A Tutorial Guide to Mixed-Integer Programming Models and Solution Techniques

J. Cole Smith and Z. Caner Taşkın
Department of Industrial and Systems Engineering
University of Florida
Gainesville, FL 32611
cole@ise.ufl.edu, taskin@ufl.edu

March 26, 2007

Abstract

Mixed-integer programming theory provides a mechanism for optimizing decisions that take place in complex systems, including those encountered in biology and medicine. This chapter is intended for researchers and practitioners wanting an introduction to the field of mixed-integer programming. We begin by discussing basic mixed-integer programming formulation principles and tricks, especially with regards to the use of binary variables to form logical statements. We then discuss two core techniques, branch-and-bound and cutting-plane algorithms, used to solve mixed-integer programs. We illustrate the use of mixed-integer programming in the context of several medical applications, and close with a featured study on Intensity Modulated Radiation Therapy planning.

1 Introduction

This chapter describes the use of mixed-integer programming in optimizing complex systems, such as those arising in biology, medicine, transportation, telecommunications, sports, and national security. Consider, for instance, an emergency that results in 100 injuries. A triage center is established to administer first aid and assign victims to one of three nearby hospitals, each of which is capable of handling a limited number of patients. Each hospital may have varying equipment and staff levels, and each will be located at a different distance from the emergency. The optimization problem that arises is to assign patients to hospitals in a way that maximizes the effectiveness of care that can be given to the victims, while obeying physical capacity restrictions imposed by the hospitals. Experts often attempt to solve these problems based on intuition and experience, but the resulting solution is almost invariably suboptimal due to the inherent complexities of such problems. In applications of critical importance, there is sufficient motivation to turn to mathematical techniques that can provably obtain a "best" solution.

Mixed-integer programming is a subset of the broader field of mathematical programming. Mathematical programming formulations include a set of variables, which represent actions that can be taken in the system being modelled. One then attempts to optimize (either in the minimization or maximization sense) a function of these variables, which maps each possible set of decisions into a single score that assesses the quality of the solution. These scores are often in units of currency representing total cost incurred or revenue gained. The limitations of the system are included as a set of constraints, which are usually stated by restricting functions of the decision variables to be equal to, not more than, or not less than, a certain numerical value. Another type of constraint can simply restrict the set of values to which a variable might be assigned.

Several applications involve decisions that are *discrete* (e.g., to which hospital an emergency patient should be assigned), while some other decisions are *continuous* in nature (e.g., determining the dosage of fluids to be administered to a patient). On the surface, the ability to enumerate all possible values that a discrete decision can take seems appealing; however, in most applications, the discrete variables are interrelated, requiring an enumeration of all combinations of values that the entire set of discrete variables can take.

What are the implications of complete enumeration techniques on processing time? Suppose that there exist n variables, each of which can take on a value of zero or one. Furthermore, suppose that each configuration of these variables can be evaluated (tested for feasibility to the problem constraints and scored) using n computer operations. Since there are 2 choices for each variable, there are 2^n configurations. Even if we are using a computer capable of processing 10 trillion operations per second (or 10 teraflops, and at the time of this writing, only 58 of the world's top 500 supercomputers are capable of such a feat), if $n = 50$, the computer will take 1.5 hours to finish enumerating all possibilities. One might be tempted to simply let the computer run all night if need be for important problems, and while this is indeed valid for the case in which $n = 50$, the computational growth rate for these problems is astounding: for $n = 60$, the computer will require 80 days to terminate, and for $n = 70$, the computer will require 262 years. Another question regards the evolution of computing speed, noting that faster computers are constantly emerging. If the program must be finished within two hours, the current 10 teraflop machine will permit the solution of problems with $n = 50$. If a quantum leap is discovered that results in the invention of 10,000 teraflop machine, this fictional computer would only be able to handle problems with $n = 60$ within two hours. Computer speedups, however impressive, are simply no match for exponential enumeration problems.

Therefore, a more efficient technique is required to solve problems containing discrete variables. Mixed-integer programming techniques do not *explicitly* examine every possible combination of discrete solutions, but instead examine a subset of possible solutions, and use optimization theory to prove that no other solution can be better than the best one found. This type of technique is referred to as *implicit enumeration*.

This chapter is not a thorough review of integer programming literature, but is intended for technical researchers who may or may not have any familiarity with linear programming, but who are looking for an entry-level introduction to modelling and solution via integer and mixed-integer programming. The text by Wolsey [18] provides an accessible account of fundamental integer programming methods and theory, while the updated classical work of Nemhauser and Wolsey [11] discusses integer programming and combinatorial theory in detail.

We discuss the general form of mixed-integer programming problems in Section 2, and provide general tips for formulating problems as mixed-integer programs. A brief discussion of the branch-and-bound implicit enumeration technique for solving mixed-integer programs, as is relevant to practitioners, is given in Section 3. Next, Section 4 provides an example of mixed-integer programs in a real radiation therapy application, illustrating the material presented in the prior two sections. Finally, we conclude this chapter in Section 5.

2 Modelling Principles

We begin this section by discussing the general form of linear and mixed-integer programming problems in Section 2.1. We then give common steps and principles behind modelling problems of this form in Section 2.2, and suggest a few common ways that mixed-integer variables can be used to model complex conditions arising in real-world scenarios.

2.1 General Form

First, we present the general form of a linear programming problem. Linear programming problems (usually called "linear programs," and abbreviated as "LPs") contain a set of *decision variables*, which are the unknown quantities or decisions that are to be optimized. In the context of linear and mixed-integer programming problems, the function that assesses the quality of the solution, called the "objective function," should be a linear function of the decision variables. An LP will either minimize or maximize the value of the objective function. Finally, the decisions that must be made are subject to certain requirements and restrictions of a system. We enforce these restrictions by including a set of *constraints* in the model. Each constraint requires that a linear function of the decision variables is either equal to, not less than, or not more than, a scalar value. A common condition simply states that each decision variable must be nonnegative. In fact, all linear programming problems can be transformed into an equivalent minimization problem with nonnegative variables and equality constraints [1].

Thus, suppose we denote $x_1, \ldots, x_n$ to be our set of decision variables. Linear programming problems take on the form:

$$\begin{aligned}
\text{Minimize or maximize} \quad & c_1x_1 + c_2x_2 + \cdots + c_nx_n && \text{(1a)}\\
\text{subject to} \quad & a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \ (\leq, =, \text{ or } \geq)\ b_1 && \text{(1b)}\\
& a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \ (\leq, =, \text{ or } \geq)\ b_2 && \text{(1c)}\\
& \cdots \\
& a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \ (\leq, =, \text{ or } \geq)\ b_m && \text{(1d)}\\
& x_j \geq 0 \quad \forall j = 1, \ldots, n. && \text{(1e)}
\end{aligned}$$

Values c_j, $\forall j = 1, \ldots, n$, are referred to as objective coefficients, and are often associated with the costs associated with their corresponding decisions in minimization problems, or the revenue generated from the corresponding decisions in maximization problems. The values $b_1, \ldots, b_m$ are the right-hand-side values of the constraints, and often represent amounts of available resources (especially for $\leq$ constraints) or requirements (especially for $\geq$ constraints). The a_{ij}-values thus typically denote how much of resource/requirement i is consumed/satisfied by decision j.

Note that nonlinear terms are not allowed in the model, prohibiting for instance the multiplication of two decision variables, the maximum of several variables, or the absolute value of a variable. (These conditions are often desired, but must be achieved by different techniques.) Also, any inequalities present in the model are never strict.

Problems of the form (1) are called "linear programs" because the objective function and constraint functions are all linear. Integer programs (IPs) are stated in an identical fashion, except that all decision variables are constrained to take on integer values. (Hence, integer programs are sometimes called "integer linear programs.") A mixed-integer program (MIP) is a linear program with the added restriction that some, but not necessarily all, of the variables must be integer-valued. Several studies also replace the term "integer" with "0-1" or "binary" when variables are restricted to take on either 0 or 1 values. For the purposes of this chapter, we focus on MIPs, with IPs being modelled and solved as a special case of MIPs.

A solution that satisfies all constraints is called a *feasible solution.* Feasible solutions that achieve the best objective function value (according to whether one is minimizing or maximizing) are called *optimal solutions.* Sometimes no solution exists to an MIP, and the MIP itself is called *infeasible.* On the other hand, some feasible MIPs have no optimal solution, because it is possible to achieve infinitely good objective function values with feasible solutions. Such problems are called *unbounded.*

2.2 Modelling Mixed-Integer Programming Problems

The modelling of complex systems using mixed-integer programs is often more of an art than a science. Typically, a three-step looped process is used to model MIPs. The first step involves defining a set of decision variables that represent choices that must be optimized in the system. The second step usually involves the statement of constraints in the model, with the third step requiring the statement of an objective function (although the last two steps can be done in either order).

It is very common, though, to recognize during model construction that the initial set of decision variables defined for the model are inadequate. Often, decision variables that seem to be implied consequences of other actions must also be defined. The addition of new variables after an unsuccessful attempt at formulating constraints and objectives is the "loop" in the process.

The correct definition of decision variables can be especially complicated in modelling with integer variables. If one is allowed to use binary variables in a formulation, it is possible to represent yes-or-no decisions, enforce if-then statements, and even permit some sorts of nonlinearity in the model (which can be transformed to an equivalent mixed-integer *linear* program).

To illustrate the modelling process, we consider the following three example systems.

Example 1. An outbreak of an infectious disease has been observed in a set N of locations. There exists a set M of teams capable of investigating these outbreaks. Team $i \in M$ can conclude its investigation of the outbreak in location $j \in N$ in t_{ij} hours. Each team can either investigate zero, one, or two outbreaks. If a team investigates two outbreaks, they must travel from one location to the next. The travel time from location $j_1 \in N$ to location $j_2 \in N$ is $d_{j_1 j_2}$. Once all outbreaks have been investigated, a disease control center can take action to combat the outbreak. The goal is to minimize the amount of time necessary to complete the investigation of all locations. □

Example 2. A medical practice is attempting to acquire a certain drug from a set M of suppliers. The practice wishes to have a stock of d_t units of this drug in month t, for $t = 1, \ldots, \bar{t}$. Purchasing one unit of the drug from supplier $i \in M$ during time period $t \in \{1, \ldots, \bar{t}\}$ costs c_{it} dollars. However, in order to purchase drugs from supplier $i \in M$, at any time period, the practice must purchase a minimum of ℓ_i units of the drug during that time period. Fortunately, the practice has room for h units of inventory, and so at most h units of the drug can be stored from one period to the next. If the practice finds itself with too many units of the drug, it can simply throw away the extra supply. The goal is to minimize the cost required to purchase the drugs for time periods $1, \ldots, \bar{t}$. □

Example 3. A nurse is assigned a set N of patients. For each patient $i \in N$, the nurse must spend p_i minutes examining the patient. The nurse must then wait somewhere between ℓ_i and u_i minutes before checking up on the patient, which requires q_i minutes. The nurse, of course, cannot be in two places at the same time, although we will assume for simplicity that the travel time to walk from one patient's room to another is zero. The objective is to minimize the total amount of time required to tend to all patients. For instance, suppose that N contains three patients:

- Patient 1's first visit lasts $p_1 = 5$ minutes. The patient will then be checked on between $\ell_1 = 30$ and $u_1 = 45$ minutes later, after which the nurse will spend $q_1 = 5$ more minutes of care on the second visit.
- Patient 2 needs $p_2 = 10$ minutes of initial care, has an inter-care gap of $\ell_2 = 25$ and $u_2 = 35$ minutes, and requires $q_2 = 5$ minutes of further care.
- Patient 3 needs $p_3 = 10$ minutes of initial care, has an inter-care gap of $\ell_3 = 30$ and $u_3 = 35$ minutes, and requires $q_3 = 10$ minutes of further care.

One solution would start treatment on patients 1, 2, and 3 in this order. The nurse's schedule would then be optimized according to the first column of Table 1 and the total treatment time would be 65 minutes. However, if the patients are treated in the opposite order, the total treatment time becomes only 60 minutes, as shown in the second column of Table 1. Indeed, the timing involved in this problem is quite difficult to optimize by hand. □

Attempting to formulate any of these problems as an LP is problematic. With continuous variables, the first example will likely end up assigning part of a team to one location, and part of a team to another location. The second one will not be able to represent the minimum purchase aspect of the problem. The third will not necessarily be able to keep the nurse from splitting attention simultaneously among multiple patients. Before attempting

Patients ordered 1, 2, 3	Patients ordered 3, 2, 1
Time 0−5: Patient 1	Time 0−10: Patient 3
Time 5−15: Patient 2	Time 10−20: Patient 2
Time 15−25: Patient 3	Time 20−25: Patient 1
Time 25−35: Idle	Time 25−40: Idle
Time 35−40: Patient 1	Time 40−50: Patient 3
Time 40−45: Patient 2	Time 50−55: Patient 2
Time 45−55: Idle	Time 55−60: Patient 1
Time 55−65: Patient 3	

Table 1: Two schedules for the nurse scheduling problem.

to formulate these problems, let us discuss a few common tips and tricks in modelling with integer variables.

1. *Integrality of quantities.* In staffing and purchasing decisions, it is often impossible to take fractional actions. One cannot hire, for instance, 6.5 new staff members, or purchase 1.3 hospital beds. The most obvious use of integer variables thus arises in requesting integer amounts of quantities that can only be ordered in integer amounts. In general, the optimal solution of an integer program need not be a rounded-off version of an optimal solution to a linear program.

2. *If-then statements.* Consider two continuous (i.e., possibly fractional) variables, x and y, defined so that $0 \le x \le 10$ and $0 \le y \le 10$. Suppose we wish to make a statement that if $x > 4$, then $y \le 6$. (If $x \le 4$, then we do not wish to further constrain y.) On the surface, since no integer quantities are requested, it does not appear that integer variables will be necessary. However, the general form of linear programs as given in equations (1) does not permit if-then statements like the one above. Instead, if-then statements can be enforced with the aid of a binary variable, z. We wish to make $z = 1$ if $x > 4$ (note that we make no claims on z if $x \le 4$). This can be accomplished by adding the constraint

$$x \le 4 + 6z, \tag{2}$$

since the event that $x > 4$ implies that $z = 1$. (Even if $z = 1$, the largest value for x is 10, which now makes a constraint of the form $x \le 10$ unnecessary.) If $z = 1$, then we must also require that $y \le 6$. This is achieved by reducing the upper bound of 10 on y to 6 if z is equal to 1 as follows:

$$y \le 10 - 4z, \tag{3}$$

where once again, the bound constraint $y \le 10$ may now be omitted. In general, suppose we wish to make the following statement: "if $q_1x_1 + \cdots + q_nx_n > Q$, then

$r_1x_1 + \cdots + r_nx_n \leq R$." We would include the following conditions in our model:

$$q_1x_1 + \cdots + q_nx_n \leq Q + M'z \tag{4}$$
$$r_1x_1 + \cdots + r_nx_n \leq M'' - (M'' - R)z \tag{5}$$
$$z \text{ binary}, \tag{6}$$

where M' and M'' are "sufficiently large" constants. These values should be just large enough to not add unintentional restrictions to the model. For instance, we are not attempting to place any hard restriction on the quantity $q_1x_1 + \cdots + q_nx_n$ (written conveniently as $\mathbf{q}^\top\mathbf{x}$ in vector form). If $z = 1$, the upper bound on $\mathbf{q}^\top\mathbf{x}$ is $Q + M'$, and hence M' must be large enough so that even if constraint (4) is removed from the model, $\mathbf{q}^\top\mathbf{x}$ would still never be more than $Q + M'$. Likewise, if $z = 0$, we must choose a value M'' large enough in (5) such that $\mathbf{r}^\top\mathbf{x}$ could never be more than M'' even without the restriction (5). It is worth noting that assigning arbitrarily large values for M' and M'' is *not* recommended, for reasons that will become more apparent in Section 3.

3. *Enforce at least k out of p restrictions.* This situation is similar to if-then constraints in the way we model such restrictions. For a simple example, suppose we have nonnegative variables $x_1, \ldots, x_n$, and wish to require that at least three of these variables take on values of 5 or more. Then we can define variables $z_1, \ldots, z_n$, such that if $z_j = 1$, then $x_j \geq 5$, $\forall j = 1, \ldots, n$. This simple if-then constraint can easily be modelled by employing the following constraints:

$$x_j \geq 5z_j \quad \forall j = 1, \ldots, n. \tag{7}$$

Clearly, if $z_j = 1$, then $x_j \geq 5$. If $z_j = 0$, it is still possible for $x_j \geq 5$, but no such restrictions are enforced. We need to guarantee that three variables take on values of 5 or more, and so we add the following "k-out-of-p" constraint:

$$z_1 + \ldots + z_n = 3. \tag{8}$$

Again, this constraint does not state that exactly three variables will be at least 5, but rather that at least three variables are guaranteed to be at least 5. This same trick can be used to enforce the condition that at least k out of p sets of constraints are satisfied, and so on, often by using M-values as introduced in the section on if-then constraints.

4. *Nonlinear product terms.* In some circumstances, nonlinear terms can be transformed into linear terms by the use of linear constraints. First, note that if x_j is a binary variable, then $x_j = x_j^q$ for any positive constant q. After that substitution is made, suppose that we have a nonlinear term of the form $x_1 \cdot x_2 \cdots x_k \cdot y$, where $x_1, \ldots, x_k$ are binary variables and $0 \leq y \leq u$ is another variable, either continuous or integer. That is, all but perhaps one of the terms is a binary variable. First, replace the nonlinear term with a single continuous variable, w. Using the if-then concept expressed above, note that if x_j equals zero for any $j \in \{1, \ldots, k\}$, then w equals zero as well. Also,

note that w can never be more than the upper bound, u, on the y-variable. Hence, we obtain the constraints

$$w \leq ux_j \quad \forall j = 1, \dots, k. \tag{9}$$

Of course, to guarantee that w *equals* zero in case any x_j-variable equals to zero, we must also state a nonnegativity constraint

$$w \geq 0. \tag{10}$$

Now, suppose that all $x_1 = \cdots = x_k = 1$. In this case, we need to add constraints that enforce the condition that $w = y$. Regardless of the x-variable values, w cannot be more than y, and so we state the constraint

$$w \leq y. \tag{11}$$

However, in order to get the constraint "$w \geq y$ if $x_1 = \cdots = x_k = 1$," we include the constraint

$$w \geq u(x_1 + \cdots + x_k - k) + y. \tag{12}$$

If each x-variable equals to 1, then (12) states that $w \geq y$, which along with (11) guarantees that $w = y$. On the other hand, if at least one $x_j = 0$, $j \in \{1, \dots, k\}$, then the term $u(x_1 + \cdots + x_k - k)$ is not more than $-u$, and the right-hand-side of (12) is not positive; hence, (12) allows w to take on the correct value of zero (as would be enforced by (9) and (10)). As a final note, observe that even if y is an integer variable, we need not insist that w is an integer variable as well, since (9)$-$(12) guarantees that $w = x_1 \cdots x_k \cdot y$, which must be an integer given integer x- and y-values.

Let us now briefly continue the examples introduced earlier in this section in the light of the modelling tricks introduced above.

Example 1, continued. Define binary decision variables x_{ij}, which equal one if team $i \in M$ investigates an outbreak at location $j \in N$, and zero otherwise. Proceeding with this initial set of variables, we would then attempt to formulate the constraints of the problem. In fact, the only restrictions encountered thus far is the fact that each location must be investigated by exactly one team, the fact that each team can investigate no more than two locations, and the fact that each x-variable must be binary. These constraints are respectively given by:

$$\sum_{i \in M} x_{ij} = 1 \quad \forall j \in N \tag{13}$$

$$\sum_{j \in N} x_{ij} \leq 2 \quad \forall i \in M \tag{14}$$

$$x_{ij} \text{ binary} \quad \forall i \in M, j \in N. \tag{15}$$

Finally, we would attempt to formulate the objective function. Note that the amount of time required for team $i \in M$ to complete their investigation(s) is the amount of time required to perform the investigations, $\sum_{j\in N} t_{ij}x_{ij}$, plus the travel time between location sites. This travel time can be represented by $\sum_{\text{all } j_1\neq j_2\in N} d_{j_1j_2}x_{ij_1}x_{ij_2}$, noting that inter-location travel time is required only if a team is required to visit both sites. The objective function has the following form:

$$\text{Minimize maximum}_{i\in M}\left\{\sum_{j\in N} t_{ij}x_{ij} + \sum_{\text{all } j_1\neq j_2\in N} d_{j_1j_2}x_{ij_1}x_{ij_2}\right\}.$$

This function is nonlinear for two reasons. One, the investigation completion time is a nonlinear function due to the multiplication of x-variables. Two, the quantity being minimized is the maximum of the teams' investigation completion times.

At this point, we must add additional variables to the problem. Replace each nonlinear term $x_{ij_1}x_{ij_2}$ with a new variable $w_{ij_1j_2}$, and add linearization constraints

$$w_{ij_1j_2} \leq x_1 \text{ similar to (9)} \tag{16a}$$

$$w_{ij_1j_2} \geq 0 \text{ similar to (10)} \tag{16b}$$

$$w_{ij_1j_2} \leq x_2 \text{ similar to (11), treating } x_2 \text{ as ``}y\text{''} \tag{16c}$$

$$w_{ij_1j_2} \geq x_1 + x_2 - 1 \text{ similar to (12).} \tag{16d}$$

The objective function now becomes:

$$\text{Minimize maximum}_{i\in M}\left\{\sum_{j\in N} t_{ij}x_{ij} + \sum_{\text{all } j_1\neq j_2\in N} d_{j_1j_2}w_{ij_1j_2}\right\}.$$

In order to remove the "minimax" structure of this objective function, we rely on a common trick for linear programming. First, we add a variable θ that represents the maximum completion time. We would then minimize θ, subject to the conditions that θ must be at least as large as the completion time for team 1, θ must be at least as large as the completion time for team 2, and so on, for each team. These constraints are:

$$\theta \geq \sum_{j\in N} t_{ij}x_{ij} + \sum_{\text{all } j_1\neq j_2\in N} d_{j_1j_2}w_{ij_1j_2} \quad \forall i \in M. \tag{17}$$

Of course, the question arises as to why θ will be *equal* to the maximum of the teams' completion times, since (17) merely enforces the condition that θ is at least as large as the maximum of these times. The answer is that the objective function will minimize θ, and so θ will take on the smallest possible value permitted by (17), which will indeed be the maximum of completion times.

The overall model is then given by:

$$\begin{array}{ll} \text{Minimize} & \theta \\ \text{subject to} & \text{Constraints (13) -- (17).} \end{array}$$

Example 2, continued. An initial attempt at modelling this problem would define decision variables x_{it}, $\forall i \in M,\ t = 1, \ldots, \bar{t}$, equal to the amount of drugs purchased from supplier i in period t. It is also necessary in general to define variables g_i, $\forall i \in M$, which denote how many units of drugs are thrown away after period i (because we purchased too many from a supplier due to minimum purchase limits, and perhaps also due to inventory limits). However, we must ensure that the number of drugs purchased from supplier $i \in M$ at any time period is either zero or at least ℓ_i. Hence, let us define binary decision variables z_{it}, which equal one if we purchase any supplies from supplier $i \in M$ at time period $t \in \{1, \ldots, \bar{t}\}$, and zero otherwise.

However, stating the demand and inventory constraints is awkward (though possible) without another set of variables. Note that we must require $\sum_{i \in M} x_{i1} - g_1 \geq d_1$ in order to satisfy period 1 demand. In period 2, we have that the inventory plus the amount of drugs ordered in period 2, minus whatever is thrown away after period 2, must be at least d_2. This condition is stated as $((\sum_{i \in M} x_{i1} - g_1) - d_1) + \sum_{i \in M} x_{i2} - g_2 \geq d_2$. For period 3, this constraint becomes $(((\sum_{i \in M} x_{i1} - g_1) - d_1) + (\sum_{i \in M} x_{i2} - g_2) - d_2) + \sum_{i \in M} x_{i3} - g_3 \geq d_3$. It is evident that the expression quickly becomes quite large as the time period t increases.

Instead, we can define inventory variables y_t, for $t = 1, \ldots, \bar{t}$, which represent the amount of drug inventory remaining after period t. From this point, it is easier to establish "flow balance" constraints, which state that the amount of drugs coming into the practice at each period (inventory from the previous period, plus amount purchased from suppliers during the current period) equals to the drugs that leave the practice at the current period (those demanded by patients, plus those put into inventory after the period, plus those thrown away). These balance constraints are given as

$$y_{t-1} + \sum_{i \in M} x_{it} = d_t + y_t + g_t \quad \forall t = 1, \ldots, \bar{t}, \tag{18}$$

where y_0 is defined to be zero. The minimum purchase quantity constraints are given as

$$x_{it} \geq \ell_i z_{it} \quad \forall i \in M,\ t = 1, \ldots, \bar{t} \tag{19}$$

$$x_{it} \leq M_{it} z_{it} \quad \forall i \in M,\ t = 1, \ldots, \bar{t}, \tag{20}$$

where M_{it} is a sufficiently large constant, $\forall i,\ t$. If $z_{it} = 1$, then $\ell_i \leq x_{it} \leq M_{it}$, while if $z_{it} = 0$, then $x_{it} = 0$. A possible value for M_{it} would be the maximum of ℓ_i and the remaining demands $\sum_{u=t}^{\bar{t}} d_u$. Finally, we require constraints that state bounds on the x-, g-, and z-variables.

$$x_{it} \geq 0, \text{ and } z_{it} \text{ binary} \quad \forall i \in M,\ t = 1, \ldots, \bar{t} \tag{21}$$

$$0 \leq y_t \leq h \text{ and } g_t \geq 0 \quad \forall t = 1, \ldots, \bar{t}. \tag{22}$$

The overall formulation can now be stated as:

$$\begin{aligned} \text{Minimize} \quad & \sum_{i \in M} \sum_{t=1}^{\bar{t}} c_{it} x_{it} && (23) \\ \text{subject to} \quad & \text{Constraints (18) – (22).} \end{aligned}$$

Example 3, continued. There exist several methods of modelling and solving this problem. One technique defines a continuous variable f_i to be the first time that the nurse sees patient $i \in N$, and s_i to be the second time that the nurse sees patient $i \in N$. These variable definitions allow us to state the minimum and maximum gaps between the first and second visits by the nurse:

$$s_i - (f_i + p_i) \geq \ell_i \quad \forall i \in N \tag{24}$$
$$s_i - (f_i + p_i) \leq u_i \quad \forall i \in N. \tag{25}$$

However, we must now enforce the restriction that the nurse does not tend to two or more patients at the same time. We must ensure that for each pair of nurse visits (first visits to different patients, first and second visits to different patients, and second visits to different patients), one visit starts before the other begins, or vice versa. (The two visits to the same patient are disjoint due to (24).)

Define binary variables z_{ij}^{11} to equal one if the first visit to patient $i \in N$ occurs before the first visit to patient $j \in N$, and zero otherwise. Also, define z_{ij}^{12} if the first visit to patient $i \in N$ occurs before the second visit to patient $j \in N$, and zero otherwise. Similarly, z_{ij}^{21} relates the second visit of patient $i \in N$ to the first visit of patient $j \in N$, and z_{ij}^{22} relates the second visit of patient $i \in N$ to the second visit of patient $j \in N$. The following constraints relate the z-variables to the f- and s-variables.

$$f_j - (f_i + p_i) \geq -M_{ij}^{11}(1 - z_{ij}^{11}) \quad \forall i, j \in N,\ i \neq j \tag{26}$$
$$s_j - (f_i + p_i) \geq -M_{ij}^{12}(1 - z_{ij}^{12}) \quad \forall i, j \in N,\ i \neq j \tag{27}$$
$$f_j - (s_i + q_i) \geq -M_{ij}^{21}(1 - z_{ij}^{21}) \quad \forall i, j \in N,\ i \neq j \tag{28}$$
$$s_j - (s_i + q_i) \geq -M_{ij}^{22}(1 - z_{ij}^{22}) \quad \forall i, j \in N,\ i \neq j, \tag{29}$$

where the M-values once again are sufficiently large constants. For instance, (26) states that if $z_{ij}^{11} = 1$, then $f_j \geq f_i + p_i$, meaning that the nurse finishes the visit to patient i before the visit to patient j occurs. We now need to state constraints ensuring that one visit finishes before the next one starts, or vice versa.

$$z_{ij}^{11} + z_{ji}^{11} = 1 \quad \forall i, j \in N,\ i < j \tag{30}$$
$$z_{ij}^{22} + z_{ji}^{22} = 1 \quad \forall i, j \in N,\ i < j \tag{31}$$
$$z_{ij}^{12} + z_{ji}^{21} = 1 \quad \forall i, j \in N,\ i \neq j. \tag{32}$$

Constraints (30) and (31) respectively ensure that no pair of first visits and no pair of second visits overlap. Constraints (32) ensure that no pair of first/second visits overlaps. (In fact, the number of z-variables in the model can now be halved by substituting out z-variables according to (30), (31), and (32). However, we retain these variables here for ease of exposition.)

Finally, we again have a minimax objective in which the time of the final patient visit must be minimized. Again, we define θ to be the time of the final patient visit, which must equal to $s_i + q_i$, for some $i \in N$. The final model is then given as:

$$\begin{aligned}
\text{Minimize} \quad & \theta && (33)\\
\text{subject to} \quad & \text{Constraints } (24) - (32) && \\
& \theta \geq s_i + q_i \quad \forall i \in N && (34)\\
& s_i,\ f_i \geq 0 \quad \forall i \in N && (35)\\
& z_{ij}^{11},\ z_{ij}^{12},\ z_{ij}^{21},\ z_{ij}^{22} \text{ binary} \quad \forall i, j \in N, i \neq j. && (36)
\end{aligned}$$

This problem is in fact adapted from a study on radar pulse interleaving, which contains similar challenges to this nurse scheduling problem. See [4, 6, 16] for a more thorough examination of interleaving applications and integer programming techniques.

3 MIP Solution Techniques

Often, there are alternative ways of modelling optimization problems as MIPs. There sometimes exist trade-offs in these different modelling approaches. Some models may be smaller (in terms of the number of constraints and variables required), but may be more difficult to solve than larger models. In fact, the difference can be significant, and can make a difference in whether or not MIPs can be solved quickly enough to be practically useful. (For instance, the situation described in Example 1 mentioned in the previous section must be solved before the outbreak spreads!)

Improving the efficiency of solving MIP models requires an understanding of how MIP solvers work. Premium MIP solvers, such as CPLEX (ILOG, Inc.), Xpress-MP (Dash Optimization), SYMPHONY and CBC (COIN-OR project), and Solver (Frontline Systems, Inc.), employ a combination of *branch-and-bound* and *cutting-plane* techniques. While a review of how to use these software packages is well beyond the scope of this tutorial, it is important to understand the basics of MIP solution algorithms in order to understand the key principles in MIP modelling.

For the rest of this section, assume that we are solving a minimization MIP. (Maximization MIPs are solved in much the same fashion.) To help illustrate the branch-and-bound process, we consider the following example MIP (actually a pure IP since all variables are integer).

$$\begin{aligned}
\text{Minimize} \quad & 4x_1 + 6x_2 && (37a)\\
\text{subject to} \quad & 2x_1 + 2x_2 \geq 5 && (37b)\\
& x_1 - x_2 \leq 1 && (37c)\\
& x_1, x_2 \geq 0 \text{ and integer.} && (37d)
\end{aligned}$$

The first concept that we discuss in solving MIPs is that of *relaxations*. A relaxation of an MIP is a problem such that (a) any solution to the MIP corresponds to a feasible solution to the relaxed problem, and (b) each solution to the MIP has an objective function value greater than or equal to that of the corresponding solution to the relaxed problem. The most commonly used relaxation for an MIP is its *LP relaxation*, which is identical to the MIP with the exception that variable integrality restrictions are eliminated. Clearly, any integer-feasible solution to the MIP is also a solution to its LP relaxation, with matching objective function values.

Envision a bag containing several orange and blue marbles. Each marble represents a solution to the LP relaxation, but only the orange marbles also represent solutions to the MIP. Each marble has a weight, corresponding to its objective function value. The task in linear programming is to find the lightest marble. The task in solving an MIP is to find the lightest orange marble.

When describing the branch-and-bound algorithm for MIPs, it is helpful to know how LPs are solved. See [1, 8, 15, 17] for an explanation of linear programming theory and methodology. For the purposes of this chapter, we simply note that linear programs can be solved quickly (in time bounded by a polynomial of the problem's input size). Graphically, Figure 1 illustrates the feasible region (set of all feasible solutions) to the LP relaxation of formulation (37). Note that the gradient of the objective function is (4,6) (taken from the coefficients of (37a)). This means that the objective function is increasing in this direction, and hence we wish to follow the direction $(-4, -6)$ as far as possible in the feasible region. Put another way, think of $(-4, -6)$ as the direction of gravity, and place a pebble in the feasible region. The point to which the pebble falls, (1.75, 0.75), is the optimal solution to the LP relaxation, and has an objective function value of 11.5.

Returning to the bag of marbles analogy, solving the LP relaxation has yielded a blue marble (fractional, not integer, solution) with a weight of 11.5. This implies that all orange marbles have a weight of 11.5 or more, since weight of the lightest marble in the bag was 11.5. (In general, we cannot claim that the optimal solution to the LP is unique, and so we allow for the possibility that MIP solutions exist with an identical objective function to the optimal LP solution.) The important result is that a *lower bound* on the optimal MIP solution is obtained from the LP relaxation. No solution to the MIP (37) can be found with an objective function value less than 11.5.

Of course, the solution (1.75, 0.75) is not a feasible solution to (37). In fact, all feasible solutions have the trait that *either* $x_1 \leq 1$ *or* $x_1 \geq 2$. In fact, we can split the problem (37) into two subproblems: one in which $x_1 \leq 1$ (called region 1), and one in which $x_1 \geq 2$ (called region 2). All solutions to the original MIP are contained in exactly one of these two new subproblems. This process is called *branching*, and we could have also branched on x_2 instead, by requiring that either $x_2 \leq 0$ or $x_2 \geq 1$. Conceptually, branching is equivalent to taking our bag of marbles and splitting it into two bags. All blue marbles corresponding to solutions in which $1 < x_1 < 2$ are thrown away (importantly, this includes the lightest marble that we previously found). No orange marbles are thrown away: they either belong to the "$x_1 \leq 1$" bag or the "$x_1 \geq 2$" bag. Now, we will search for the lightest orange marble in each bag, compare them, and take the lightest of the two.

The feasible regions of the two new subproblems are depicted in Figure 2. When $x_1 \leq 1$, the optimal solution is (1, 1.5) with objective function value 13. When $x_1 \geq 2$, the optimal solution is (2, 1) with objective function value 14. In the $x_1 \leq 1$ region, the lower bound is 13. In the $x_1 \leq 2$ region, though, the best solution happens to be an integer solution. That is, the lightest marble in this bag out of all marbles happens to be orange, and so it is obviously also the lightest orange marble. Therefore, the best integer solution in the $x_1 \geq 2$ region has an objective function value of 14; there is no need to further search that region. This region is said to be fathomed by integrality. We store the solution (2, 1), and call it

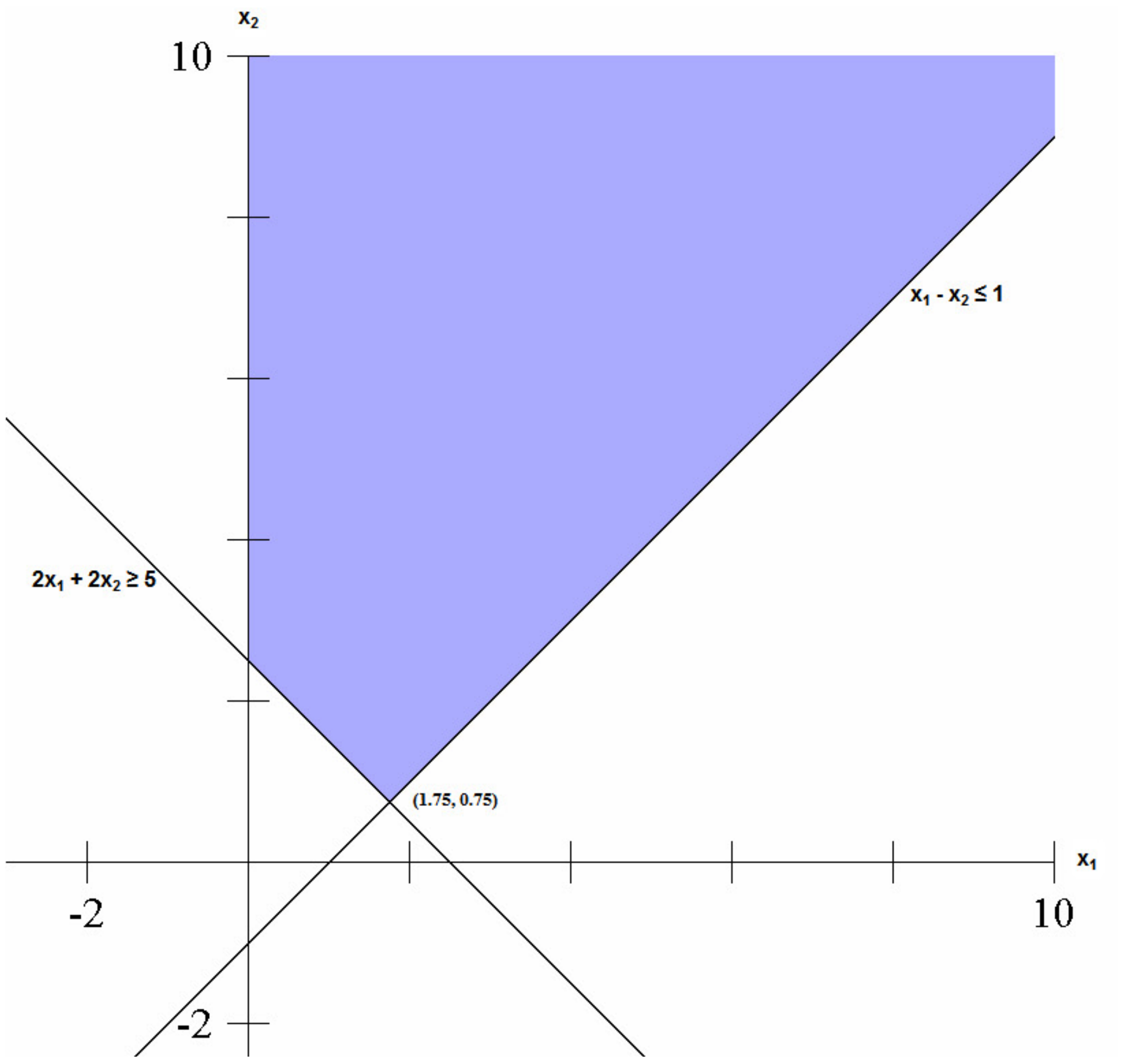


Figure 1: Feasible region of the LP relaxation

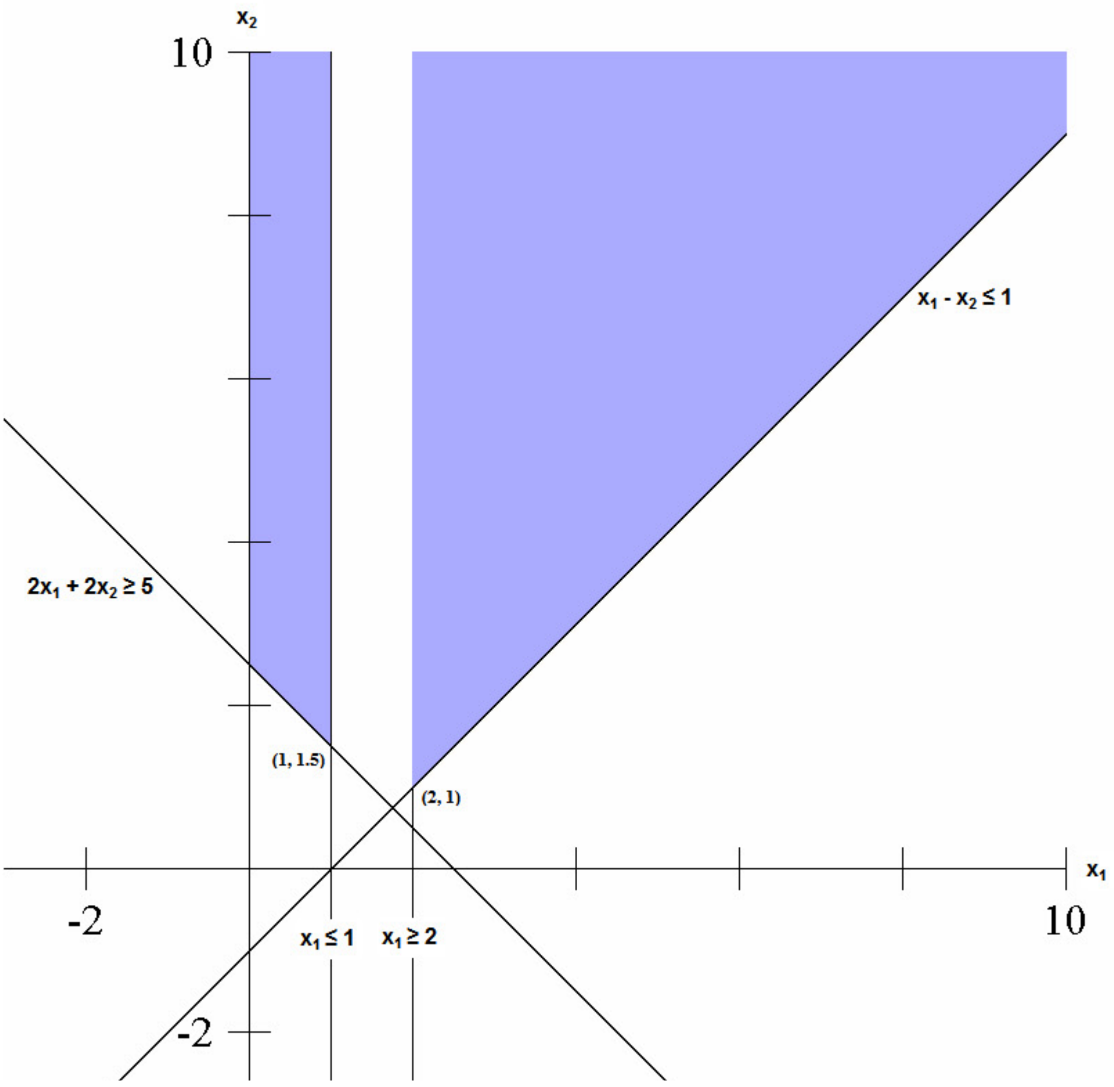


Figure 2: Feasible regions of the subproblems

our *incumbent solution.* If no better solution is found, it will become our optimal solution.

At this point, there is one *active* region (or "active node" in the context of branch-and-bound trees, which we will describe shortly), which is region 1. An active region is one that has not been branched on, and that must still be explored, because there is a possibility that it contains a solution better than the incumbent solution. The initial region is not active, because we have branched on it. Region 2 is not active because we have found the best integer solution in that region. Region 1, however, is still active and must be explored. The lower bound over this region is 13; thus, we know that the optimal solution to the entire problem must have an objective function value somewhere between 13 and 14 (inclusive). We recursively divide region 1, in which $x_1 \leq 1$. Since the optimal solution in this region was (1, 1.5), we branch by creating two new subproblems: one in which both $x_1 \leq 1$ and $x_2 \leq 1$ (called region 3), and one in which both $x_1 \leq 1$ and $x_2 \geq 2$ (called region 4). Once

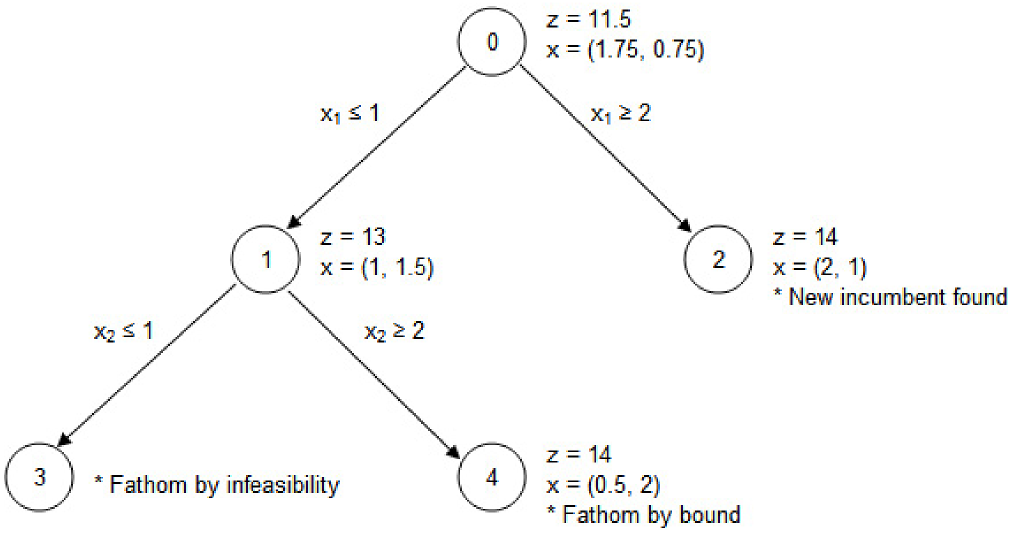


Figure 3: Branch-and-Bound tree

again, all integer solutions in region 1 are contained in either region 3 or region 4.

However, note that region 3 is empty, because the stipulation that both x_1 and x_2 are no more than 1 makes it impossible to satisfy (37b). There are therefore no integer solutions in this region either, and so we stop searching region 3. This region is said to be fathomed by infeasibility. The optimal solution to region 4's linear relaxation is (0.5, 2), with objective function value 14. We still have not found the best integer solution over region 4, but we know that the best integer solution (if one even exists) has an objective function value of 14 or more. However, our incumbent solution has an objective function value of 14. We have *not* found the best integer solution in region 4, but we do know that the best solution in region 4 will not improve the incumbent solution we have found. Thus, we are not interested in any integer feasible solution in region 4, and we stop searching that region. (An alternative optimal integer solution can exist in that region, but we are not seeking to find *all* optimal solutions, just one.) Region 4 is said to be fathomed by bound.

Figure 3 depicts a tree representation of this search process, which is called the "branch-and-bound tree." Each node of the tree represents a feasible region. Now, there are no more regions to be examined (no more active nodes), and the algorithm terminates with the incumbent solution, (2, 1), as an optimal solution.

A formal description of the branch-and-bound algorithm for minimization problems is given as follows.

Step 0. Set the incumbent objective $v = \infty$ (assuming that no initial feasible integer solution is available). Set the active node count $k = 1$ and denote the original problem as an "active" node. Go to Step 1.

Step 1. If $k = 0$, then stop: the incumbent solution is an optimal solution. (If there is no incumbent, i.e., $v = \infty$, then the original problem has no integer solution.) Else, if $k \geq 1$, go to Step 2.

Step 2. Choose any active node, and call it the "current" node. Solve the LP relaxation of the current node, and make it inactive. If there is no feasible solution, then go to Step 3. If the solution to the current node has objective value $z^* \geq v$, then go to Step 4. Else, if the solution is all integer (and $z^* < v$), then go to Step 5. Otherwise, go to Step 6.

Step 3. Fathom by infeasibility. Decrease k by 1 and return to Step 1.

Step 4. Fathom by bound. Decrease k by 1 and return to Step 1.

Step 5. Fathom by integrality. Replace the incumbent solution with the solution to the current node. Set $v = z^*$, decrease k by 1, and return to Step 1.

Step 6. Branch on the current node. Select any variable that is fractional in the LP solution to the current node. Denote this variable as x_s and denote its value in the optimal solution as f. Create two new active nodes: one by adding the constraint $x_s \leq \lfloor f \rfloor$ to the current node, and the other by adding $x_s \geq \lceil f \rceil$ to the current node. Add 1 to k (two new active nodes, minus one due to branching on the current node) and return to Step 1.

Note that in Step 0, we could run a *heuristic procedure* to quickly obtain a good-quality solution to the MIP with no guarantees on its optimality. This solution would then become our initial incumbent solution, and could possibly help conserve branch-and-bound memory requirements by increasing the rate at which active nodes are fathomed in Step 4. In Step 2, we may have several choices of active nodes on which to branch, and in Step 6, we may have several choices on which variable to perform the branching operation. There has been much empirical research designed to establish good general rules to make these choices, and these rules are implemented in commercial solvers. However, for specific types of formulations, one can often improve the efficiency of the branch-and-bound algorithm by experimenting with node selection and variable branching rules.

The best-case scenario in solving a problem by branch-and-bound is that the original node yields an optimal LP solution that happens to be integer, and the algorithm terminates immediately. Indeed, in (37), if we simply add the constraint $x_1 + x_2 \geq 3$ and solve the LP relaxation, we would obtain the optimal solution (2, 1) immediately. This also underscores the importance of making the M-values introduced in the previous section as small as possible. The smaller these M-values are, the fewer fractional solutions exist in the linear programming relaxation. Paying close attention to these M-values often results in significant improvements in the computational efficiency of MIP algorithms.

Thus, a classical way to reduce the presence of fractional solutions is to find *valid inequalities*, which do not cut off any integer solutions, but do cut off some fractional solutions. In terms of marbles, these inequalities remove some blue marbles from the bag, but never orange marbles, and do so without branching into multiple bags. A *cutting plane* is a valid inequality that removes the optimal LP relaxation solution from the feasible region. In theory, MIPs can be solved without branching either by (a) including enough valid inequalities before solving the LP relaxation, so that the LP relaxation provides an integer solution, or (b) looping between solving the LP relaxation, adding a cutting plane, and re-solving the LP relaxation, until the LP relaxation yields an integer solution. By themselves, these approaches are typically intractable and may suffer from numerical instability problems. However, the most effective implementations often use a combination of valid inequalities added *a priori* to the model, after which branch-and-bound is executed, with cutting planes periodically added to the nodes of the branch-and-bound tree. This approach is called "branch-and-cut."

Valid inequality and cutting-plane approaches can either be automatic or problem-specific. Classical automatic approaches are described by Nemhauser and Wolsey [11]. More relevant to the material in this chapter are problem-specific valid inequality generation techniques. For instance, in Example 2 in the previous section, suppose that the demand of drugs in periods 1 and 2 is 100 units. Suppose that the minimum order quantity from each supplier is 150 units. If drugs become less expensive as time goes on, then the LP relaxation may try to place only 2/3 of a minimum order in each period, so that only 100 drugs (instead of the full complement of 150) are purchased in each period. Anticipating this class of fractional solutions, we note that at least one order must be placed in period 1 (since no orders result in unsatisfied demand). This valid inequality is stated as

$$\sum_{i \in M} z_{i1} \geq 1. \tag{38}$$

The addition of such inequalities to the MIP formulation often aid its performance, although occasionally they make little difference, or even worsen the performance of the branch-and-bound solver. Negative impacts usually occur when the inclusion of extra valid inequalities makes the formulation larger without sufficiently reducing its feasible region. Determining which valid inequalities are computationally beneficial is usually a matter of trial-and-error.

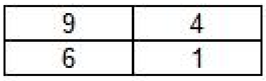
4 Example Radiation Therapy Application

In this section we describe an application of mixed-integer programming in Intensity Modulated Radiation Therapy (IMRT) planning. The underlying mechanism of radiotherapy is to radiate tumor tissues with high energy beams, which kill cancer cells. However, high energy radiation also kills healthy tissues it passes through, possibly resulting in serious degradation in the patient's quality of life. IMRT is a technique designed to help solve this dilemma. IMRT delivers small amounts of doses from multiple beam angles, which intersect at tumor tissues. As a result, the tumor tissue receives enough radiation to kill the cancer cells, but the healthy tissues are spared. The IMRT planning problem is usually solved in three interdependent phases.

- Beam Angle Optimization (BAO): Selection of the beam angles to use
- Fluence Map Optimization (FMO): Determination of intensity profile to deliver from each beam angle
- Leaf Sequencing: Realization of the intensity profiles under the capabilities of available machinery

Planning problems in IMRT have been investigated by several researchers [2, 3, 12]. The BAO problem can be formulated as an MIP model [9, 10]. The FMO problem can be formulated as a large scale LP model [14] or a nonlinear programming model [13]. The Leaf Sequencing problem can be formulated as an MIP model as we describe in this section. However, since the problem size of real-world IMRT instances is very large and these problems are inherently complex, heuristic procedures are typically used to solve real instances in a reasonable amount of time [5]. The reader is referred to [7] for a recent book chapter about mixed integer programming applications in IMRT. In this section, we will focus on the Leaf Sequencing problem and derive an MIP formulation for solving it optimally.

Leaf Sequencing Problem We are given an $m \times n$ matrix B that consists of integers called "beamlets." Matrix B represents an intensity profile that needs to be delivered from a given beam angle. We need to find an optimal way of decomposing this matrix into a set of uniform-intensity shapes, in this case rectangles, that the available machinery can deliver.

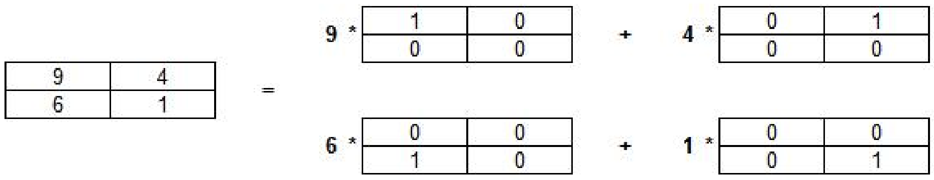


9	4
6	1

Figure 4: Fluence map example

Figure 4 represents a small example fluence map. One way to decompose this map into deliverable shapes is given in Figure 5. Define *beam-on time* as the amount of time required to deliver the doses prescribed by a set of deliverable shapes. In this problem, we measure the beam-on time as the sum of doses. The solution represented in Figure 5 results in four shapes and a total beam-on time of $9+4+6+1 = 20$ time units. Assuming that the machine requires 15 time units to switch from one shape to another, where time units are relative to one unit of beam-on time, the total time required by the configuration represented in Figure 5 is $15 \times 4 + 20 = 80$ time units.

An alternative decomposition is given in Figure 6, resulting in only three shapes and a total beam-on time of only 9 time units. The total time for this decomposition is only $15 \times 3 + 9 = 54$. This simple example illustrates that the total time required to deliver an intensity profile for a given beam varies significantly depending on the leaf sequence



$$\begin{array}{|c|c|}\hline 9 & 4 \\ \hline 6 & 1 \\ \hline\end{array} = 9 * \begin{array}{|c|c|}\hline 1 & 0 \\ \hline 0 & 0 \\ \hline\end{array} + 4 * \begin{array}{|c|c|}\hline 0 & 1 \\ \hline 0 & 0 \\ \hline\end{array} + 6 * \begin{array}{|c|c|}\hline 0 & 0 \\ \hline 1 & 0 \\ \hline\end{array} + 1 * \begin{array}{|c|c|}\hline 0 & 0 \\ \hline 0 & 1 \\ \hline\end{array}$$

Figure 5: Leaf sequence 1

$$\begin{array}{|c|c|}\hline 9 & 4 \\ \hline 6 & 1 \\ \hline \end{array} = 1 * \begin{array}{|c|c|}\hline 1 & 1 \\ \hline 1 & 1 \\ \hline \end{array} + 5 * \begin{array}{|c|c|}\hline 1 & 0 \\ \hline 1 & 0 \\ \hline \end{array} + 3 * \begin{array}{|c|c|}\hline 1 & 1 \\ \hline 0 & 0 \\ \hline \end{array}$$

Figure 6: Leaf sequence 2

employed. Since multiple beam angles are used in IMRT, the total time a patient has to spend receiving treatment, and thus the total exposure to dangerous unintentional radiation during treatment, can be reduced significantly by finding an optimal leaf sequence for each beam.

Next, we develop an MIP model to solve a special version of the leaf sequencing problem in which all shapes used must be rectangular. Let K be the set of all rectangular shapes that can be used. Define real variables x_k that denote the amount of time the rectangle $k \in K$ is in use. Let the parameter $I_{ijk} = 1$ if the rectangle $k \in K$ covers the bixel located at coordinates (i, j) and 0 otherwise. Since the required dose to each bixel must be delivered exactly, we have the following constraint:

$$\sum_{k \in K} I_{ijk} x_k = b_{ij} \quad \forall i = 1, \ldots, m, \; j = 1, \ldots, n. \tag{39}$$

Since a component of the total time is proportional to the number of rectangles used, we need to define binary variables y_k that denote whether the rectangle $k \in K$ is in use. By definition, **if** $x_k > 0$ **then** $y_k = 1$. Using the modelling tricks introduced in Section 2.2, we enforce the following constraint:

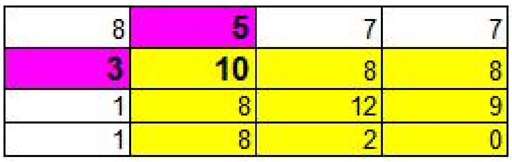
$$x_k \leq M_k y_k \quad \forall k \in K, \tag{40}$$

where M_k is a sufficiently large number. Since we want M_k to be as small as possible without imposing an artificial restriction on x_k, a good choice for M_k is the minimum value covered by the rectangle $k \in K$. In mathematical terms this can be written as $M_k = \min_{ij}\{b_{ij} | I_{ijk} = 1\}$.

Now we can write the objective function as minimization of total treatment time. Assuming that switching from one rectangle to another requires a multiple of A times a unit of beam-on time, the total treatment time is the sum of "setup time" for the rectangles and beam-on time. The overall model is then given by:

$$\begin{aligned} \text{Minimize} \quad & A \sum_{k \in K} y_k + \sum_{k \in K} x_k & \text{(41a)} \\ \text{subject to} \quad & \text{Constraints } (39) - (40) & \\ & x_k \geq 0 \text{ and } y_k \text{ binary} \quad \forall k \in K. & \text{(41b)} \end{aligned}$$

The model given above can be improved by adding valid inequalities. An idea in deriving valid inequalities is to examine each bixel $(i^\star, j^\star)$, $1 \leq i^\star \leq m$, $1 \leq j^\star \leq n$, and determine whether or not a rectangle whose upper-left-hand corner is located at $(i^\star, j^\star)$ *must* exist in any optimal solution. We say that such a rectangle starts at that bixel. The main observation is that if the required intensity of a bixel is strictly greater than that of its neighbor to the left (i.e., if $b_{i^\star j^\star} > b_{i^\star,(j^\star-1)}$), then any rectangle that starts to the left of $(i^\star, j^\star)$ cannot



8	5	7	7
3	10	8	8
1	8	12	9
1	8	2	0

Figure 7: Start index example

deliver enough dose to $(i^\star, j^\star)$ itself. (We treat $b_{i0} = 0$ for all i.) The reason is that any rectangle that starts to the left of $(i^\star, j^\star)$ and covers it also covers the left-neighbor bixel. Since the left-neighbor bixel cannot be overdosed and since its required intensity is strictly less than that of $(i^\star, j^\star)$, the intensity of any such rectangle cannot be large enough to cover $(i^\star, j^\star)$ by itself. Therefore a rectangle that starts in the same column as $(i^\star, j^\star)$ and covers this bixel must exist in the solution to (41).

A slight extension of the same idea is based on comparing the intensity requirement of $(i^\star, j^\star)$ with the intensity requirements of the adjacent bixels on the left and above it. If the intensity requirement of $(i^\star, j^\star)$ is strictly greater than sum of the intensity requirements of these two neighbors, then a rectangle must start at $(i^\star, j^\star)$. More formally:

$$b_{i^\star j^\star} > b_{(i^\star-1)j^\star} + b_{i^\star(j^\star-1)} \Rightarrow \text{ a rectangle must start at } (i^\star, j^\star), \tag{42}$$

where $b_{0j} = 0$, $1 \leq j \leq n$.

Figure 7 illustrates this idea. Since $b_{2,2} > b_{1,2} + b_{2,1}$ we can conclude that a rectangle must start at $(2, 2)$. Note that a special case that is always satisfied by (42) occurs at the upper-left bixel (1,1), assuming $b_{11} > 0$.

We can preprocess the data before formulating the MIP model and determine all coordinates satisfying (42). Let $\mathcal{S}$ be the set of all such coordinates, and define ST_{ij} be the set of rectangles that start at (i, j), for all $(i, j) \in \mathcal{S}$. Then we can add the following valid inequalities to model (41):

$$\sum_{k \in ST_{ij}} y_k \geq 1 \quad \forall (i, j) \in \mathcal{S}. \tag{43}$$

5 Conclusion

In this chapter, we have discussed basic formulation and modelling principles of mixed-integer programming, fundamental MIP solution algorithms, and the use of MIPs for solving several types of problems. The ability to model different types of conditions such as integrality requirements, logical expressions (e.g., "if-then" and "either-or" relationships), and certain nonlinear expressions (e.g., minimum, maximum, absolute value, and some product terms), makes mixed-integer programming a very flexible technique for solving optimization problems. Several commercial and open source software systems for MIP are readily available. Most MIP solvers can be used as a "black box," allowing the user to focus on modelling instead of solution algorithm development. However, experienced users can also interact with the solver using general-purpose programming languages such as C, C++, Java, C#, and Visual Basic. This flexibility allows users to guide the solution algorithm in order to exploit special structures of the problem on hand, resulting in more efficient solver performance.

Even though MIP models are designed to find a provably optimal solution, it is possible to stop execution once a "good enough" solution is found. In other words, it is possible to use MIP-based algorithms as heuristics in computationally difficult problems. However, there is an important distinction between problem specific heuristics and MIP-based algorithms: unlike the former, MIP-based algorithms are capable of measuring the quality of the solution found with respect to the (unknown) optimal solution. These features make mixed-integer programming a suitable technique for solving difficult optimization problems, including those in healthcare applications.

References

[1] M. S. Bazaraa, J. J. Jarvis, and H. D. Sherali. *Linear Programming and Network Flows.* John Wiley & Sons, New York, second edition, 1990.

[2] G. Bednarz, D. Michalski, C. Houser, M. S. Huq, Y. Xiao, P. R. Anne, and J. M. Galvin. The use of mixed-integer programming for inverse treatment planning with pre-defined field segments. *Physics in Medicine and Biology*, 47(13):2235–2245, 2002.

[3] J. Deasy, E. K. Lee, T. Bortfeld, M. Langer, K. Zakarian, J. Alaly, Y. Zhang, H. Liu, R. Mohan, R. Ahuja, A. Pollack, J. Purdy, and R. Rardin. A collaboratory for radiation therapy treatment planning optimization research. *Annals of Operations Research*, 148(1):55–63, 2006.

[4] M. Elshafei, H. D. Sherali, and J. C. Smith. Radar pulse interleaving for multi-target tracking. *Naval Research Logistics*, 51(4):72–94, 2004.

[5] K. Engel. A new algorithm for optimal multileaf collimator field segmentation. *Discrete Applied Mathematics*, 152:35–51, 2005.

[6] A. Farina and P. Neri. Multitarget interleaved tracking for phased-array radar. *IEEE Proceedings, Part F: Communications, Radar & Signal Processing*, 127(4):312–318, 1980.

[7] M. Ferris, R. Meyer, and W. D'Souza. Radiation treatment planning: Mixed integer programming formulations and approaches. In G. Appa, L. Pitsoulis, and H. P. Williams, editors, *Handbook on Modelling for Discrete Optimization*, volume 88, pages 317–340. Springer, New York, NY, 2006.

[8] F. S. Hillier and G. J. Lieberman. *Introduction to Operations Research.* McGraw-Hill, New York, NY, eighth edition, 2005.

[9] E. K. Lee, T. Fox, and I. Crocker. Integer programming applied to intensity-modulated radiation therapy treatment planning. *Annals of Operations Research*, 119:165–181, 2003.

[10] G. L. Lim, J. Choi, and R. Mohan. Iterative solution methods for beam angle and fluence map optimization in intensity modulated radiation therapy planning. Technical Report 10-01, Department of Industrial Engineering, University of Houston, Houston, TX, 2006.

[11] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization.* John Wiley & Sons, New York, NY, 1999.

[12] F. Preciado-Walters, R. Rardin, M. Langer, and V. Thai. A coupled column generation, mixed integer approach to optimal planning of intensity modulated radiation therapy for cancer. *Mathematical Programming*, 101(2):319–338, 2004.

[13] H. E. Romeijn, R. K. Ahuja, J. F. Dempsey, and A. Kumar. A column generation approach to radiation therapy treatment planning using aperture modulation. *SIAM Journal on Optimization*, 15(3):838–862, 2005.

[14] H. E. Romeijn, R. K. Ahuja, J. F. Dempsey, A. Kumar, and J. G. Li. A novel linear programming approach to fluence map optimization for intensity modulated radiation therapy treatment planning. *Physics in Medicine and Biology*, 48(21):3521–3542, 2003.

[15] A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley & Sons, New York, NY, 1986.

[16] H. D. Sherali and J. C. Smith. Interleaving two-phased jobs on a single machine with application to radar pulse interleaving. *Discrete Optimization*, 2(4):348–361, 2005.

[17] W. L. Winston and M. Venkataramanan. *Introduction to Mathematical Programming: Applications and Algorithms, Volume 1.* Duxbury Press, Belmont, CA, fourth edition, 2002.

[18] L. A. Wolsey. *Integer Programming.* John Wiley & Sons, New York, NY, 1998.