A, x_B, f is the convex hull of (2.7)'s feasible set. A lower bound on the optimal value can now be obtained by minimizing $5x_A + 3x_B - f$ subject to (2.8).

To take advantage of this relaxation, the solver must somehow recognize that the feasible set in continuous space is a union of polyhedra. This can be done by collecting the constraints of (2.7) into a single *linear disjunction*, resulting in the model:

linear: $\max\ 5x_{\mathrm{A}} + 3x_{\mathrm{B}} - f$

linearDisjunction:

$$\begin{bmatrix} \delta_0 \\ x_{\mathrm{A}} = x_{\mathrm{B}} = 0 \\ f = 0 \end{bmatrix} \vee \begin{bmatrix} \delta_1 \\ 2x_{\mathrm{A}} + x_{\mathrm{B}} \le 10 \\ f = 35 \end{bmatrix} \vee \begin{bmatrix} \delta_2 \\ x_{\mathrm{A}} + 2x_{\mathrm{B}} \le 10 \\ f = 45 \end{bmatrix}$$

domains: $x_{\mathrm{A}}, x_{\mathrm{B}} \ge 0,\ \ \delta_k \in \{\text{true}, \text{false}\},\ k = 0, 1, 2$

The linear disjunction has precisely the same meaning as the logic and conditional constraints of (2.7). Its presence tells the solver to formulate a relaxation for the union of polyhedra described by its three disjuncts. In general, the user should be alert for metaconstraints that can exploit problem structure, although in this case an intelligent modeling system could automatically rewrite (2.7) as (2.8).

2.4.3 Branching Search

The problem can now be solved by branching on the Boolean variables. The relaxation of the problem at the root node has solution $(y_1, y_2) = (0, 1)$, with $(x_{\mathrm{A}}, x_{\mathrm{B}}, f) = (10, 0, 35)$. Because this solution is feasible in (2.7) with $(\delta_0, \delta_1, \delta_2) = (\text{false}, \text{false}, \text{true})$, there is no need to branch. One should operate the plant in mode 2 and make product A only.

In this problem, there is no branching because (2.8) is a convex hull relaxation not only for the corresponding linear disjunction but for the entire problem. In more complex problems, an auxiliary variable y_k may take a fractional value, in which case one can branch by setting $\delta_k = \text{true}$ and $\delta_k = \text{false}$.

2.4.4 Inference

Inference plays a role in such problems when the logical conditions become more complicated. Suppose, for example, that plant 1 can operate in two modes (indicated by Boolean variables δ_{1k} for $k = 0, 1$) and plant 2 in three modes (indicated by variables δ_{2k}). There is also a rule that if plant 1 operates in mode 1, then plant 2 cannot operate in mode 2. The model might be as follows (where $\neg$ means *not*).

$$\text{linear: } \max\ cx$$

$$\text{logic: } \begin{cases} \delta_{10} \vee \delta_{11} \\ \delta_{20} \vee \delta_{21} \vee \delta_{22} \\ \delta_{11} \rightarrow \neg\delta_{22} \end{cases}$$

$$\text{conditional: } \delta_{ik} \Rightarrow A^{ik}x \geq b^{ik}, \quad \text{all } i, k$$

$$\text{domains: } x \geq 0, \quad \delta_{ik} \in \{\text{true}, \text{false}\}$$

To extract as many linear disjunctions as possible, one can use the *resolution* algorithm (discussed in Section 6.4.2) to compute the *prime implications* of the logical formulas in the above model:

$$\begin{array}{ll} \delta_{10} \vee \delta_{11} & \delta_{10} \vee \neg\delta_{22} \\ \delta_{20} \vee \delta_{21} \vee \delta_{22} & \neg\delta_{11} \vee \delta_{20} \vee \delta_{21} \\ \neg\delta_{11} \vee \neg\delta_{22} & \delta_{10} \vee \delta_{20} \vee \delta_{21} \end{array}$$

These are the undominated disjunctions implied by the logical formulas. Three of the prime implications contain no negative terms, and they provide the basis for linear disjunctions. The model becomes:

$$\text{linear: } \max\ cx$$

$$\text{logic: } \begin{cases} \neg\delta_{11} \vee \neg\delta_{22} \\ \delta_{10} \vee \neg\delta_{22} \\ \neg\delta_{11} \vee \delta_{20} \vee \delta_{21} \end{cases}$$

$$\text{linear disjunction:}$$

$$\bigvee_{k \in \{0,1\}} \begin{bmatrix} \delta_{1k} \\ A^{1k}x \geq b^{1k} \end{bmatrix}$$

$$\bigvee_{k \in \{0,1,2\}} \begin{bmatrix} \delta_{2k} \\ A^{2k}x \geq b^{2k} \end{bmatrix}$$

$$\begin{bmatrix} \delta_{10} \\ A^{10}x \geq b^{10} \end{bmatrix} \vee \begin{bmatrix} \delta_{20} \\ A^{20}x \geq b^{20} \end{bmatrix} \vee \begin{bmatrix} \delta_{21} \\ A^{21}x \geq b^{21} \end{bmatrix}$$

$$\text{domains: } x \geq 0, \quad \delta_{ik} \in \{\text{true}, \text{false}\}$$

Again, an intelligent modeling system can carry out this process automatically.

Inference plays a further role as branching proceeds. When branching fixes a Boolean variable to true or false, it may be possible to

deduce that some other variables must be true or false, thus reducing their domains to a singleton. For example, if δ_{10} is fixed to false, then δ_{11} must be true and δ_{22} false. The resolution method draws all such inferences.

Exercises

2.6. Formulate the following problem with the help of conditional constraints. A lumber operation wishes to build temporary roads from forests 1 and 2 to sawmills A and B. Due to topography and environmental regulations, roads can be built only in certain combinations: (a) from 1 to A, 1 to B, and 2 to B; (b) from 1 to A, 1 to B, and 2 to A; (c) from 1 to A, 2 to A, and 2 to B. Each sawmill j requires d_j units of timber, the unit cost of shipping timber over a road from forest i to sawmill j is c_{ij}, and the fixed cost of a road from i to j is f_{ij}. Choose which roads to build, and how much timber to transport over each road, so as to minimize cost while meeting the sawmill demands. Now, formulate the problem without conditional constraints and with a disjunctive linear constraint.

2.7. Consider the problem

$$\begin{aligned}&\text{linear: } \min\ cx\\ &\text{logic: } \begin{cases} y_1 \vee y_2 \\ y_1 \rightarrow y_3 \end{cases}\\ &\text{conditional: } y_k \Rightarrow \left(A^k x \geq b^k\right),\ k = 1, 2, 3\end{aligned}$$

where each y_k is a Boolean variable. Write the problem without conditional constraints and using as many linear disjunctions as possible. *Hint:* The formula $y_1 \rightarrow y_3$ is equivalent to $\neg y_1 \vee y_3$.

2.5 Employee Scheduling

An employee scheduling problem is useful for introducing several of the modeling and inference techniques that have been developed in the constraint programming field. Since the objective is to find a feasible rather than an optimal solution, relaxation bounds do not play a role. Nonetheless, the integration of technologies is a key ingredient of the solution method, partly because the filtering algorithms rely on such optimization techniques as maximum flow algorithms and dynamic programming.

A certain hospital ward requires that a head nurse be on duty seven days a week, twenty-four hours a day. There are 3 eight-hour shifts, and on a given day each shift must be staffed by a different nurse. The schedule must be the same every week. Four nurses (denoted A, B, C, and D) are available, all of whom must work at least five days a week. Since there are 21 eight-hour periods a week, this implies that three nurses will work five days and one will work six days. For continuity, no shift should be staffed by more than two different nurses during the week.

Two additional rules reduce the burden of adjusting to shift changes. No employee is asked to work different shifts on two consecutive days; there must be at least one day off in between. Also, an employee who works shift 2 or 3 must do so at least two days in a row. Shift 1 is the daytime shift and requires less adjustment.

2.5.1 Formulating the Problem

There are two natural ways to think about an employee schedule. One, shown in Table 2.2, is to view it as assigning a nurse to each shift on each day. Another is to see it as assigning a shift (or day off) to each nurse on each day; this is illustrated by Table 2.3, in which shift 0 represents a day off. Either viewpoint gives rise to a different problem formulation, neither of which is convenient for expressing all the con-

Table 2.2 Employee scheduling viewed as assigning workers to shifts.

	Sun	*Mon*	*Tue*	*Wed*	*Thu*	*Fri*	*Sat*
Shift 1	A	B	A	A	A	A	A
Shift 2	C	C	C	B	B	B	B
Shift 3	D	D	D	D	C	C	D

Table 2.3 Employee scheduling viewed as assigning shifts to workers.

	Sun	*Mon*	*Tue*	*Wed*	*Thu*	*Fri*	*Sat*
Worker A	1	0	1	1	1	1	1
Worker B	0	1	0	2	2	2	2
Worker C	2	2	2	0	3	3	0
Worker D	3	3	3	3	0	0	3

straints of the problem. Fortunately, there is no need to choose between them. One can use both formulations in the same model and connect them with *channeling constraints*. This not only accommodates all the constraints but also makes propagation more effective.

To proceed with the first formulation, let variable w_{sd} be the nurse that is assigned to shift s on day d. Three of the scheduling requirements are readily expressed in this formulation, using metaconstraints that are well known to constraint programmers. One is the *all-different* constraint, alldiff(X), where X denotes a set $\{x_1, \ldots, x_n\}$ of variables. It simply requires that $x_1, \ldots, x_n$ take different values. It can express the requirement that three different nurses be scheduled each day:

$$\text{alldiff}(w_{\cdot d}), \text{ all } d$$

The notation $w_{\cdot d}$ refers to $\{w_{1d}, w_{2d}, w_{3d}\}$.

The *cardinality* constraint can be used to require that every nurse be assigned at least five days of work. The constraint in general is written

$$\text{cardinality}(X \mid v, \ell, u)$$

where $X = \{x_1, \ldots, x_n\}$ and v is an m-tuple $(v_1, \ldots, v_m)$ of values. $\ell = (\ell_1, \ldots, \ell_m)$ and $u = (u_1, \ldots, u_m)$ contain lower and upper bounds respectively. The vertical bar in the argument list indicates that everything before the bar is a variable and everything after is a parameter. The constraint requires, for each $i = 1, \ldots, m$, that at least ℓ_i and at most u_i of the variables in X take the value v_i. To require that each nurse work at least five and at most six days, one can write

$$\text{cardinality}(w_{\cdot\cdot} \mid (\text{A}, \text{B}, \text{C}, \text{D}), (5, 5, 5, 5), (6, 6, 6, 6))$$

where $w_{\cdot\cdot}$ refers to the set of all the variables w_{sd}.

The *nvalues* constraint is written

$$\text{nvalues}(X \mid \ell, u)$$

and requires that the variables in $X = \{x_1, \ldots, x_n\}$ take at least ℓ and at most u different values. To require that at most two nurses work any given shift, one can write

$$\text{nvalues}(w_{s\cdot} \mid 1, 2), \text{ all } s$$

Both nvalues and cardinality generalize the alldiff constraint, but one should use alldiff when possible, since it invokes a more efficient filtering algorithm.

The remaining constraints are not easily expressed in the notation developed so far, because they relate to the pattern of shifts worked by a given nurse. For this reason, it is useful to move to the formulation suggested by Table 2.3. Let y_{id} be the shift assigned to nurse i on day d, where shift 0 denotes a day off. It is first necessary to ensure that all three shifts be assigned for each day. The alldiff constraint serves the purpose:

$$\text{alldiff}(y_{\cdot d}), \text{ all } d$$

This condition is implicit in the constraints already written, but it is generally good practice to write redundant constraints when one is aware of them, in order to strengthen propagation.

The *stretch* constraint was expressly developed to impose conditions on stretches or contiguous sequences of shift assignments in employee scheduling problems. Given a tuple $x = (x_1, \ldots, x_n)$ of variables, a stretch is a maximal sequence of consecutive variables that take the same value. Thus, if $x = (x_1, x_2, x_3) = (a, a, b)$, x contains a stretch of value a having length 2 and a stretch of b having length 1, but it does not contain a stretch of a having length 1. The stretch constraint is written

$$\text{stretch}(x \,|\, v, \ell, u, P)$$

where $x = (x_1, \ldots, x_n)$, while v, ℓ, and u are defined as in the cardinality constraint. P is a set of *patterns*, each of which is a pair (v, v') of distinct values. The stretch constraint requires, for each v_i in v, that every stretch of value v_i have length at least ℓ_i and at most u_i. It also requires that whenever a stretch of value v comes immediately before a stretch of value v', the pair (v, v') must occur in the pattern set P. The requirements concerning consecutive nursing shifts can now be written

$$\text{stretchCycle}(y_{i\cdot} \,|\, (2,3), (2,2), (6,6), P), \text{ all } i$$

where P contains all patterns that include a day off:

$$P = \{(s, 0), (0, s) \,|\, s = 1, 2, 3\}$$

A cyclic version of the constraint is necessary because every week must have the same schedule. The cyclic version treats the week as a cycle and allows a single stretch to extend across the weekend.

Finally, the two formulations must be forced to have the same solution. This is accomplished with *channeling constraints*, which in this case take the form

$$w_{y_{id}d} = i, \text{ all } i, d \tag{2.9}$$

and

$$y_{w_{sd}s} = s, \text{ all } s, d \tag{2.10}$$

Constraint (2.9) says that on any given day d, the nurse assigned to the shift to which nurse i is assigned must be nurse i, and similarly for (2.10). The subscripts y_{id} in (2.9) and w_{sd} in (2.10) are *variable indices* or *variable subscripts*.

The model is now complete. It can be written with five metaconstraints, plus domains:

$$\begin{aligned}
&\text{alldiff: } \left\{ \begin{array}{l} (w_{\cdot d}) \\ (y_{\cdot d}) \end{array} \right\}, \text{ all } d \\
&\text{cardinality: } (w_{\cdot\cdot} \,|\, (\mathrm{A},\mathrm{B},\mathrm{C},\mathrm{D}), (5,5,5,5), (6,6,6,6)) \\
&\text{nvalues: } (w_{s\cdot} \,|\, 1, 2), \text{ all } s \\
&\text{stretchCycle: } (y_{i\cdot} \,|\, (2,3), (2,2), (6,6), P), \text{ all } i \\
&\text{linear: } \left\{ \begin{array}{l} w_{y_{id}d} = i, \text{ all } i \\ y_{w_{sd}d} = s, \text{ all } s \end{array} \right\}, \text{ all } d \\
&\text{domains: } \left\{ \begin{array}{l} w_{sd} \in \{\mathrm{A},\mathrm{B},\mathrm{C},\mathrm{D}\}, \ s = 1,2,3 \\ y_{id} \in \{0,1,2,3\}, \ i = \mathrm{A},\mathrm{B},\mathrm{C},\mathrm{D} \end{array} \right\}, \text{ all } d
\end{aligned} \tag{2.11}$$

The linear constraints are so classified because they are linear aside from the variable indices, which will be eliminated as described below. One can appreciate the convenience of metaconstraints by attempting to formulate this problem with, say, an MILP model.

2.5.2 Inference: Domain Filtering

The alldiff constraint poses an interesting filtering problem that, fortunately, can be easily solved. Suppose, for example, that the current domains of assignment variables w_{s1} are:

$$w_{11} \in \{\mathrm{A},\mathrm{B}\}, \ w_{21} \in \{\mathrm{A},\mathrm{B}\}, w_{31} \in \{\mathrm{A},\mathrm{B},\mathrm{C},\mathrm{D}\}$$

The days are numbered $1,\ldots,7$ so that the subscript 1 in w_{s1} refers to Sunday. Thus, only nurse A or B can be assigned to shift 1 or 2 on Sunday, while any nurse can be assigned to shift 3. Since

$$\text{alldiff}(w_{11}, w_{21}, w_{31})$$

must be enforced, one can immediately deduce that neither A nor B can be assigned to shift 3. Thus, the domain of w_{31} can be reduced to $\{\text{C}, \text{D}\}$. This type of reasoning can be generalized by viewing the solution of alldiff as a matching problem on a bipartite graph, for which there are verys fast algorithms. Optimality conditions for maximum cardinality matching allow one to identify values that can be removed from domains. These ideas are presented in Section 6.8. In a similar fashion, network flow models can provide filtering for the cardinality and nvalues constraints.

Filtering for the stretch constraint is more complicated, but nonetheless tractable. Suppose, for example, that the domains of y_{Ad} contain the values listed beneath each variable:

y_{A1}	y_{A2}	y_{A3}	y_{A4}	y_{A5}	y_{A6}	y_{A7}
0	0		0		0	0
1			1	1	1	1
2	2	2		2		2
3	3	3	3		3	3

(2.12)

This means, for instance, that nurse A must work either shift 2 or shift 3 on Tuesday. After some thought, one can deduce from the stretch constraint in (2.11) that several values can be removed, resulting in the following domains:

y_{A1}	y_{A2}	y_{A3}	y_{A4}	y_{A5}	y_{A6}	y_{A7}
0			0		0	0
				1	1	1
2	2	2				2
3	3	3				3

(2.13)

Note that two variables are fixed. A polynomial-time dynamic programming algorithm can remove all infeasible values for any given stretch constraint. It is described in Section 6.11.

2.5.3 Inference for Variable Indices

The variably indexed expression $w_{y_{id}d}$ in (2.11) can be processed with the help of an *element* constraint. Domain reduction algorithms for this

constraint are well known in the constraint programming community, and elementary polyhedral theory provides a continuous relaxation for the constraint.

The type of element constraint required here has the form

$$\text{element}(y, x, z) \tag{2.14}$$

where y is an integer-valued variable, x is a tuple $(x_1, \ldots, x_m)$ of variables, and z is a variable. The constraint requires that z be equal to the yth variable in the list $x_1, \ldots, x_m$. One can therefore deal with a variably indexed expression like x_y by replacing it with z and adding the element constraint (2.14). In particular, the constraint $w_{y_{id}d} = i$ is parsed by replacing it with $z_{id} = i$ and generating the constraint

$$\text{element}\,(y_{id}, (w_{0d}, \ldots, w_{3d}), z_{id})$$

Similarly, $y_{w_{sd}d} = s$ is replaced with $\bar{z}_{sd} = s$ and

$$\text{element}\,(w_{sd}, (y_{Ad}, \ldots, y_{Dd}), \bar{z}_{sd})$$

Filtering for element is a simple matter. An example will illustrate this and show how propagation of channeling constraints between two models can reduce domains. Focusing on Sunday (day 1), suppose the domains of w_{s1} and y_{i1} contain the elements listed beneath each variable:

w_{01}	w_{11}	w_{21}	w_{31}
A		A	A
B	B		B
C	C	C	
	D	D	D

y_{A1}	y_{B1}	y_{C1}	y_{D1}
0	0		0
1	1	1	
	2	2	2
3		3	3

Suppose that no further propagation is possible among the constraints involving only the variables w_{sd}, and similarly for the constraints involving only the variables y_{id}. Nonetheless, the channeling constraints can yield further domain reduction. For instance, since y_{A1} has domain $\{0, 1\}$, the constraint

$$\text{element}\,(y_{\text{A1}}, (w_{01}, \ldots, w_{31}), z_{\text{A1}})$$

implies that y_{A1} must select either w_{01} or w_{11} to be equated with z_{A1}. But $z_{\text{A1}} = \text{A}$, and only w_{01}'s domain contains A. Thus, y_{A1} must select

w_{01}, and y_{A1}'s domain can be reduced to $\{0\}$. Similarly, the constraints $\bar{z}_{01} = 0$ and

$$\text{element}\,(w_{01}, (y_{\mathrm{A1}}, \ldots, y_{\mathrm{D1}}), \bar{z}_{01})$$

remove C from the domain $\{\mathrm{A}, \mathrm{B}, \mathrm{C}\}$ of w_{01}. Deductions of this kind reduce the domains to:

w_{01}	w_{11}	w_{21}	w_{31}
A			A
B	B		
	C	C	
		D	D

y_{A1}	y_{B1}	y_{C1}	y_{D1}
0	0		
	1	1	
		2	2
3			3

Exercises

2.8. Formulate the following problem using appropriate global constraints (alldiff, cardinality, nvalues, stretchCycle) and channeling constraints. There are six security guards and four stations. Each station must be staffed by exactly one guard each night of the week. A guard must be on duty four or five nights a week. No station should be staffed by more than three different guards during the week. A guard must never staff the same station two or more nights in a row and must staff at least three different stations during the week. A guard must not staff stations 1 and 2 on consecutive nights (in either order), and similarly for stations 3 and 4. Every week will have the same schedule. *Hints.* Let w_{sd} be the guard at station s on night d, and let y_{id} be the station assigned to guard i on night d. Because two guards are unassigned on a given night, two dummy stations are necessary to make the channeling constraints work.

2.9. Write all feasible solutions of the stretchCycle constraint in (2.11) to confirm that the domains in (2.12) can be reduced to (2.13) and cannot be reduced further.

2.10. Formulate this lot-sizing problem using a combination of variable indices, linear constraints, and stretch constraints. In each period t, there is a demand d_{it} for product i, and this demand must be met from the stock of product i at the end of period t. At most, one product can be manufactured in each period t, represented by variable y_t. If nothing is manufactured, $y_t = 0$, where product 0 is a dummy product. The quantity of product i manufactured in any period must be either q_i or zero. When product i is manufactured, its manufacture must continue no fewer than ℓ_i and no more than

u_i periods in a row. The manufacture of product i in any period must be followed in the next period by the manufacture of one of the products in S_i (one may assume $0, i \in S_i$). The unit holding cost per period for product i is h_i, and the unit manufacturing cost is g_i. The setup cost of making a transition from product i in one period to product j in the next is c_{ij} (where possibly i and/or j is 0). Minimize total manufacturing, holding, and setup costs over an n-period horizon while meeting demand. After formulating the problem, indicate how to replace the variable indices with element constraints. *Hints:* Let variable x_{ij} represent the quantity of product i manufactured in period t, and s_{it} the stock at the end of the period. The element(y, x, z) constraint also has a form element$(y, z \,|\, a)$, where a is a tuple of constants. It sets z equal to a_y.

2.11. A possible difficulty with the model of the previous exercise is that the setup cost after an idle period is always the same, regardless of which (nondummy) product was manufactured last. How can the model be modified to allow setup cost to depend on the last nondummy product? *Hint:* Define several dummy products. For example, let $y_t = -i$ indicate that no product is manufactured in peropd t, and the last product manufactured is i.

2.12. Formulate the following problem using linear disjunctions, variable indices, and stretchCycle constraints. The week is divided into n periods, and in each period t, d_t megawatts of electricity must be generated. Each power plant i generates at most q_i megawatts while operating. Once plant i is started, it must run for at least ℓ_i periods, and once shut down, it must remain idle for at least ℓ'_i periods. The cost of producing one megawatt of power for one period at plant i is g_i, and the cost of starting up the plant is c_i. Determine which plants to operate in each period to minimize cost while meeting demand. The schedule must follow a weekly cycle. Indicate how to replace the variable indices with element constraints. *Hint:* Let $y_{it} = 1$ if plant i operates in period t, and let the setup cost incurred by plant i in period t be $c_{iy_{i,t-1}y_{it}}$, where $c_{i01} = c_i$ and $c_{i00} = c_{i01} = c_{i11} = 0$.

2.6 Continuous Global Optimization

Optimization problems need not contain discrete variables to be combinatorial in nature. A continuous optimization problem may have a large number of locally optimal solutions, which are solutions that are optimal in a neighborhood about them. Nonlinear programming solvers, highly developed as they are, are often geared to finding only a local optimum. To identify a global optimum, there may be no alternative but to examine the entire solution space, at least implicitly.

The most popular and effective global solvers use a branch-and-bound search that is analogous to the one presented for the freight transfer problem. The main difference is that because the variable domains are continuous intervals rather than finite sets, the algorithm branches on a variable by splitting an interval into two or more intervals. This sort of branching divides continuous space into increasingly smaller boxes until a global solution can be isolated. Constraint propagation and relaxation are also somewhat different than in discrete problems, because they employ techniques that are specialized to the nonlinear functions that typically occur in continuous problems.

Global optimization is best illustrated with a very simple example that is chosen to highlight some of the simpler techniques rather than to represent a practical application. The problem is:

$$\begin{aligned} &\text{linear: } \begin{cases} \max x_1 + x_2 \\ 2x_1 + x_2 \leq 2 \end{cases} \\ &\text{bilinear: } 4x_1x_2 = 1 \\ &\text{domains: } x_1 \in [0,1],\ x_2 \in [0,2] \end{aligned} \tag{2.15}$$

The feasible set is illustrated in Figure 2.3. There are two locally optimal solutions,

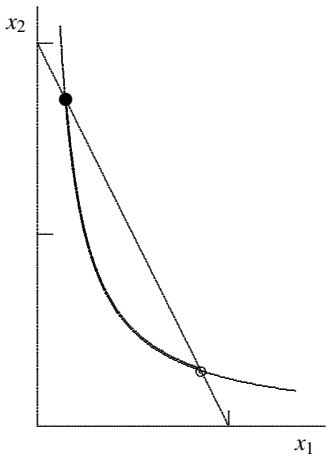

Fig. 2.3 Illustration of a continuous global optimization problem. The heavy curve represents the feasible set. The solid circle marks the optimal solution, and the open circle marks a second locally optimal solution.

$$(x_1, x_2) = (\tfrac{1}{2} + \tfrac{1}{4}\sqrt{2}, 1 - \tfrac{1}{2}\sqrt{2}) \approx (0.853553, 0.292893)$$
$$(x_1, x_2) = (\tfrac{1}{2} - \tfrac{1}{4}\sqrt{2}, 1 + \tfrac{1}{2}\sqrt{2}) \approx (0.146447, 1.707107)$$

the latter of which is globally optimal.

2.6.1 Inference: Bounds Propagation

Bounds propagation can be applied to nonlinear constraints as well as linear inequalities. This is easiest when a constraint can be "solved" for a bound on each variable in terms of a monotone function of the other variables, as is possible for linear inequalities. To obtain a new bound for a given variable, one need only substitute the smallest or largest possible values for the other variables.

In the example, $4x_1x_2 = 1$ can be solved for $x_1 = \frac{1}{4}x_2$. By substituting the upper bound of 2 for x_2, one can update the lower bound on x_1 to $\frac{1}{8}$, resulting in the domain $[0.125, 1]$. Similarly, the domain for x_2 can be reduced to $[0.25, 2]$. The linear constraint $2x_1+x_2 \leq 2$ allows further reduction of the domains to $[0.125, 0.875]$ and $[0.25, 1.75]$, respectively. By cycling back through the two constraints repeatedly, one asymptotically converges to domains of approximately $[0.146, 0.854]$ for x_1 and $[0.293, 1.707]$ for x_2. Global solvers typically truncate this process quite early, however, because the marginal gains of further iterations are not worth the time investment. The computations to follow reflect the results of cycling through the constraints only once.

2.6.2 Relaxation: Factored Functions

Linear relaxations can often be created for nonlinear constraints by *factoring* the functions involved into more elementary functions for which linear relaxations are known. For instance, the constraint $x_1x_2/x_3 \leq 1$ can be written $y_1 \leq 1$ by setting $y_1 = y_2/x_3$ and $y_2 = x_1x_2$. This factors the function x_1x_2/x_3 into the elementary operations of multiplication and division, for which tight linear relaxations have been derived.

The constraint $4x_1x_2 = 1$ in the example (2.15) can be written $4y = 1$ by setting $y = x_1x_2$. The product $y = x_1x_2$ has the well-known relaxation

$$\begin{aligned} L_2x_1 + L_1x_2 - L_1L_2 \le y \le L_2x_1 + U_1x_2 - U_1L_2 \\ U_2x_1 + U_1x_2 - U_1U_2 \le y \le U_2x_1 + L_1x_2 - L_1U_2 \end{aligned} \tag{2.16}$$

where $[L_i, U_i]$ is the current interval domain of x_i.

The example (2.15) therefore has the relaxation

$$\text{linear:} \begin{cases} \max\ x_1 + x_2 \\ 4y = 1 \\ 2x_1 + x_2 \le 2 \\ L_2x_1 + L_1x_2 - L_1L_2 \le y \le L_2x_1 + U_1x_2 - U_1L_2 \\ U_2x_1 + U_1x_2 - U_1U_2 \le y \le U_2x_1 + L_1x_2 - L_1U_2 \\ L_i \le x_i \le U_i, \quad i = 1, 2 \end{cases} \tag{2.17}$$

The constraint $4y = 1$ can be eliminated by substituting $1/4$ for y in the other constraints. Initially, $[L_1, U_1] = [0.125, 0.875]$ and $[L_2, U_2] = [0.25, 1.75]$, because these domains result from the bounds propagation described in the previous section.

2.6.3 Branching Search

At the root node of a branching tree (Figure 2.4), the initial linear relaxation (2.17) is solved to obtain the solution $(x_1, x_2) = (\frac{1}{7}, \frac{41}{24}) \approx (0.143, 1.708)$. This solution is infeasible in the original problem (2.15), because $4x_1x_2 = 1$ is not satisfied. It is therefore necessary to branch by splitting the domain of a variable.

The choice of branching heuristic can be crucial to fast solution, but for purposes of illustration one can simply split the domain $[0.25, 1.75]$ of x_2 into two equal parts. Restriction 1 in Figure 2.4 corresponds to the lower branch, $x_2 \in [0.25, 1]$. The domains reduce as shown. The solution $(x_1, x_2) = (0.25, 1)$ of the relaxation is feasible and becomes the incumbent solution. The objective function value is 1.25, which provides a lower bound on the optimal value of the problem. No further branching on the left side is necessary. This, in effect, rules out the locally optimal point $(0.854, 0.293)$ as a global optimum, and it remains only to find the other locally optimal point to a desired degree of accuracy.

Moving to restriction 2, the solution of the relaxation is $(x_1, x_2) = (\frac{41}{280}, \frac{239}{140}) \approx (0.146429, 1.707143)$. This is slightly infeasible, as $4x_1x_2$ is 0.999898 rather than 1. One could declare this solution to be feasible

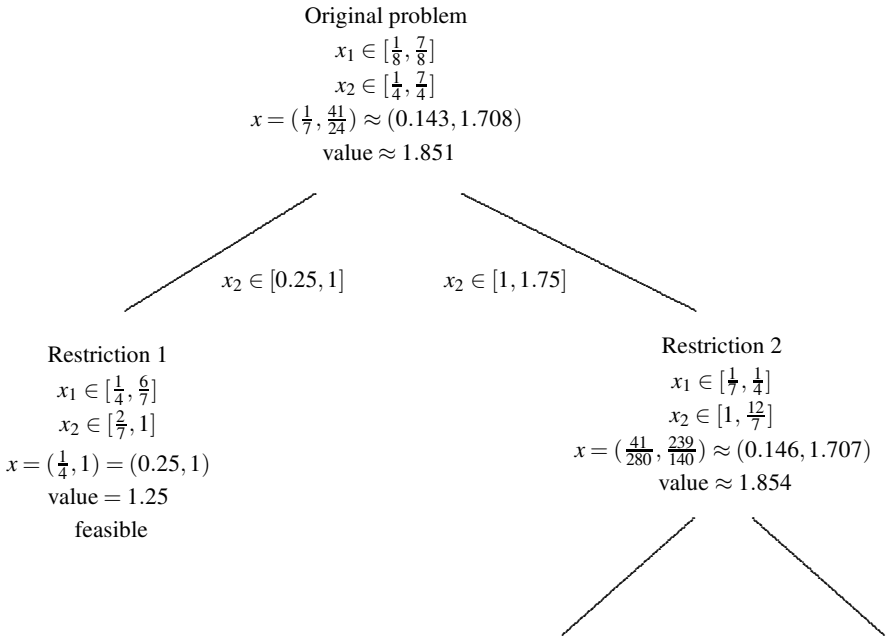

Fig. 2.4 A portion of a search tree for a global optimization problem. The reduced domains and solution of the relaxation are shown at each node.

within tolerances and terminate the search. Alternatively, one could continue branching to find a strictly feasible solution that is closer to the optimum than the incumbent.

Before branching, however, it may be possible to reduce domains by using Lagrange multipliers and the lower bound 1.25 on the optimal value. The constraints $2x_1 + x_2 \leq 2$ and $\frac{1}{4} \leq U_2 x_1 + L_1 x_2 - L_1 U_2$, respectively, have Lagrange multipliers 1.1 and 0.7 in the solution of the relaxation (2.17) (the remaining constraints have vanishing multipliers since they are not satisfied as equalities). The multiplier 1.1 indicates that any reduction Δ in the right-hand side of $2x_1+x_2 \leq 2$ reduces the optimal value of the relaxation, currently 1.854, by at least 1.1Δ. Thus any reduction Δ in the left-hand side of the constraint, currently equal to 2, has the same effect. But the value of the relaxation should not be reduced below 1.25, because the optimal value of the original problem is at least 1.25. So one can impose the bound $1.1\Delta \leq 1.854 - 1.25$, which can be written

$$1.1[2 - (2x_1 + x_2)] \leq 1.854 - 1.25$$

This implies $2x_1 + x_2 \geq 1.451$, an inequality that must be satisfied by any optimal solution found in the subtree rooted at the current node. Propagation of this inequality reduces the current domain $[1, 1.714]$ of x_2 to $[1.166, 1.714]$. Similar treatment of the inequality

$$\frac{1}{4} \leq U_2 x_1 + L_1 x_2 - L_1 U_2$$

which currently is $\frac{12}{7}x_1 + \frac{1}{7}x_2 \geq \frac{97}{196}$, has no effect on the domains.

Exercises

2.13. Derive a linear relaxation for $y = x^2$ from (2.16), and derive a linear relaxation for $y = a^x$ when $a > 0$. Now, write a factored relaxation for the problem

$$\begin{aligned} &\max\ a^{x_1} + b^{x_2} \\ &(x_1 + 2x_2)^2 \leq 1,\ \ L_j \leq x_j \leq U_j,\ j = 1, 2 \end{aligned}$$

when $a, b > 0$.

2.14. Consider the problem:

$$\begin{aligned} &\max\ -x_1^2 - x_2^2 \\ &-3x_1 - x_2 \leq -15 \\ &x_1, x_2 \geq 0 \text{ and integral} \end{aligned}$$

Suppose that the best known feasible solution for the problem is $(x_1, x_2) = (4, 3)$. The continuous relaxation has the optimal solution $(x_1, x_2) = (4.5, 1.5)$ with a Lagrange multiplier of 3 corresponding to the first constraint. Derive an inequality that must be satisfied by any optimal solution. (Round up the right-hand side, because the coefficients and variables are integral.)

2.7 Product Configuration

A product configuration example shows how inference and relaxation combine in a slightly more complex problem. It also introduces indexed linear constraints, which occur when the coefficients of a linear constraint are indexed by variables.

The problem is to decide what type of power supply, disk drive, and memory chip to install in a laptop computer, and how many of each. The objective is to minimize total weight while meeting the computer's

Table 2.4 Data for a small instance of the product configuration problem.

Component i	*Type* k	*Net power generation* A_{i1k}	*Disk space* A_{i2k}	*Memory capacity* A_{i3k}	*Weight* A_{i4k}	*Max number used*
1. Power supply	A	70	0	0	200	1
	B	100	0	0	250	
	C	150	0	0	350	
2. Disk drive	A	−30	500	0	140	3
	B	−50	800	0	300	
3. Memory chip	A	−20	0	250	20	3
	B	−25	0	300	25	
	C	−30	0	400	30	
Lower bound L_j		0	700	850	0	
Upper bound U_j		∞	∞	∞	∞	
Unit cost c_j		0	0	0	1	

requirements—upper and lower bounds on memory, disk space, net power supplied, and weight. Only one type of each component may be installed. One power supply, at most three disk drives, and at most three memory chips may be used. Data for a small instance of the problem appear in Table 2.4.

2.7.1 Formulating the Problem

To formulate the problem as an optimization model, let variable t_i be the type of component i that is chosen, and q_i the quantity. It is convenient to let variable r_j denote the total amount of attribute j that is produced, where $j = 1, 2, 3, 4$ correspond, respectively, to net power, disk space, memory, and weight. The computer requires a minimum L_j and maximum U_j of each attribute j.

In general, the objective in such a problem is to minimize the cost of supplying the attributes, where c_j is the unit cost of attribute j. In this case, only the weight is of concern, so that $(c_1, \ldots, c_4) = (0, 0, 0, 1)$.

The interesting aspect of the problem is modeling the connection between the amount r_j of attribute j supplied and the configuration chosen. The component characteristics can be arranged in a three-dimensional array A, in which A_{ijk} is the net amount of attribute j produced by type k of component i. A natural way to write r_j is to sum the contributions of each component

$$r_j = \sum_i A_{ijt_i} q_i$$

where A_{ijt_i} is the amount contributed by the selected type of component i. This leads immediately to the optimization model

$$\text{linear:} \begin{cases} \min \sum_j c_j r_j & (a) \\ r_j = \sum_i A_{ijt_i} q_i, \quad \text{all } j & (b) \end{cases} \tag{2.18}$$

$$\text{domains:} \begin{cases} t_1 \in \{\text{A,B,C}\}, t_2 \in \{\text{A,B}\}, t_3 \in \{\text{A,B,C}\} \\ r_j \in [L_j, U_j], \quad \text{all } j \\ q_1 \in \{1\}, \quad q_2, q_3 \in \{1, 2, 3\} \end{cases}$$

The optimal solution of this model is $t = (t_1, t_2, t_3) = (\text{C, A, B})$ and $q = (1, 2, 3)$, which results in weight 705. This calls for installation of the largest power supply (type C), two of the smaller disk drives (type A), and three of the medium-sized memory chips (type B).

Constraints (b) of (2.18) are linear equations in which the coefficients have variable indices t_i. For each j, (b) is processed as an *indexed linear* metaconstraint:

$$\text{indexedLinear:} \begin{cases} r_j = \sum_i z_{ij} \\ \text{elementCoeff: } (t_i, q_i, z_{ij} \,|\, (A_{ij1}, \ldots, A_{ijm})), \text{ all } i \end{cases} \tag{2.19}$$

The *element coefficient* constraint is a specially structured element constraint. It sets z_{ij} equal to the t_i^{th} tuple in the list $A_{ij1}q_i, \ldots, A_{ijm}q_i$. The solution software should generate this reformulation automatically.

2.7.2 Inference: Indexed Linear Constraint

Domain reduction for the linear constraints in (2.18) can be achieved by bounds propagation, as discussed earlier. It remains to propagate the indexed linear metaconstraint.

Domain filtering for the element coefficient constraint is similar to that of an element constraint but takes advantage of its special form. The details are presented in Sections 6.7.1–6.7.2. Additional filtering may be derived from the fact that the indexed linear constraint implies integer knapsack inequalities. If $[L_j, U_j]$ is the current domain of r_j, then

$$L_j \leq \sum_i A_{ijt_i} q_i \leq U_j$$

This in turn yields the integer knapsack inequalities

$$L_j \leq \sum_i \max_{k \in D_{t_i}} \{A_{ijk}\}\, q_i$$

$$\sum_i \min_{k \in D_{t_i}} \{A_{ijk}\}\, q_i \leq U_j$$

which can be used to reduce the domains of the variables q_i.

Three of the attributes in the example problem (power, disk space, and memory), respectively, yield the following knapsack inequalities:

$$\begin{aligned} 0 &\leq \max\{70, 100, 150\}\, q_1 + \max\{-30, -50\}\, q_2 + \\ &\qquad \max\{-20, -25, -30\}\, q_3 \\ 700 &\leq \max\{0, 0, 0\}\, q_1 + \max\{500, 800\}\, q_2 + \max\{0, 0, 0\}\, q_3 \\ 850 &\leq \max\{0, 0, 0\}\, q_1 + \max\{0, 0\}\, q_2 + \max\{250, 300, 400\}\, q_3 \end{aligned} \tag{2.20}$$

which simplify to

$$0 \leq 150q_1 - 30q_2 - 20q_3, \quad 700 \leq 800q_2, \quad 850 \leq 400q_3 \tag{2.21}$$

Weight yields no useful inequality as its lower bound is zero. Propagation of these inequalities reduces the domain of q_3 to $\{3\}$ but does not affect the other domains. When all the constraints of (2.18) are propagated, the domains are reduced to the following:

$$
\begin{array}{lll}
q_1 \in \{1\} & q_2 \in \{1,2\} & q_3 \in \{3\} \\
t_1 \in \{C\} & t_2 \in \{A,B\} & t_3 \in \{B,C\} \\
z_{11} \in [150,150] & z_{21} \in [-75,-30] & z_{31} \in [-90,-75] \\
z_{12} \in [0,0] & z_{22} \in [700,1600] & z_{32} \in [0,0] \\
z_{13} \in [0,0] & z_{23} \in [0,0] & z_{33} \in [900,1200] \\
z_{14} \in [350,350] & z_{24} \in [140,600] & z_{34} \in [75,90] \\
r_1 \in [0,45] & r_2 \in [700,1600] & r_3 \in [900,1200] \\
r_4 \in [565,1040] & &
\end{array}
\tag{2.22}
$$

2.7.3 Relaxation: Indexed Linear Constraint

Problem (2.18) is linear except for the indexed linear constraints and the integrality restriction on q_i. It can therefore be relaxed by dropping the integrality condition and finding a linear relaxation for the indexed linear constraints.

The relaxation consists of two parts: (i) a relaxation for the knapsack inequalities (2.21) and (ii) a relaxation the element coefficient constraint. The knapsack inequalities themselves provide a linear relaxation, which can be strengthened with knapsack cuts, Gomory cuts (Section 7.3.2), and other cutting planes for integer linear inequalities.

The key to relaxing an element constraint is to note that it implies a disjunction. In particular, the element coefficient constraint in (2.19) implies the disjunction of linear equations:

$$\bigvee_{k \in D_{t_i}} (z_{ij} = A_{ijk} q_i) \tag{2.23}$$

This says that at least one of the equations $z_{ij} = A_{ijk} q_i$ must hold. Relaxations for disjunctions of linear systems in general are discussed in Section 7.4. The particular disjunction (2.23) can be relaxed as follows:

$$z_{ij} = \sum_{k \in D_{t_i}} A_{ijk} q_{ik}, \quad q_i = \sum_{k \in D_{t_i}} q_{ik}$$

where $q_{ik} \geq 0$ are new variables. This is a convex hull relaxation, which is the tightest possible linear relaxation. It can be tightened further by using known bounds on the variables, say $L_j \leq z_{ij} \leq U_j$ and $L_{q_i} \leq q_i \leq U_{q_i}$, by adding the constraints

$$
\begin{aligned}
&L_j\delta_k \le A_{ijk}q_{ik} \le U_j\delta_k, \ \text{ all } k \in D_{t_i}\\
&L_{q_i}\delta_k \le q_{ik} \le U_{q_i}\delta_k, \ \ \delta_k \ge 0, \ \text{ all } k \in D_{t_i}\\
&\sum_{k\in D_{t_i}} \delta_k = 1
\end{aligned}
$$

A solution of the relaxation is feasible for (2.23) if exactly one of the δ_k, say δ_{k^*}, is equal to 1, which indicates that $q_{ik^*} = q_i$ and $z_{ij} = A_{ijk^*}q_i$.

Based on this idea, the relaxation of (2.18) becomes:

$$
\text{linear:}\begin{cases}
\min \sum_j c_j r_j & (a)\\
r_j = \sum_i \sum_{k\in D_{t_i}} A_{ijk}q_{ik}, \ \text{ all } j & (b)\\
q_i = \sum_{k\in D_{t_i}} q_{ik}, \ \text{ all } i & (c)\\
L_j\delta_k \le A_{ijk}q_{ik} \le U_j\delta_k, \ \ k \in D_{t_i}, \text{ all } i & (d)\\
L_{q_i}\delta_k \le q_{ik} \le U_{q_i}\delta_k, \ \ \delta_k \ge 0, \ \ k \in D_{t_i}, \text{ all } i & (e)\\
\sum_{k\in D_{t_i}} \delta_k = 1 & (f)\\
L_j \le r_j \le U_j, \ \text{ all } j & (g)\\
L_{q_i} \le q_i \le U_{q_i}, \ \text{ all } i & (h)\\
L_j \le \sum_i \max_{k\in D_{t_i}} \{A_{ijk}q_i\}, \ \text{ all } j & (i)\\
\sum_i \min_{k\in D_{t_i}} \{A_{ijk}q_i\} \le U_j, \ \text{ all } j & (j)\\
q_{ik} \ge 0, \ \text{ all } i,k & (k)
\end{cases}
\qquad (2.24)
$$

domains: $q_i, q_{ik}, r_j, \delta_k \in \mathbb{R}, \ \text{ all } i,j,k$

The relaxation can be strengthened with knapsack cuts for (i) and (j) and other cutting planes, such as Gomory cuts (Section 7.3.2).

The solution of the relaxation (2.24), given the reduced domains in (2.22), is $q_1 = q_{13} = 1$, $q_2 = q_{2A} = 2$, and $q_3 = q_{3B} = 3$, with the other q_{ik}'s equal to zero. Because the q_i's and δ_k's are integer, the solution is feasible and no branching is necessary. The δ_k's indicate that $(t_1, t_2, t_3) = (C, A, B)$. Also $(r_1, \ldots, r_4) = (15, 1000, 900, 705)$, which indicates that the minimum weight of the computer is $r_4 = 705$.

2.7.4 Branching Search

A solution of the relaxation (2.24) is feasible for the original problem (2.18) unless it violates the integrality constraint for some q_i or fails to satisfy the indexed linear constraint. In the former case, branching follows the usual pattern. If $\hat{q}_i$ is a noninteger value in the solution of the relaxation, the domain $\{L_{q_i}, \ldots, U_{q_i}\}$ of q_i is split into $\{L_{q_i}, \ldots, \lfloor \hat{q}_i \rfloor\}$ and $\{\lceil \hat{q}_i \rceil, \ldots, U_{q_i}\}$. At either branch, the smaller domains are propagated and the resulting relaxation solved.

The solution of the relaxation violates the indexed linear constraint for some i when at least two q_{ik}s are positive, say q_{ik_1} and q_{ik_2}. In this case, the search can branch by splitting the domain of t_i into two sets, one excluding k_1 and the other excluding k_2.

Exercises

2.15. What knapsack covering inequality can be inferred from the metaconstraint (2.19)? Assume each $x_i \geq 0$.

2.16. A farmer wishes to apply fertilizer to each of several plots. The additional crop yield from plot i per unit of type k fertilizer applied is a_{ik}, and the runoff into streams is c_{ik}. There are storage facilities on the farm for at most K different kinds of fertilizer. Use variable indices and the nvalues constraint to formulate the problem of identifying which fertilizer to apply to each plot, and how much, to maximize total additional yield subject to an upper limit U on the amount of runoff. Now, reformulate the problem using indexed linear constraints in place of variable indices.

2.8 Planning and Scheduling

An elementary planning and scheduling problem illustrates *logic-based Benders decomposition*, a form of *constraint-directed search*. It is formally similar to branching in that it enumerates a sequence of problem restrictions, but the restrictions are created differently. Relaxations also help guide the search, but again in a different way.

The restrictions are formed by enumerating assignments to a subset of the variables (the *search variables*) and, for each assignment, solving a restricted problem (*subproblem*) for the remaining variables.

The solution of the subproblem yields a constraint (*Benders cut*) that encodes information about the solution. The next assignment enumerated must satisfy all the Benders cuts so far generated. The object is to avoid assignments that are similar to those already tried, so that a better one might be found. The search is constraint-directed in that it is directed by Benders cuts. Relaxation also helps direct the search, because the current set of Benders cuts is in effect a relaxation of the original problem.

Classical Benders decomposition requires that the subproblem be a linear or nonlinear programming problem, because the Benders cuts rely on Lagrange multipliers obtained from the subproblem. However, these multipliers can be more generally viewed as the solution of an *inference dual* of the subproblem. Because an inference dual can be defined for a subproblem of any form, this leads to a generalization of Benders decomposition. The generalization is "logic-based" in the sense that the inference dual is solved by constructing a proof of optimality for the subproblem solution.

The planning and scheduling problem presented here represents an important class of problems that frequently occur in manufacturing, supply chain management, and elsewhere. The goal is to assign tasks to facilities and then schedule the tasks. The facilities might be factories, machines in a factory, transport modes, delivery vehicles, or computer processors.

In the problem instance at hand, five jobs are to be allocated to two machines, named A and B, and scheduled on them. Each job j has a release time r_j and a deadline d_j. The time required to process job j on machine i is p_{ij}. The specific problem data appear in Table 2.5. Note that machine A is faster than machine B. The objective is to minimize makespan; that is, to minimize the finish time of the last job to finish.

2.8.1 Formulating the Problem

It is convenient to use a metaconstraint noOverlap to represent the scheduling portion of the problem. The constraint may, in general, be written

$$\text{noOverlap}(s \mid p)$$

where $s = (s_1, \ldots, s_n)$ are the start times of the jobs to be scheduled, and $p = (p_1, \ldots, p_n)$ are the processing times. The constraint is

Table 2.5 Data for a simple two-machine machine scheduling problem.

Job j	*Release time* r_j	*Deadline* d_j	*Processing time* $p_{\mathrm{A}j}$	$p_{\mathrm{B}j}$
1	0	10	1	5
2	0	10	3	6
3	2	7	3	7
4	2	10	4	6
5	4	7	2	5

satisfied when the jobs do not overlap. That is,

$$s_j + p_j \leq s_k \text{ or } s_k + p_k \leq s_j, \text{ all jobs } j, k \text{ with } j \neq k$$

Only two types of decision variables are needed to formulate the problem—the start time s_j already mentioned, and the machine x_j to which job j is assigned. If there are n jobs and m machines, the formulation is

$$\text{linear: min } M$$

$$\text{subproblem: } \begin{cases} \{x_1, \ldots, x_n\} \\ \text{linear: } \begin{cases} M \geq s_j + p_{x_j j}, & \text{all } j \\ s_j + p_{x_j j} \leq d_j, & \text{all } j \end{cases} \\ \text{noOverlap: } ((s_j \mid x_j = i) \,|\, (p_{ij} \mid x_j = i)), & \text{all } i \end{cases}$$

$$\text{domains: } s_j \in [r_j, \infty),\ x_j \in \{1, \ldots, m\}, \text{ all } j$$

In the objective function, M represents the makespan. The inequality constraints define the makespan and enforce the deadlines. A noOverlap constraint is imposed for each machine. In these constraints, the notation $(s_j \mid x_j = i)$ denotes the tuple of start times s_j for jobs assigned to machine i, and similarly for the processing times.

The *subproblem* constraint makes it possible to write a noOverlap constraint whose argument list would otherwise depend on the values of the assignment variables x_j. The subproblem constraint begins with a set of variables, in this case $\{x_1, \ldots, x_n\}$, that are treated as constants inside the scope of the constraint. The noOverlap argument list is therefore fixed. In general, the solver must have at hand one or more decomposition methods that can deal with a subproblem constraint. In this case, Benders decomposition is used.

A natural decomposition distinguishes the assignment portion from the scheduling portion. One can search over various assignments of jobs to machines and, for each, try to find a feasible schedule for the jobs assigned to each machine. The assignment variables x_j are therefore the search variables, and each subproblem is a scheduling problem that decouples into separate scheduling problems for the individual machines.

The inequality constraints could in principle be written outside the subproblem constraint, where they would be parsed as indexed linear constraints, as in the product configuration problem of the previous section. However, the decomposition used here places them in the subproblem, where they regulate the scheduling of assigned jobs.

2.8.2 Relaxation: The Master Problem

The master problem (relaxation) minimizes makespan subject to the Benders cuts generated so far. It can be solved by whatever method is most suitable for its structure. One option is to solve it as an MILP problem, because it is naturally expressed in this form. For this purpose, the variables x_i can be replaced with 0-1 variables x_{ij}, where $x_{ij} = 1$ when job i is assigned to machine j.

The master problem has the form

$$\begin{array}{l} \min\ M \\ \text{Benders cuts} \\ \text{relaxation of subproblem (optional)} \\ x_{ij} \in \{0,1\},\ \text{all } i,j \end{array} \tag{2.25}$$

The Benders cuts are described in the next section. The subproblem relaxation allows the master problem to select reasonable assignments before many Benders cuts have been accumulated.

A simple subproblem relaxation observes, for example, that the jobs assigned to a machine must fit within the earliest release time and latest deadline of those jobs. In fact, this is true of any subset of the jobs assigned to a given machine. To formulate this condition, let $J(t_1, t_2)$ be the set of jobs whose time windows lie in the interval $[t_1, t_2]$. So

$$J(t_1, t_2) = \{j \mid [r_j, d_j] \subset [t_1, t_2]\}$$

The total processing times of the jobs in $J_i(t_1, t_2)$ that are assigned to a given machine i must not exceed $t_2 - t_1$:

$$\sum_{j \in J(t_1,t_2)} p_{ij} x_{ij} \leq t_2 - t_1 \tag{2.26}$$

It suffices to consider release times for t_1 and deadlines for t_2. In the problem instance at hand,

$$\begin{array}{ll} J(0,7) = \{3,5\} & J(2,10) = \{3,4,5\} \\ J(0,10) = \{1,2,3,4,5\} & J(4,7) = \{5\} \\ J(2,7) = \{3,5\} & J(4,10) = \{5\} \end{array}$$

Some of these sets give rise to vacuous or redundant inequalities (2.26). For instance, the inequality for $J(0,7)$ is $p_{i3}x_{i3} + p_{i5}x_{i5} \leq 7$, which is $3x_{\mathrm{A}3} + 2x_{\mathrm{A}5} \leq 7$ for $i = \mathrm{A}$ and $7x_{\mathrm{B}3} + 5x_{\mathrm{B}5} \leq 7$ for $i = \mathrm{B}$. The former is obviously redundant since $x_{ij} \in \{0,1\}$. The latter is dominated by another inequality for machine B (namely, $7x_{\mathrm{B}3} + 5x_{\mathrm{B}5} \leq 5$). The nonredundant inequalities for machine A are

$$\begin{aligned} J(0,10): &\sum_{j \in \{1,2,3,4,5\}} p_{\mathrm{A}j}\, x_{\mathrm{A}j} \leq 10 \\ J(2,10): &\sum_{j \in \{3,4,5\}} p_{\mathrm{A}j}\, x_{\mathrm{A}j} \leq 8 \end{aligned} \tag{2.27}$$

and those for machine B are

$$\begin{aligned} J(0,10): &\sum_{j \in \{1,2,3,4,5\}} p_{\mathrm{B}j}\, x_{\mathrm{B}j} \leq 10 \\ J(2,7): &\sum_{j \in \{3,5\}} p_{\mathrm{B}j}\, x_{\mathrm{B}j} \leq 5 \\ J(2,10): &\sum_{j \in \{3,4,5\}} p_{\mathrm{B}j}\, x_{\mathrm{B}j} \leq 8 \\ J(4,7) \quad &\sum_{j \in \{5\}} p_{\mathrm{B}j}\, x_{\mathrm{B}j} \leq 3 \end{aligned} \tag{2.28}$$

Section 7.13.3 shows how to identify nonredundant inequalities of this sort in a systematic way.

Further inequalities can be added to the master problem to constrain the makespan M:

$$
\begin{aligned}
M &\geq \sum_{j\in\{1,2,3,4,5\}} p_{ij}\, x_{ij}, \quad i = \mathrm{A}, \mathrm{B} \\
v &\geq 2 + \sum_{j\in\{3,4,5\}} p_{ij}\, x_{ij}, \quad i = \mathrm{A}, \mathrm{B} \\
v &\geq 4 + \sum_{j\in\{5\}} p_{ij}\, x_{ij}, \quad i = \mathrm{A}, \mathrm{B}
\end{aligned}
\tag{2.29}
$$

The three sets of inequalities correspond to the release times 0, 2, and 4. The first includes the jobs in $J(0,\infty)$, the second the jobs in $J(2,\infty)$, and the third the jobs in $J(4,\infty)$. The subproblem relaxation in the master problem (2.25) therefore consists of inequalities (2.27)–(2.29).

2.8.3 Inference: Benders Cuts

The inference stage consists of inferring Benders cuts from the subproblem that results when the master problem variables are fixed to their current values.

The subproblem separates into an independent scheduling problem on each machine. Thus, if $\bar{x}$ is the solution of the previous master problem, the subproblem on each machine i is

$$
\begin{aligned}
&\min\ M_i \\
&s_j + p_{ij} \leq d_j, \ \text{ all } j \text{ with } \bar{x}_{ij} = 1 \\
&\mathrm{noOverlap}((s_j \mid \bar{x}_{ij} = 1) \,|\, (p_{ij} \mid \bar{x}_{ij} = 1)) \\
&s_j \in [r_j, \infty), \ \text{all } j
\end{aligned}
$$

If M_i^* is the optimal makespan on machine i for each i, then the optimal makespan overall is $\max_i\{M_i^*\}$.

The subproblem does not separate in this way if there are precedence constraints between jobs, because the time at which a job can be scheduled may depend on the times at which jobs are scheduled on other machines. Yet even when there are precedence constraints, separability can be preserved if they involve only jobs that must be scheduled on the same machine. Thus, if jobs j and k must be scheduled on the same machine, and j must precede k, one can add constraint $x_j = x_k$ to the master problem and the constraint $s_j + d_{ij} \leq s_k$ to the scheduling problem on every machine.

As noted above, the initial solution of the master problem assigns job 4 to machine B and the other jobs to machine A. The minimum

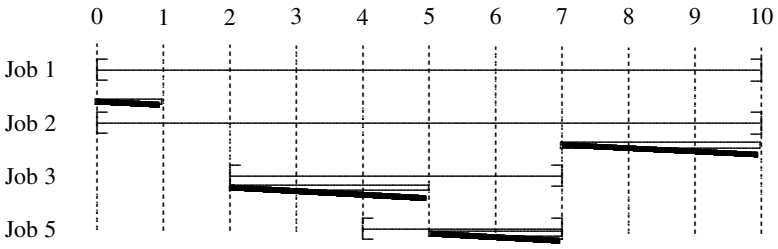

Fig. 2.5 A minimum makespan schedule on machine A for jobs 1, 2, 3, and 5 of Table 2.5. The horizontal lines represent time windows.

makespan schedule on machine B simply starts job 4 at its release time, resulting in a makespan of 8. Thus, whenever job 4 is assigned to machine B (i.e., whenever $x_{4\mathrm{B}} = 1$), the makespan will be at least 8. This gives rise to the Benders cut

$$M \geq 8x_{\mathrm{B}4}$$

Any solution that achieves a makespan better than 8 must avoid assigning job 4 to machine B.

The minimum makespan schedule for jobs 1, 2, 3, and 5 on machine A appears in Figure 2.5. The resulting makespan is 10, which produces the Benders cut

$$M \geq 10(x_{A1} + x_{A2} + x_{A3} + x_{A5} - 3) \tag{2.30}$$

Thus, any solution that obtains a makespan better than 10 must avoid assigning at least one of these jobs to machine A.

The master problem is now

$$\begin{aligned}
&\min\ M \\
&\text{inequalities (2.27)–(2.29)} \\
&M \geq 10 \\
&M \geq 10(x_{\mathrm{A}1} + x_{\mathrm{A}2} + x_{\mathrm{A}3} + x_{\mathrm{A}5} - 3) \\
&x_{ij} \in \{0, 1\}, \quad \text{all } i, j
\end{aligned}$$

Solution of this problem results in the same machine assignments as before, but the optimal value is now 10. Since this is a lower bound on the minimum makespan, and a feasible solution with makespan 10 was found by solving the subproblem, the algorithm terminates. The schedule found by solving the subproblem is optimal.

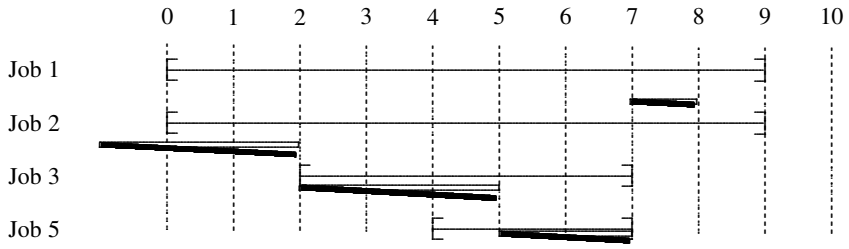

Fig. 2.6 Edge-finding proof of infeasibility of scheduling jobs 1, 2, 3, and 5 on machine A to achieve a makespan of 9.

In practice, the success of a Benders method often rests on finding strong Benders cuts. One way to deduce stronger cuts is to examine more closely the reasoning process that proves optimality in the subproblem. In the present case, a well-known *edge-finding* technique, deduces that the makespan on machine A is still 10 even if only jobs 2, 3, and 5 (and not job 1) are assigned to the machine. This is done by deducing that if the makespan is less than or equal to 9, job 2 must start before its release time (Fig. 2.6). One can therefore state a stronger Benders cut than (2.30):

$$M \geq 10(x_{A2} + x_{A3} + x_{A5} - 2) \tag{2.31}$$

Edge finding must often be combined with other procedures, such as branching, but one can nonetheless keep track of which jobs play a role in deducing the optimal makespan.

One difficulty with a cut of the form (2.31) is that it imposes no bound on M when a proper subset of jobs 2, 3, and 5 is assigned to machine A. One should be able to say something about the resulting minimum makespan if only jobs 2 and 3 are assigned to this machine, for example. In fact, one can often derive a cut that remains useful in such cases. Section 6.14.3 shows how to do this when all the release times are equal.

Exercises

2.17. Write (2.26) for each release time t_1 and each deadline t_2 ($t_1 < t_2$) for machine A in the problem of Table 2.5. Verify that (2.27) are the nonredundant inequalities.

2.18. Suppose that jobs 1, 2, 3, and 4 from Table 2.5 are assigned to machine A. What is the minimum makespan? Write the corresponding Benders cut containing the fewest variables.

2.19. A number of projects must be carried out in a shop, and each project j must start and finish within a time window $[r_j, d_j]$. Once started, the project must run p_j days without interruption. Only one project can be underway at any one time. Each month, the shop is shut down briefly to clean and maintain the equipment, and no project can be in process during this period. The goal is to find a feasible schedule. Formulate this problem and indicate how to solve it with logic-based Benders decomposition. *Hint:* Let each month's schedule be a subproblem.

2.20. A vehicle routing problem requires that a fleet of vehicles make deliveries to customers within specified time windows. Each customer j must take delivery between e_j and d_j, and vehicle i requires time p_{ijk} to travel from customer j to customer k. The objective is to minimize the number of vehicles. Describe a Benders-based solution method in which the master problem assigns customers to vehicles and the subproblem routes each vehicle. The subproblem for each vehicle is a traveling salesman problem with time windows. The subproblem can be written with variable indices and the circuit constraint (see Section 6.13).

2.9 Routing and Frequency Assignment

A final example illustrates decomposition in a more complex setting. The arcs of a directed network represent optical fibers with limited capacity (Fig. 2.7). There are requests for communication channels to be established between certain pairs of nodes. The channels must be routed over the network, and they must be assigned frequencies so that all the channels passing along any given arc have different frequencies. There are a limited number of frequencies available, and it may not be possible to establish all the channels requested. The objective is to maximize the number of channels established.

The problem can be decomposed into its routing and frequency-assignment elements. The routing problem is amenable to an MILP formulation, and the frequency assignment problem is conveniently written with alldiff constraints—provided that a subproblem constraint is used to fix the flows before the frequency assignment problem is stated.

The routing problem is similar to the well-known multicommodity network flow problem. This problem generalizes the capacitated

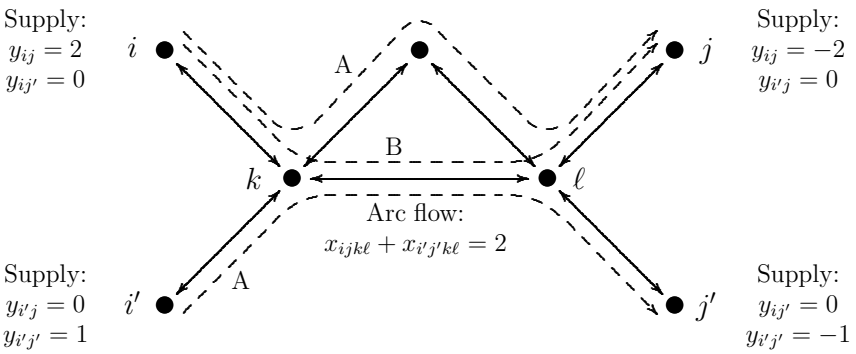

Fig. 2.7 A message routing and frequency assignment problem. Two message channels are requested from i to j and one from i' to j'. The arcs have capacity 2, and frequencies A, B are available. The dashed lines show an optimal solution.

min-cost network flow problem by distinguishing several commodities that must be transported over the network. There is a net supply of each commodity at each node, and the total flow on each arc must be within the arc capacity. In the message routing problem, each origin-destination pair represents a different commodity.

The message routing problem is not identical to the multicommodity flow problem because the net supplies are not fixed, due to the fact that some requests may not be satisfied. As a result, one would not be able to use a global constraint designed for multicommodity flow problems, even if one existed. Nonetheless, it is fairly easy to write the MILP constraints directly.

For each pair of nodes (i, j), let D_{ij} be the number of i-to-j channels requested (possibly zero). A key decision is which requests to honor, and one can therefore let integer variable y_{ij} be the number of channels from i to j that are actually established. (It is assumed here that different channels from i to j can be routed differently.) The net supply of commodity (i, j) is y_{ij} at node i, $-y_{ij}$ at node j, and zero at other nodes. Let $x_{ijk\ell}$ be the flow of commodity (i, j) on arc (k, ℓ), and $C_{k\ell}$ the capacity of the arc. To simplify notation, arcs missing from the network can be viewed as arcs with a capacity of zero. The flow model is

$$\text{MILP:} \begin{cases} \max \sum_{ij} y_{ij} \\ \sum_{\ell \neq i} x_{iji\ell} - \sum_{k \neq i} x_{ijki} = y_{ij}, \quad \text{all } i, j \\ \sum_{\ell \neq j} x_{ijj\ell} - \sum_{k \neq i} x_{ijkj} = -y_{ij}, \quad \text{all } i, j \\ \sum_{\ell \neq i,j,k} x_{ijk\ell} - \sum_{\ell \neq i,j,k} x_{ij\ell k} = 0, \quad \text{all } i, j, k \text{ with } k \neq i, j \\ \sum_{ij} x_{ijk\ell} \leq C_{k\ell}, \quad \text{all } k, \ell \\ x_{ijk\ell} \geq 0, \quad \text{all } i, j, k, \ell \\ 0 \leq y_{ij} \leq D_{ij}, \quad \text{all } i, j \end{cases}$$

$$\text{domains: } x_{ijk\ell}, y_{ij} \in \mathbb{Z}, \quad \text{all } i, j, k, \ell$$

After the communications channels are routed, a frequency f_{ij} can be assigned to each pair i, j so that the frequencies assigned to channels passing through any given arc are all different. The model is therefore completed by writing

$$\text{subproblem:} \begin{cases} \{x_{ijk\ell}, \quad \text{all } i, j, k, \ell\} \\ \text{alldiff: } \{f_{ij} \mid x_{ijk\ell} > 0\}, \quad \text{all } k, \ell \end{cases}$$

$$f_{ij} \in F, \quad \text{all } i, j \text{ with } i \neq j$$

where F is the set of available frequencies.

The problem can be solved by a branch-infer-and-relax method that propagates the alldiff constraint at every node of the search tree. The variables f_{ij} that appear in the propagated alldiff constraint for a given k, ℓ are those for which $x_{ijk\ell}$ is fixed to a positive number by branching. After the relaxation is solved, the search can branch on an alldiff that is violated by the solution of the relaxation. For this purpose, the variables f_{ij} in the alldiff are those for which $x_{ijk\ell}$ has a positive integer value in the solution of the current relaxation. One scheme for branching on a violated alldiff is described in Section 5.1.3.

2.10 Bibliographic Notes

Section 2.2. Various elements of the search-infer-and-relax framework presented here were proposed in [23, 101, 276, 278, 279, 282, 295, 297]. An extension to dynamic backtracking and heuristic methods is given in [286, 287].

Section 2.3. Cuts for general integer knapsack constraints are discussed in [124, 350, 397, 398]. A comprehensive treatment can be found in [26], which strengthens the inequalities discussed here. Nearly all development of knapsack cuts, however, has been concerned with 0-1 knapsack constraints, beginning with [30, 254, 384, 506].

Section 2.4. Conditional modeling is advocated in [295]. The idea of disjunctive modeling goes back at least to [228, 243] and is developed in [31, 32, 50, 104, 245, 295, 335, 409, 437, 483] and elsewhere.

Section 2.5. The employee scheduling model is similar to one presented in [422]. An overview of employee timetabling appears in [354].

Section 2.6. Continuous global optimization is extensively surveyed in [372] and a handbook [388]. The integrated approach taken here is roughly similar to that described in [468, 469, 470] and implemented in the solver BARON. Factorization of functions for purposes of relaxation was introduced by [352]. Global optimization problems can have discrete as well as continuous variables; recent solution technology is discussed in [70].

Section 2.7. The product configuration model is based on [474, 517]. The generic element constraint was introduced by [266]. The form of the constraint used here appears in [474].

Section 2.8. Classical Benders decomposition is due to [73] and was generalized to nonlinear programming in [224]. Logic-based Benders decomposition was introduced in [298] and developed in [279, 296]. Its application to planning and scheduling, with a CP subproblem, was proposed in [279], first implemented in [300], and extended in [257]. The machine scheduling example described here is adapted from [283, 517]. Edge finding originates in [117, 118]. The subproblem constraint is proposed in [293].

Section 2.9. The example is adapted from [459]. The subproblem constraint is unnecessary if the alldiff is an "open" constraint [490], meaning that the argument list depends on the value of another variable. Open constraints are not standard in CP solvers, however.

http://www.springer.com/978-1-4614-1899-3

Integrated Methods for Optimization
Hooker, J.N.
2012, XVIII, 642 p., Hardcover
ISBN: 978-1-4614-1899-3