AIMMS Modeling Guide - Integer Programming Tricks

This file contains only one chapter of the book. For a free download of the complete book in pdf format, please visit www.aimms.com.

AIMMS B.V.
Diakenhuisweg 29-35
2033 AP Haarlem
The Netherlands
Tel.: +31 23 5511512

AIMMS Inc.
11711 SE 8th Street
Suite 303
Bellevue, WA 98005
USA
Tel.: +1 425 458 4024

AIMMS Pte. Ltd.
55 Market Street #10-00
Singapore 048941
Tel.: +65 6521 2827

AIMMS
SOHO Fuxing Plaza No.388
Building D-71, Level 3
Madang Road, Huangpu District
Shanghai 200025
China
Tel.: ++86 21 5309 8733

Email: info@aimms.com
WWW: www.aimms.com

Chapter 7

Integer Linear Programming Tricks

This chapter

As in the previous chapter "Linear Programming Tricks", the emphasis is on abstract mathematical modeling techniques but this time the focus is on *integer* programming tricks. These are not discussed in any particular reference, but are scattered throughout the literature. Several tricks can be found in [Wi90]. Other tricks are referenced directly.

Limitation to linear integer programs

Only *linear* integer programming models are considered because of the availability of computer codes for this class of problems. It is interesting to note that several practical problems can be transformed into linear integer programs. For example, integer variables can be introduced so that a nonlinear function can be approximated by a "piecewise linear" function. This and other examples are explained in this chapter.

7.1 A variable taking discontinuous values

A jump in the bound

This section considers an example of a simple situation that cannot be formulated as a linear programming model. The value of a variable must be either zero or between particular positive bounds (see Figure 7.1). In algebraic notation:

$$x = 0 \quad \text{or} \quad l \le x \le u$$

This can be interpreted as two constraints that cannot both hold simultaneously. In linear programming only simultaneous constraints can be modeled.

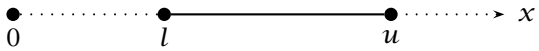

Figure 7.1: A discontinuous variable

Applications

This situation occurs when a supplier of some item requires that if an item is ordered, then its batch size must be between a particular minimum and maximum value. Another possibility is that there is a set-up cost associated with the manufacture of an item.

Modeling discontinuous variables

To model discontinuous variables, it is helpful to introduce the concept of an *indicator variable*. An indicator variable is a binary variable (0 or 1) that indicates a certain state in a model. In the above example, the indicator variable y is linked to x in the following way:

$$y = \begin{cases} 0 & \text{for } x = 0 \\ 1 & \text{for } l \le x \le u \end{cases}$$

The following set of constraints is used to create the desired properties:

$$\begin{aligned} x &\le uy \\ x &\ge ly \\ y &\quad \text{binary} \end{aligned}$$

It is clear that $y = 0$ implies $x = 0$, and that $y = 1$ implies $l \le x \le u$.

7.2 Fixed costs

The model

A fixed cost problem is another application where indicator variables are added so that two mutually exclusive situations can be modeled. An example is provided using a single-variable. Consider the following linear programming model (the sign "$\gtrless$" denotes either "$\le$", "$=$", or "$\ge$" constraints).

Minimize: $C(x)$

Subject to:

$$\begin{aligned} a_i x + \sum_{j \in J} a_{ij} w_j &\gtrless b_i \qquad \forall i \in I \\ x &\ge 0 \\ w_j &\ge 0 \qquad \forall j \in J \end{aligned}$$

Where: $$C(x) = \begin{cases} 0 & \text{for } x = 0 \\ k + cx & \text{for } x > 0 \end{cases}$$

As soon as x has a positive value, a fixed cost is incurred. This cost function is not linear and is not continuous. There is a jump at $x = 0$, as illustrated in Figure 7.2.

Application

In the above formulation, the discontinuous function is the objective, but such a function might equally well occur in a constraint. An example of such a fixed-cost problem occurs in the manufacturing industry when set-up costs are charged for new machinery.

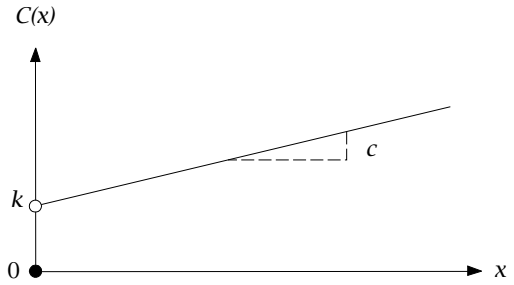

Figure 7.2: Discontinuous cost function

Modeling fixed costs

A sufficiently large upper bound, u, must be specified for x. An indicator variable, y, is also introduced in a similar fashion:

$$y = \begin{cases} 0 & \text{for } x = 0 \\ 1 & \text{for } x > 0 \end{cases}$$

Now the cost function can be specified in both x and y:

$$C^*(x, y) = ky + cx$$

The minimum of this function reflects the same cost figures as the original cost function, except for the case when $x > 0$ and $y = 0$. Therefore, one constraint must be added to ensure that $x = 0$ whenever $y = 0$:

$$x \le uy$$

The equivalent mixed integer program

Now the model can be stated as a mixed integer programming model. The formulation given earlier in this section can be transformed as follows.

Minimize: $ky + cx$

Subject to:

$$a_i x + \sum_{j \in J} a_{ij} w_j \gtrless b_i \qquad \forall i \in I$$

$$x \le uy$$

$$x \ge 0$$

$$w_j \ge 0 \qquad \forall j \in J$$

$$y \quad \text{binary}$$

7.3 Either-or constraints

The model

Consider the following linear programming model:

Minimize: $\sum_{j \in J} c_j x_j$

Subject to:

$$\sum_{j \in J} a_{1j} x_j \leq b_1 \qquad (1)$$

$$\sum_{j \in J} a_{2j} x_j \leq b_2 \qquad (2)$$

$$x_j \geq 0 \qquad \forall j \in J$$

Where: *at least one of the conditions (1) or (2) must hold*

The condition that at least one of the constraints must hold cannot be formulated in a linear programming model, because in a linear program *all* constraints must hold. Again, a binary variable can be used to express the problem. An example of such a situation is a manufacturing process, where two modes of operation are possible.

Modeling either-or constraints

Consider a binary variable y, and sufficiently large upper bounds M_1 and M_2, which are upper bounds on the activity of the constraints. The bounds are chosen such that they are as tight as possible, while still guaranteeing that the left-hand side of constraint i is always smaller than $b_i + M_i$. The constraints can be rewritten as follows:

$$(1) \quad \sum_{j \in J} a_{1j} x_j \leq b_1 + M_1 y$$

$$(2) \quad \sum_{j \in J} a_{2j} x_j \leq b_2 + M_2 (1 - y)$$

When $y = 0$, constraint (1) is imposed, and constraint (2) is weakened to $\sum_{j \in J} a_{2j} x_j \leq b_2 + M_2$, which will always be non-binding. Constraint (2) may of course still be satisfied. When $y = 1$, the situation is reversed. So in all cases one of the constraints is imposed, and the other constraint may also hold. The problem then becomes:

The equivalent mixed integer program

Minimize:

$$\sum_{j \in J} c_j x_j$$

Subject to:

$$\sum_{j \in J} a_{1j} x_j \leq b_1 + M_1 y$$

$$\sum_{j \in J} a_{2j} x_j \leq b_2 + M_2 (1 - y)$$

$$x_j \geq 0 \qquad \forall j \in J$$

$$y \quad \text{binary}$$

7.4 Conditional constraints

The model

A problem that can be treated in a similar way to either-or constraints is one that contains conditional constraints. The mathematical presentation is limited to a case, involving two constraints, on which the following condition is imposed.

$$\begin{aligned} &\textit{If} \quad \textit{(1)} \quad (\sum_{j\in J} a_{1j}x_j \le b_1) \quad \textit{is satisfied,} \\ &\textit{then} \quad \textit{(2)} \quad (\sum_{j\in J} a_{2j}x_j \le b_2) \quad \textit{must also be satisfied.} \end{aligned}$$

Logical equivalence

Let A denote the statement that the logical expression "Constraint (1) holds" is true, and similarly, let B denote the statement that the logical expression "Constraint (2) holds" is true. The notation $\neg A$ and $\neg B$ represent the case of the corresponding logical expressions being false. The above conditional constraint can be restated as: *A implies B*. This is logically equivalent to writing *(A and $\neg B$) is false*. Using the negation of this expression, it follows that *$\neg$(A and $\neg B$)* is true. This is equivalent to *($\neg A$ or B) is true*, using Boolean algebra. It is this last equivalence that allows one to translate the above conditional constraint into an either-or constraint.

Modeling conditional constraints

One can observe that

$$\textit{If} \quad (\sum_{j\in J} a_{1j}x_j \le b_1) \quad \textit{holds, then} \quad (\sum_{j\in J} a_{2j}x_j \le b_2) \quad \textit{must hold,}$$

is equivalent to

$$(\sum_{j\in J} a_{1j}x_j > b_1) \quad \textit{or} \quad (\sum_{j\in J} a_{2j}x_j \le b_2) \quad \textit{must hold.}$$

Notice that the sign in (1) is reversed. A difficulty to overcome is that the strict inequality "not (1)" needs to be modeled as an inequality. This can be achieved by specifying a small tolerance value beyond which the constraint is regarded as broken, and rewriting the constraint to:

$$\sum_{j\in J} a_{1j}x_j \ge b_1 + \epsilon$$

This results in:

$$\sum_{j\in J} a_{1j}x_j \ge b_1 + \epsilon, \quad \textit{or} \quad \sum_{j\in J} a_{2j}x_j \le b_2 \quad \textit{must hold.}$$

This last expression strongly resembles the either-or constraints in the previous section. This can be modeled in a similar way by introducing a binary

variable y, a sufficiently large upper bound M on (2), and a sufficiently lower bound L on (1). The constraints can be rewritten to get:

$$\sum_{j \in J} a_{1j} x_j \geq b_1 + \epsilon - Ly$$
$$\sum_{j \in J} a_{2j} x_j \leq b_2 + M(1 - y)$$

You can verify that these constraints satisfy the original conditional expression correctly, by applying reasoning similar to that in Section 7.3.

7.5 Special Ordered Sets

This section

There are particular types of restrictions in integer programming formulations that are quite common, and that can be treated in an efficient manner by solvers. Two of them are treated in this section, and are referred to as *Special Ordered Sets* (SOS) of type 1 and 2. These concepts are due to Beale and Tomlin ([Be69]).

SOS1 constraints

A common restriction is that out of a set of yes-no decisions, at most one decision variable can be yes. You can model this as follows. Let y_i denote zero-one variables, then

$$\sum_i y_i \leq 1$$

forms an example of a SOS1 constraint. More generally, when considering variables $0 \leq x_i \leq u_i$, then the constraint

$$\sum_i a_i x_i \leq b$$

can also become a SOS1 constraint by adding the requirement that at most one of the x_i can be nonzero. In AIMMS there is a constraint attribute named `Property` in which you can indicate whether this constraint is a SOS1 constraint. Note that in the general case, the variables are no longer restricted to be zero-one variables.

SOS1 and performance

A general SOS1 constraint can be classified as a logical constraint and as such it can always be translated into a formulation with binary variables. Under these conditions the underlying branch and bound process will follow the standard binary tree search, in which the number of nodes is an exponential function of the number of binary variables. Alternatively, if the solver recognizes it as a SOS1 constraint, then the number of nodes to be searched can be reduced. However, you are advised to only use SOS sets if there exists an natural order relationship among the variables in the set. If your model contains multiple SOS sets, you could consider specifying priorities for some of these SOS sets.

SOS1 branching

To illustrate how the SOS order information is used to create new nodes during the branch and bound process, consider a model in which a decision has to be made about the size of a warehouse. The size of the warehouse should be either 10000, 20000, 40000, or 50000 square feet. To model this, four binary variables x_1, x_2, x_3 and x_4 are introduced that together make up a SOS1 set. The order among these variables is naturally specified through the sizes. During the branch and bound process, the split point in the SOS1 set is determined by the weighted average of the solution of the relaxed problem. For example, if the solution of the relaxed problem is given by $x_1 = 0.1$ and $x_4 = 0.9$, then the corresponding weighted average is $0.1 \cdot 10000 + 0.9 \cdot 50000 = 46000$. This computation results in the SOS set being split up between variable x_3 and x_4. The corresponding new nodes in the search tree are specified by (1) the nonzero element is the set $\{x_1, x_2, x_3\}$ (i.e. $x_4 = 0$) and (2) $x_4 = 1$ (and $x_1 = x_2 = x_3 = 0$).

SOS2 constraints

Another common restriction, is that out of a set of nonnegative variables, at most two variables can be nonzero. In addition, the two variables must be adjacent to each other in a fixed order list. This class of constraint is referred to as a type SOS2 in AIMMS. A typical application occurs when a non-linear function is approximated by a piecewise linear function. Such an example is given in the next section.

7.6 Piecewise linear formulations

The model

Consider the following model with a separable objective function:

Minimize:
$$\sum_{j \in J} f_j(x_j)$$

Subject to:
$$\sum_{j \in J} a_{ij} x_j \gtrless b_i \qquad \forall i \in I$$
$$x_j \geq 0 \qquad \forall j \in J$$

Separable function

In the above general model statement, the objective is a *separable function*, which is defined as the sum of functions of scalar variables. Such a function has the advantage that nonlinear terms can be approximated by piecewise linear ones. Using this technique, it may be possible to generate an integer programming model, or sometimes even a linear programming model (see [Wi90]). This possibility also exists when a constraint is separable.

Examples of separable functions

Some examples of separable functions are:

$$x_1^2 + 1/x_2 - 2x_3 = f_1(x_1) + f_2(x_2) + f_3(x_3)$$
$$x_1^2 + 5x_1 - x_2 = g_1(x_1) + g_2(x_2)$$

The following examples are not:

$$x_1x_2 + 3x_2 + x_2^2 = f_1(x_1, x_2) + f_2(x_2)$$
$$1/(x_1 + x_2) + x_3 = g_1(x_1, x_2) + g_2(x_3)$$

Approximation of a nonlinear function

Consider a simple example with only one nonlinear term to be approximated, namely $f(x) = \frac{1}{2}x^2$. Figure 7.3, shows the curve divided into three pieces that are approximated by straight lines. This approximation is known as piecewise linear. The points where the slope of the piecewise linear function changes (or its domain ends) are referred to as breakpoints. This approximation can be expressed mathematically in several ways. A method known as the λ-formulation is described below.

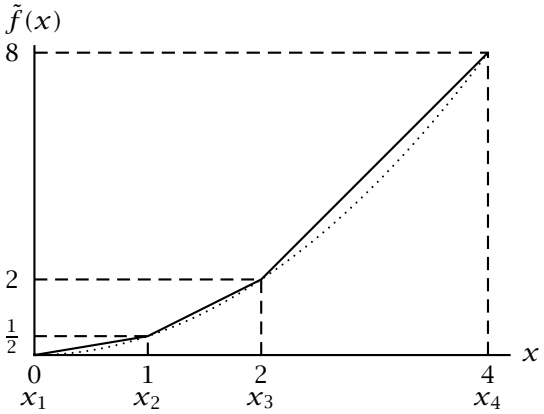

Figure 7.3: Piecewise linear approximation of $f(x) = \frac{1}{2}x^2$

Weighted sums

Let x_1, x_2, x_3 and x_4 denote the four breakpoints along the x-axis in Figure 7.3, and let $f(x_1), f(x_2), f(x_3)$ and $f(x_4)$ denote the corresponding function values. The breakpoints are 0, 1, 2 and 4, and the corresponding function values are 0, $\frac{1}{2}$, 2 and 8. Any point in between two breakpoints is a weighted sum of these two breakpoints. For instance, $x = 3 = \frac{1}{2} \cdot 2 + \frac{1}{2} \cdot 4$. The corresponding approximated function value $\tilde{f}(3) = 5 = \frac{1}{2} \cdot 2 + \frac{1}{2} \cdot 8$.

λ-Formulation

Let $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ denote four nonnegative weights such that their sum is 1. Then the piecewise linear approximation of $f(x)$ in Figure 7.3 can be written as:

$$\lambda_1 f(x_1) + \lambda_2 f(x_2) + \lambda_3 f(x_3) + \lambda_4 f(x_4) = \tilde{f}(x)$$
$$\lambda_1 x_1 + \lambda_2 x_2 + \lambda_3 x_3 + \lambda_4 x_4 = x$$
$$\lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 = 1$$

with the added requirement that at most two adjacent λ's are greater than zero. This requirement together with the last constraint form the SOS2 con-

straint referred to at the end of the previous section. The SOS2 constraints for all separable functions in the objective function together guarantee that the points $(x, \tilde{f}(x))$ always lie on the approximating line segments.

Adjacency requirements sometimes redundant

The added requirement that at most two adjacent λ's are greater than zero can be modeled using additional binary variables. This places any model with SOS2 constraints in the realm of integer (binary) programming. For this reason, it is worthwhile to investigate when the added adjacency requirements are redundant. Redundancy is implied by the following conditions.

1. The objective is to *minimize* a separable function in which all terms $f_j(x_j)$ are *convex* functions.
2. The objective is to *maximize* a separable function in which all terms $f_j(x_j)$ are *concave* functions.

Convexity and concavity

A function is convex when the successive slopes of the piecewise linear approximation are nondecreasing, and concave if these slopes are non-increasing. A concave cost curve can represent an activity with economies of scale. The unit costs decrease as the number of units increases. An example is where quantity discounts are obtained.

The case of a non-convex function

The adjacency requirements are no longer redundant when the function to be approximated is non-convex. In this case, these adjacency requirements must be formulated explicitly in mathematical terms.

SOS2 in AIMMS

In AIMMS you do not need to formulate the adjacency requirements explicitly. Instead you need to specify `sos2` in the `property` attribute of the constraint in which the λ's are summed to 1. In this case, the solver in AIMMS guarantees that there will be at most two adjacent nonzero λ's in the optimal solution. If the underlying minimization model is convex, then the linear programming solution will satisfy the adjacency requirements. If the model is not convex, the solver will continue with a mixed integer programming run.

7.7 Elimination of products of variables

This section

This section explains a method for linearizing constraints and objective functions in which the products of variables are incorporated. There are numerous applications that give rise to nonlinear constraints and the use of integer variables. These problems become very difficult, if not impossible, to solve.

Replacing product term

In general, a product of two variables can be replaced by one new variable, on which a number of constraints is imposed. The extension to products of more than two variables is straightforward. Three cases are distinguished. In the third case, a separable function results (instead of a linear one) that can then be approximated by using the methods described in the previous section.

Two binary variables

Firstly, consider the binary variables x_1 and x_2. Their product, x_1x_2, can be replaced by an additional binary variable y. The following constraints force y to take the value of x_1x_2:

$$\begin{aligned} & y \le x_1 \\ & y \le x_2 \\ & y \ge x_1 + x_2 - 1 \\ & y \quad \text{binary} \end{aligned}$$

One binary and one continuous variable

Secondly, let x_1 be a binary variable, and x_2 be a continuous variable for which $0 \le x_2 \le u$ holds. Now a continuous variable, y, is introduced to replace the product $y = x_1x_2$. The following constraints must be added to force y to take the value of x_1x_2:

$$\begin{aligned} & y \le ux_1 \\ & y \le x_2 \\ & y \ge x_2 - u(1 - x_1) \\ & y \ge 0 \end{aligned}$$

The validity of these constraints can be checked by examining Table 7.1 in which all possible situations are listed.

x_1	x_2	x_1x_2	**constraints**	**imply**
0	$w : 0 \le w \le u$	0	$y \le 0$ $y \le w$ $y \ge w - u$ $y \ge 0$	$y = 0$
1	$w : 0 \le w \le u$	w	$y \le u$ $y \le w$ $y \ge w$ $y \ge 0$	$y = w$

Table 7.1: All possible products $y = x_1x_2$

Two continuous variables

Thirdly, the product of two continuous variables can be converted into a separable form. Suppose the product x_1x_2 must be transformed. First, two (continuous) variables y_1 and y_2 are introduced. These are defined as:

$$y_1 = \frac{1}{2}(x_1 + x_2)$$
$$y_2 = \frac{1}{2}(x_1 - x_2)$$

Now the term x_1x_2 can be replaced by the separable function

$$y_1^2 - y_2^2$$

which can be approximated by using the technique of the preceding section. Note that in this case the non-linear term can be eliminated at the cost of having to approximate the objective. If $l_1 \le x_1 \le u_1$ and $l_2 \le x_2 \le u_2$, then the bounds on y_1 and y_2 are:

$$\frac{1}{2}(l_1 + l_2) \le y_1 \le \frac{1}{2}(u_1 + u_2) \quad \text{and} \quad \frac{1}{2}(l_1 - u_2) \le y_2 \le \frac{1}{2}(u_1 - l_2)$$

Special case

The product x_1x_2 can be replaced by a single variable z whenever

- the lower bounds l_1 and l_2 are nonnegative, and
- one of the variables is not referenced in any other term except in products of the above form.

Assume, that x_1 is such a variable. Then substituting for z and adding the constraint

$$l_1x_2 \le z \le u_1x_2$$

is all that is required to eliminate the nonlinear term x_1x_2. Once the model is solved in terms of z and x_2, then $x_1 = z/x_2$ when $x_2 > 0$ an x_1 is undetermined when $x_2 = 0$. The extra constraints on z guarantee that $l_1 \le x_1 \le u_1$ whenever $x_2 > 0$.

7.8 Summary

In practical applications, integer linear programming models often arise when discontinuous restrictions are added to linear programs. In this chapter, some typical examples have been shown, along with methods to reformulate them as integer programs. The binary "indicator variable" plays an important role. With the aid of binary variables it is possible to model discontinuities in variables or objectives, as well as either-or constraints and conditional constraints. By conducting a piecewise linear approximation of a nonlinear program, containing a separable nonlinear objective function, it may be possible to generate a linear programming model or perhaps an integer programming model. At the end of the chapter, methods for eliminating products of variables are described.

Bibliography

[Be69] E.M.L. Beale and J.A. Tomlin, *Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables*, Proceedings of the 5th International Conference on Operations Research (Tavistock, London) (J. Lawrence, ed.), 1969.

[Wi90] H.P. Williams, *Model building in mathematical programming*, 3rd ed., John Wiley & Sons, Chichester, 1990.