5

Branch and cut algorithms

Abilio Lucena

Laboratório Nacional de Computação Científica,
Rio de Janeiro, Brazil 22290

John E. Beasley

The Management School, Imperial College, London SW7 2AZ, England

1 Introduction

Let $\mathbb{R}_+^p$ and $\mathbb{Z}_+^n$ be, respectively, the set of non-negative real-valued p-dimensional vectors and the set of non-negative integral valued n-dimensional vectors. For variables $x \in \mathbb{R}_+^p$ and $y \in \mathbb{Z}_+^n$, a *linear mixed-integer programming* (MIP) problem is given by

$$\min\{cx + hy : \quad Ax + Gy \geq b, \quad x \in \mathbb{R}_+^p, \ y \in \mathbb{Z}_+^n\}, \tag{1.1}$$

where c is a p-vector, h is an n-vector, b is an m-vector, A is an $m \times p$ matrix, and G is an $m \times n$ matrix. The *feasible region* associated with (1.1) is

$$P = \{Ax + Gy \geq b, \quad x \in \mathbb{R}_+^p, \ y \in \mathbb{Z}_+^n\}, \tag{1.2}$$

while

$$z = cx + hy \tag{1.3}$$

is the *objective function* to be optimized over P. Whenever all variables in (1.1) are restricted to be integral, the problem is known as a *linear integer programming* (IP) problem. Conversely, a pure *linear programming* (LP) problem results when all variables in (1.1) are restricted to be real valued.

As is frequently the case for MIP, instead of attempting to optimize (1.3) directly over P, it may be advantageous to divide that region into a finite number of smaller regions and optimize the objective function over each smaller region individually. In the process one must ensure that an optimal solution to (1.1) is contained in at least one of the smaller regions that were generated. This basic solution strategy can be further extended to include the newly defined MIP problems. In particular, the idea would be attractive when the resulting

The authors wish to thank Dr. C.E. Ferreira of the University of São Paulo for some helpful comments on this chapter.

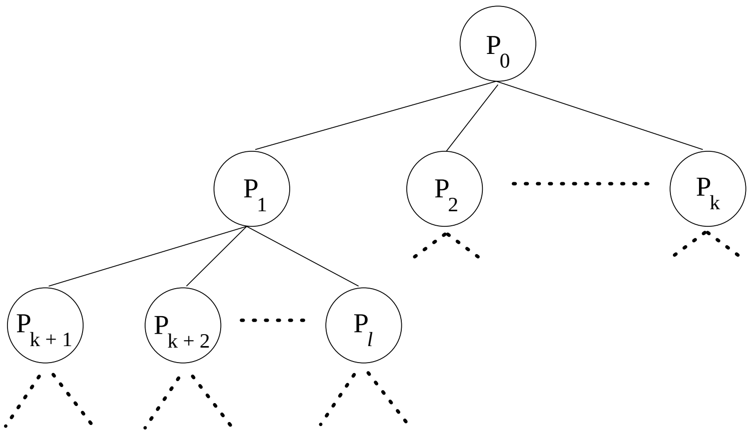

FIG. 1. Search tree

problems cannot be easily solved with the tools to hand. For most MIP problems, taken to the limit, the idea would imply almost total enumeration. Therefore it would only be of use for extremely small problems. A convenient representation for the process just described is in terms of a tree (see Fig. 1). A node in the *search tree* of Fig. 1, say node k, represents the MIP problem given by

$$\min\{cx + hy : (x, y) \in P_k\}, \tag{1.4}$$

where $P_k \subseteq P$. Notice that problem (1.1) would be represented, in this case, at the root node ($k = 0$) of the tree (i.e. $P_0 = P$). As one moves down the search tree, the regions P_k become smaller and smaller. Eventually, one may be able, for a given P_k, to either solve (1.4) to optimality, or else determine that no optimal solution to (1.1) is contained in P_k. For such a scheme to be applicable in practice, one tends to work with *relaxations* of (1.4) instead of (1.4) proper. For instance, a relaxation of (1.4) is

$$\min\{w(x, y) : (x, y) \in R_k\}, \tag{1.5}$$

where $P_k \subseteq R_k$ and $w(x, y) \leq cx + hy, (x, y) \in R_k$. Clearly, (1.5) gives a lower bound on (1.4). To be fair, nevertheless, (1.5) must be a lot easier to solve than (1.4). An example of a relaxation that is going to be extensively used in this chapter is the *LP relaxation*. A valid LP relaxation of (1.1), for instance, has the objective function $w(x, y) = cx + hy$ and the feasible region $R = \{Ax + Gy \geq b,\ (x, y) \in \mathbb{R}_+^{p+n}\}$. One should notice that the solution strategy outlined above is considerably enhanced with the introduction of MIP relaxations. With that modification in place, at a given (search) tree node, whenever the associated MIP problem cannot be easily solved, a relaxation of it is solved instead. As a result, a better judgement can be made of the subsequent actions to follow. The solution strategy can be enhanced even further if one is also capable of generating a good upper bound (corresponding to a feasible solution of

1.1), $v(ub)$ say. This would result in smaller search trees since a tree node (i.e. the optimization problem defined over, say P_k) is *fathomed* (i.e. the corresponding problem can be discarded) whenever it can be established that P_k does not contain a solution that improves on $v(ub)$. The overall scheme just described is generically called *branch and bound* (see Chapter 4 in Garfinkel and Nemhauser [18], Garfinkel [17] or Chapter II.4 in Nemhauser and Wolsey [40]). For branch and bound algorithms branching deals with the process of conveniently dividing feasible regions, at every tree node, into smaller regions. Bounding deals with the generation of upper and lower bounds for the (sub)problems involved.

A *branch and cut* algorithm (see Padberg and Rinaldi [42]) can be viewed as branch and bound where one systematically attempts to obtain stronger LP relaxations (at every node of the search tree). Stronger is used here in the sense of implying larger objective function values. In particular, this is attempted with the aid of *cutting planes* (i.e. linear inequalities which, depending on their type, could be valid only for the feasible regions implied or else globally valid across the search tree). Cutting planes in branch and cut algorithms are polyhedral in nature (see Chapter I.4 in Nemhauser and Wolsey [40] for the theory of polyhedral cutting planes) and are globally valid across the search tree. This represents a major departure from traditional cutting planes such as Gomory cuts (see Gomory [23, 24]), for instance. These usually can be guaranteed to be valid only for the subproblems for which they were generated. This would imply, amongst other things, massive computer memory requirements for storing the cuts generated during the search tree. Furthermore, in general, polyhedral cutting planes do not possess the (widely reported) poor convergence properties associated with the use of general cutting planes (Gomory cuts in particular). Polyhedral cutting planes are easier to explain if one concentrates on IPs. The basic ideas conveyed, nevertheless, can be readily extended to MIPs.

Consider a polyhedral region

$$R = \{x \in \mathbb{R}^n_+ : \quad Ax \geq b\}, \tag{1.6}$$

where (A, b) is an $m \times (n+1)$ matrix. Then the IP problem we consider is:

$$\min\{cx : x \in P\}, \tag{1.7}$$

where $P = R \cap \mathbb{Z}^n_+$ is the feasible region associated with this problem. Defining $conv(P)$ as the convex hull of the elements of P, it is possible to show (see page 106 in [40]) that the IP can be solved to optimality by simply solving the LP program

$$\min\{cx : x \in conv(P)\}. \tag{1.8}$$

Therefore, the key message conveyed is that the more an LP relaxation of (1.7) approximates $conv(P)$, the better. As much as is practicable, therefore, one should aim at obtaining the strongest possible approximations of $conv(P)$ as the LP relaxation of choice. For most IP problems, an explicit description of

$conv(P)$ by linear inequalities is out of the question. Nevertheless, experience shows that dramatic computational gains can be realized by working with "good partial descriptions" of $conv(P)$. In essence, at a tree node of a branch and cut algorithm, an LP relaxation of the problem (subproblem) is made stronger with the introduction of cutting planes that induce *facets* (or, if these are not available, hopefully, *faces* of high dimension) of the polyhedron $conv(P)$.

We will describe the key elements in a branch and cut algorithm by concentrating on two combinatorial optimization problems: the Minimum Spanning Tree (MST) problem and the Steiner Problem in Graphs (SPG). For a given connected graph, MSTs can be obtained in polynomial time while the SPG is NP-hard even for planar graphs (see Garey and Johnson [16]). In spite of this substantial difference, as will become apparent in the following sections, the two problems have quite a lot in common.

The term "branch and cut" was coined by Padberg and Rinaldi [42]. Most of the basic methodology associated with branch and cut algorithms can be traced to the seminal work of Dantzig *et al.* [11] for the travelling salesman problem (TSP). A considerable amount of experimentation has occurred before arriving at the fairly consensual setting for branch and cut algorithms that exists today. An incomplete list of references that contributed to this state of affairs follows.

Grötschel [26] solved a 120-city TSP using polyhedral cutting planes (which were generated visually). Padberg and Hong [41] used an automatic procedure to solve to optimality TSPs with up to 120 cities. Crowder and Padberg [10] used the technique to solve TSPs of sizes up to 318 cities. They used polyhedral cutting planes (which were generated automatically) and a commercially available MIP solver. For [10] cutting planes were only used to obtain an initial LP relaxation bound. From that point on the MIP solver was used to perform branch and bound whenever optimality could not be achieved at the root node of the search tree. Grötschel *et al.* [27] described an exact solution algorithm for the Linear Ordering Problem that could be considered as the very first true branch and cut algorithm. In [27] polyhedral cutting planes could be generated at any node of the search tree. Padberg and Rinaldi [42] pushed the methodology to new heights with their branch and cut algorithm for the TSP. They were capable of solving to proven optimality TSPs with 2,392 cities. Finally, Applegate, Bixby, Chvátal and Cook [2] developed a branch and cut algorithm that has been able to solve to proven optimality a TSP with 7,397 cities.

2 Solving the Minimum Spanning Tree problem by LP

Let $G = (V, E)$ be a finite, connected, undirected graph with a set of vertices V and a set of edges E. Sets V and E have cardinalities $m = |V|$ and $n = |E|$ respectively. A tree is a connected acyclic partial subgraph of G. The tree is said to be spanning if it contains exactly $|V| - 1$ edges. Given costs c_e for all edges $e \in E$ the cost of a tree equals the sum of the costs of the edges it contains. The MST problem is to find a spanning tree of minimum cost. For a connected graph G, the greedy algorithm of Kruskal [31]: