INTEGER PROGRAMMING

LAURENCE A. WOLSEY

INTEGER PROGRAMMING

A practical, accessible guide to optimization problems with discrete or integer variables

Integer Programming stands out from other textbooks by explaining in clear and simple terms how to construct custom-made algorithms or use existing commercial software to obtain optimal or near-optimal solutions for a variety of real-world problems, such as airline timetables, production line schedules, or electricity production on a regional or national scale.

Incorporating recent developments that have made it possible to solve difficult optimization problems with greater accuracy, author Laurence A. Wolsey presents a number of state-of-the-art topics not covered in any other textbook. These include improved modeling, cutting plane theory and algorithms, heuristic methods, and branch-and-cut and integer programming decomposition algorithms. This self-contained text:

- Distinguishes between good and bad formulations in integer programming problems
- Applies lessons learned from easy integer programs to more difficult problems
- Demonstrates with applications theoretical and practical aspects of problem solving
- Includes useful notes and end-of-chapter exercises
- Offers tremendous flexibility for tailoring material to different needs

Integer Programming is an ideal text for courses in integer/mathematical programming—whether in operations research, mathematics, engineering, or computer science departments. It is also a valuable reference for industrial users of integer programming and researchers who would like to keep up with advances in the field.

LAURENCE A. WOLSEY is Professor of Applied Mathematics at the Center for Operations Research and Econometrics (CORE) at l'Université Catholique de Louvain at Louvain-la-Neuve, Belgium. He is the author, with George Nemhauser, of *Integer and Combinatorial Optimization* (Wiley).

INTEGER PROGRAMMING

ISBN 0-471-28366-5
90000
9 780471 283669

Integer Programming

LAURENCE A. WOLSEY

A Wiley-Interscience Publication
JOHN WILEY & SONS, INC.
New York • Chichester • Weinheim • Brisbane • Singapore • Toronto

Published simultaneously in Canada.

Library of Congress Cataloging-in-Publication Data:

Wolsey, Laurence A.
Integer programming / Laurence A. Wolsey
p. cm. — (Wiley-Interscience series in discrete mathematics and optimization)
"Wiley-Interscience publication."
Includes bibliographical references and indexes.
ISBN 0-471-28366-5 (alk. paper)
1. Integer programming. I. Title. II. Series.
T57.74W67 1998
519.7'7—dc21 98-7296

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

Example 7.3 In Figure 7.5 we again decompose S into two sets S_1 and S_2 with different upper and lower bounds.

Fig. 7.5 No pruning possible

We note first that $\overline{z} = \max_k \overline{z}^k = \max\{24, 37\} = 37$ and $\underline{z} = \max_k \underline{z}^k = \max\{13, -\} = 13$. Here no other conclusion can be drawn and we need to explore both sets S_1 and S_2 further. ∎

Based on these examples, we can list at least three reasons that allow us to prune the tree and thus enumerate a large number of solutions implicitly.

(i) Pruning by optimality: $z_t = \{\max cx : x \in S_t\}$ has been solved.

(ii) Pruning by bound: $\overline{z_t} \leq \underline{z}$.

(iii) Pruning by infeasiblity: $S_t = \phi$.

If we now ask how the bounds are to be obtained, the reply is no different from in Chapter 2. The primal (lower) bounds are provided by feasible solutions and the dual (upper) bounds by relaxation or duality.

Building an implicit enumeration algorithm based on the above ideas is now in principle a fairly straightforward task. There are, however, many questions that must be addressed before such an algorithm is well-defined. Some of the most important questions are:

What relaxation or dual problem should be used to provide upper bounds? How should one choose between a fairly weak bound that can be calculated very rapidly and a stronger bound whose calculation takes a considerable time?

How should the feasible region be separated into smaller regions $S = S_1 \cup \ldots \cup S_K$? Should one separate into two or more parts? Should one use a fixed a priori rule for dividing up the set, or should the divisions evolve as a function of the bounds and solutions obtained en route?

In what order should the subproblems be examined? Typically there is a list of active problems that have not yet been pruned. Should the next one be chosen on a the basis of last-in first-out, of best/largest upper bound first, or of some totally different criterion?

These and other questions will be discussed further once we have seen an example.

7.3 BRANCH AND BOUND: AN EXAMPLE

The most common way to solve integer programs is to use implicit enumeration, or *branch and bound*, in which linear programming relaxations provide the bounds. We first demonstrate the approach by an example:

$$z = \max 4x_1 - x_2 \tag{7.1}$$
$$7x_1 - 2x_2 \leq 14 \tag{7.2}$$
$$x_2 \leq 3 \tag{7.3}$$
$$2x_1 - 2x_2 \leq 3 \tag{7.4}$$
$$x \in Z_+^2. \tag{7.5}$$

Bounding. To obtain a first upper bound, we add slack variables x_3, x_4, x_5 and solve the linear programming relaxation in which the integrality constraints are dropped. The resulting optimal basis representation is:

$$\begin{array}{rrrrrcl}
\overline{z} = \max \frac{59}{7} & & -\frac{4}{7}x_3 & -\frac{1}{7}x_4 & & & \\
x_1 & & +\frac{1}{7}x_3 & +\frac{2}{7}x_4 & & = & \frac{20}{7} \\
& x_2 & & +x_4 & & = & 3 \\
& & -\frac{2}{7}x_3 & +\frac{10}{7}x_4 & +x_5 & = & \frac{23}{7} \\
x_1, & x_2, & x_3, & x_4, & x_5 & \geq & 0.
\end{array}$$

Thus we obtain an upper bound $\overline{z} = \frac{59}{7}$, and a nonintegral solution $(\overline{x}_1, \overline{x}_2) = (\frac{20}{7}, 3)$. Is there any straightforward way to find a feasible solution? Apparently not. By convention, as no feasible solution is yet available, we take as lower bound $\underline{z} = -\infty$.

Branching. Now because $\underline{z} < \overline{z}$, we need to divide or branch. How should we split up the feasible region? One simple idea is to choose an integer variable that is basic and fractional in the linear programming solution, and split the problem into two about this fractional value. If $x_j = \overline{x}_j \notin Z^1$, one can take:

$$S_1 = S \cap \{x : x_j \leq \lfloor \overline{x}_j \rfloor\}$$
$$S_2 = S \cap \{x : x_j \geq \lceil \overline{x}_j \rceil\}.$$

It is clear that $S = S_1 \cup S_2$ and $S_1 \cap S_2 = \phi$. Another reason for this choice is that the solution $\bar{x}$ of $LP(S)$ is not feasible in either $LP(S_1)$ or $LP(S_2)$. This implies that if there is no degeneracy (i.e., multiple optimal LP solutions), then $\max\{\overline{z}_1, \overline{z}_2\} < \overline{z}$, so the upper bound will strictly decrease.

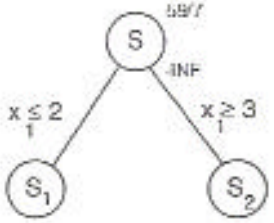

Fig. 7.6 Partial branch-and-bound tree 1

Following this idea, as $\bar{x}_1 = 20/7 \notin Z^1$, we take $S_1 = S \cap \{x : x_1 \leq 2\}$ and $S_2 = S \cap \{x : x_1 \geq 3\}$. We now have the tree shown in Figure 7.6. The subproblems (nodes) that must still be examined are called *active*.

Choosing a Node. The list of active problems (nodes) to be examined now contains S_1, S_2. We arbitrarily choose S_1.

Reoptimizing. How should we solve the new modified linear programs $LP(S_i)$ for $i = 1, 2$ without starting again from scratch?

As we have just added one single upper or lower bound constraint to the linear program, our previous optimal basis remains dual feasible, and it is therefore natural to reoptimize from this basis using the dual simplex algorithm. Typically, only a few pivots will be needed to find the new optimal linear programming solution.

Applying this to the linear program $LP(S_1)$, we can write the new constraint $x_1 \leq 2$ as $x_1 + s = 2, s \geq 0$, which can be rewritten in terms of the nonbasic variables as

$$-\tfrac{1}{7}x_3 - \tfrac{2}{7}x_4 + s = -\tfrac{6}{7}.$$

Thus we have the dual feasible representation:

$$\begin{array}{lccccccc}
\overline{z}_1 = \max \frac{59}{7} & & -\frac{4}{7}x_3 & -\frac{1}{7}x_4 & & & & \\
x_1 & & +\frac{1}{7}x_3 & +\frac{2}{7}x_4 & & & = & \frac{20}{7} \\
& x_2 & & +x_4 & & & = & 3 \\
& & -\frac{2}{7}x_3 & +\frac{10}{7}x_4 & +x_5 & & = & \frac{23}{7} \\
& & -\frac{1}{7}x_3 & -\frac{2}{7}x_4 & & +s & = & -\frac{6}{7} \\
x_1, & x_2, & x_3, & x_4, & x_5, & s & \geq & 0.
\end{array}$$

After two simplex pivots, the linear program is reoptimized, giving:

$$\begin{array}{lccccccc}
\overline{z}_1 = \max \frac{15}{2} & & & & -\frac{1}{2}x_5 & -3s & & \\
x_1 & & & & & +s & = & 2 \\
& x_2 & & & -\frac{1}{2}x_5 & +s & = & \frac{1}{2} \\
& & x_3 & & -x_5 & -5s & = & 1 \\
& & & x_4 & +\frac{1}{2}x_5 & +6s & = & \frac{5}{2} \\
x_1, & x_2, & x_3, & x_4, & x_5, & s & \geq & 0
\end{array}$$

with $\overline{z}_1 = \frac{15}{2}$, and $(\overline{x}_1^1, \overline{x}_2^1) = (2, \frac{1}{2})$.

Branching. S_1 cannot be pruned, so using the same branching rule as before, we create two new nodes $S_{11} = S_1 \cap \{x : x_2 \leq 0\}$ and $S_{12} = S_1 \cap \{x : x_2 \geq 1\}$, and add them to the node list. The tree is now as shown in Figure 7.7.

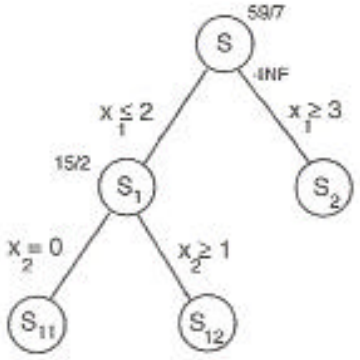

Fig. 7.7 Partial branch-and-bound tree 2

Choosing a Node. The active node list now contains S_2, S_{11}, S_{12}. Arbitrarily choosing S_2, we remove it from the node list and examine it in more detail.

Reoptimizing. To solve $LP(S_2)$, we use the dual simplex algorithm in the same way as above. The constraint $x_1 \geq 3$ is first written as $x_1 - t = 3, t \geq 0$, which expressed in terms of the nonbasic variables becomes:

$$\tfrac{1}{7}x_3 + \tfrac{2}{7}x_4 + t = -\tfrac{1}{7}.$$

From inspection of this constraint, we see that the resulting linear program

$$\begin{array}{lccccccccc}
\overline{z}_2 = \max \frac{59}{7} & & & -\frac{4}{7}x_3 & -\frac{1}{7}x_4 & & & & \\
& x_1 & & +\frac{1}{7}x_3 & +\frac{2}{7}x_4 & & & = & \frac{20}{7} \\
& & x_2 & & +x_4 & & & = & 3 \\
& & & -\frac{2}{7}x_3 & +\frac{10}{7}x_4 & +x_5 & & = & \frac{23}{7} \\
& & & \frac{1}{7}x_3 & +\frac{2}{7}x_4 & & +t & = & -\frac{1}{7} \\
& x_1, & x_2, & x_3, & x_4, & x_5, & t & \geq & 0
\end{array}$$

is infeasible, $\overline{z}_2 = -\infty$, and hence node S_2 is *pruned by infeasibility.*

Choosing a Node. The node list now contains S_{11}, S_{12}. Arbitrarily choosing S_{12}, we remove it from the list.

Reoptimizing. $S_{12} = S \cap \{x : x_1 \leq 2, x_2 \geq 1\}$. The resulting linear program has optimal solution $\overline{x}^{12} = (2, 1)$ with value 7. As $\overline{x}^{12}$ is integer, $z^{12} = 7$.

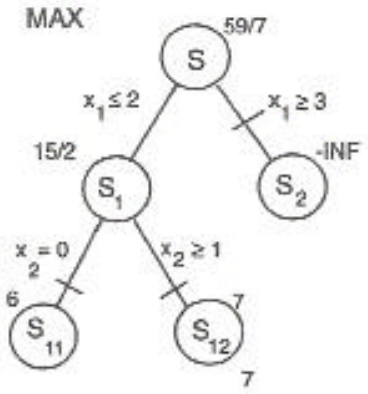

Fig. 7.8 Complete branch and bound tree

Updating the Incumbent. As the solution of $LP(S_{12})$ is integer, we update the value of the best feasible solution found $\underline{z} \leftarrow \max\{\underline{z}, 7\}$, and store the corresponding solution $(2,1)$. S_{12} is now *pruned by optimality.*

Choosing a Node. The node list now contains only S_{11}.

Reoptimizing. $S_{11} = S \cap \{x : x_1 \leq 2, x_2 \leq 0\}$. The resulting linear program has optimal solution $\overline{x}^{11} = (\frac{3}{2}, 0)$ with value 6. As $\underline{z} = 7 > \overline{z}_{11} = 6$, the node is *pruned by bound.*

Choosing a Node. As the node list is empty, the algorithm terminates. The incumbent solution $x = (2,1)$ with value $z = 7$ is optimal.

The complete branch-and-bound tree is shown in Figure 7.8. In Figure 7.9 we show graphically the feasible node sets S_i, the branching, the relaxations $LP(S_i)$, and the solutions encountered in the example.

7.4 LP-BASED BRANCH AND BOUND

In Figure 7.10 we present a flowchart of a simple branch and bound algorithm, and then discuss in more detail some of the practical aspects of developing and using such an algorithm.

Storing the Tree. In practice one does not store a tree, but just the list of *active* nodes or subproblems that have not been pruned and that still need to be explored further. Here the question arises of how much information one should keep. Should one keep a minimum of information and be prepared to repeat certain calculations, or should one keep all the information available? At a minimum, the best known dual bound and the variable lower and upper

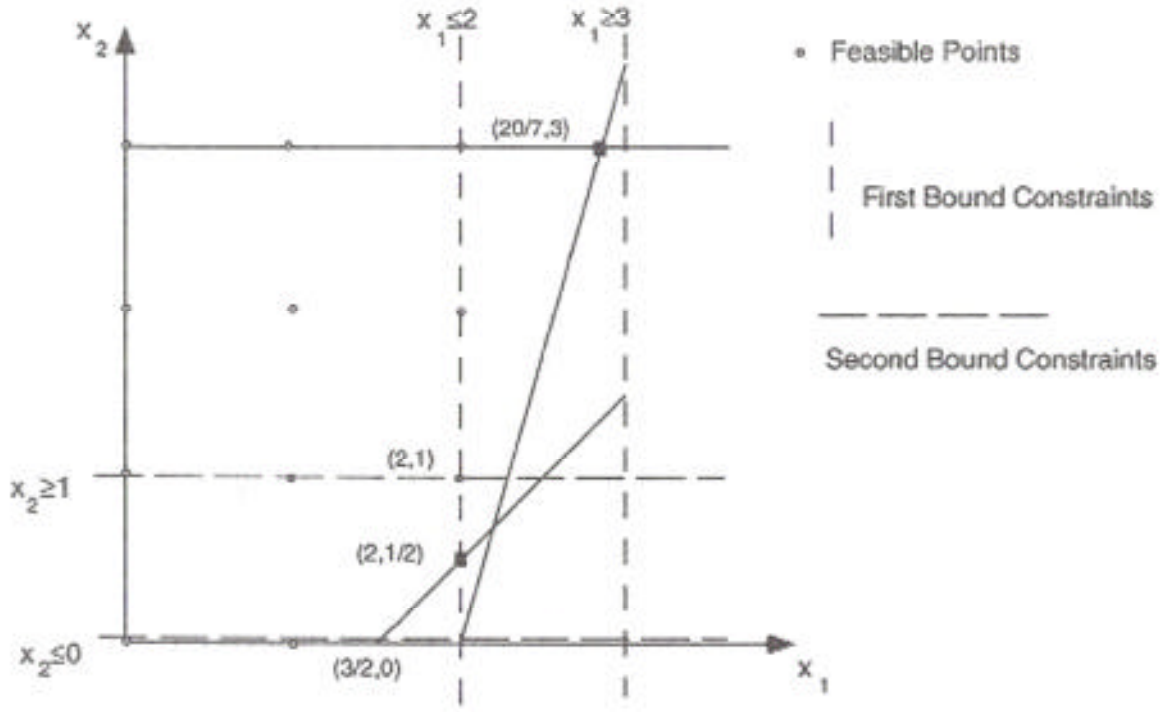

Fig. 7.9 Division of the feasible region

bounds needed to restore the subproblem are stored. Usually one also keeps an optimal or near-optimal basis, so that the linear programming relaxation can be reoptimized rapidly.

Returning to the questions raised earlier, there is no single answer that is best for all instances. One needs to use rules based on a combination of theory, common sense, and practical experimentation. In our example, the question of **how to bound** was solved by using an LP relaxation; **how to branch** was solved by choosing an integer variable that is fractional in the LP solution. However, as there is typically a choice of a set C of several candidates, we need a rule to choose between them. One common choice is *the most fractional variable*:

$$\arg\max_{j \in C} \min[f_j, 1 - f_j]$$

where $f_j = x_j^* - \lfloor x_j^* \rfloor$, so that a variable with fractional value $f_j = \frac{1}{2}$ is best. Other rules are based on the idea of *estimating* the cost of forcing the variable x_j to become integer.

How to choose a node was avoided by making an arbitrary choice. In practice there are several contradictory arguments that can be invoked:

(i) It is only possible to prune the tree significantly with a (primal) feasible solution, giving a hopefully good lower bound. Therefore one should descend as quickly as possible in the enumeration tree to find a first feasible solution. This suggests the use of a *Depth-First Search* strategy. Another argument for

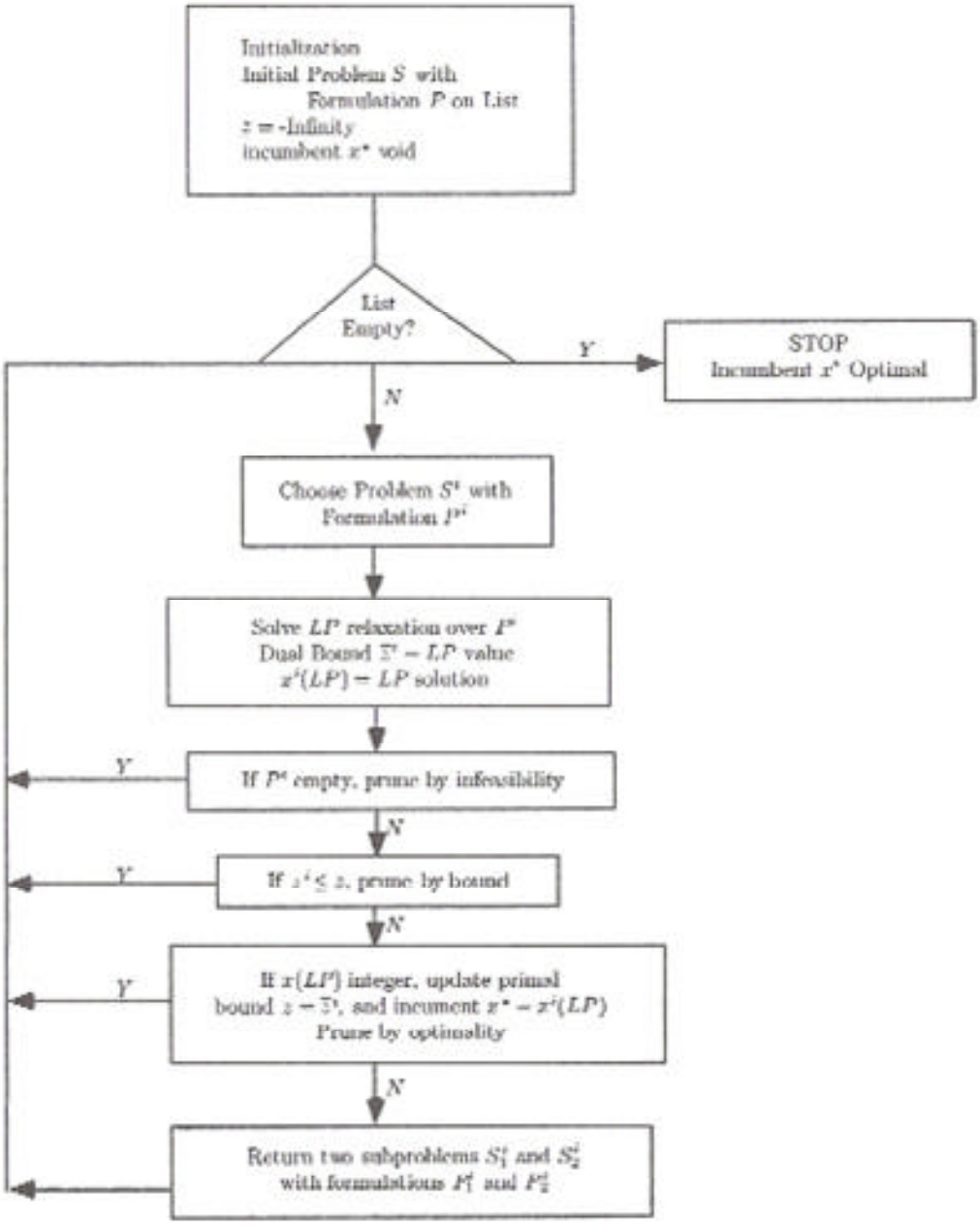

Fig. 7.10 Branch-and-bound flow chart

such a strategy is the observation that it is always easy to resolve the linear programming relaxation when a simple constraint is added, and the optimal basis is available. Therefore passing from a node to one of its immediate descendants is to be encouraged. In the example this would imply that after treating node S_1, the next node treated would be S_{11} or S_{12} rather than S_2.

(ii) To minimize the total number of nodes evaluated in the tree, the optimal strategy is to always choose the active node with the best (largest upper) bound (i.e., choose node s where $\overline{z}_s = \max_t \overline{z}_t$). With such a rule, one will never divide any node whose upper bound $\overline{z}_t$ is less than the optimal value z. This leads to a *Best-Node First* strategy. In the example of the previous section, this would imply that after treating node S_1, the next node chosen

would be S_2 with bound $\frac{59}{7}$ from its predecessor, rather than S_{11} or S_{12} with bound $\frac{15}{2}$.

In practice a compromise between these ideas is often adopted, involving an initial depth-first strategy until at least one feasible solution has been found, followed by a strategy mixing best node and depth first so as to try to prove optimality and also find better feasible solutions.

7.5 USING A BRANCH-AND-BOUND SYSTEM

Commercial branch-and-bound systems for integer and mixed integer programming are essentially as described in the previous section, and the default strategies have been chosen by tuning over hundreds of different problem instances. The basic philosophy is to solve and resolve the linear programming relaxations as rapidly as possible, and if possible to branch intelligently. Given this philosophy, all recent systems contain, or offer,

1. A powerful (automatic) preprocessor, which simplifies the model by reducing the number of constraints and variables, so that the linear programs are easier
2. The simplex algorithm with a choice of pivoting strategies, and an interior point option for solving the linear programs
3. Limited choice of branching and node selection options
4. Use of priorities

and some offer

5. GUB/SOS branching
6. Strong branching
7. Reduced cost fixing
8. Primal heuristics

In this section we briefly discuss those topics requiring user intervention. Preprocessing, which is very important, but automatic, is presented in the (optional) next section. Reduced cost fixing is treated in Exercise 7.7, and primal heuristics are discussed in Chapter 12.

Priorities. Priorities allow the user to tell the system the relative importance of the integer variables. The user provides a file specifying a value (importance) of each integer variable. When it has to decide on a branching variable, the system will choose the highest priority integer variable whose current linear programming value is fractional. At the same time the user can specify a preferred branching direction telling the system which of the two branches to