Branch-and-Cut Algorithms for Combinatorial Optimization Problems[1]

John E. Mitchell[2]
Mathematical Sciences
Rensselaer Polytechnic Institute
Troy, NY, USA
email: mitchj@rpi.edu
http://www.math.rpi.edu/~mitchj

April 19, 1999, revised September 7, 1999.

Abstract

Branch-and-cut methods are very successful techniques for solving a wide variety of integer programming problems, and they can provide a guarantee of optimality. We describe how a branch-and-cut method can be tailored to a specific integer programming problem, and how families of general cutting planes can be used to solve a wide variety of problems. Other important aspects of successful implementations are discussed in this chapter. The area of branch-and-cut algorithms is constantly evolving, and it promises to become even more important with the exploitation of faster computers and parallel computing.

1 Introduction

Many combinatorial optimization problems can be formulated as mixed integer linear programming problems. They can then be solved by branch-and-cut methods, which are exact algorithms consisting of a combination of a *cutting plane method* with a *branch-and-bound algorithm*. These methods work by solving a sequence of linear programming relaxations of the integer programming problem. Cutting plane methods improve the relaxation of the problem to more closely approximate the integer programming problem, and branch-and-bound algorithms proceed by a sophisticated divide and conquer approach to solve problems. Branch-and-bound methods are discussed elsewhere in this Handbook. In this chapter, we discuss cutting plane methods and their integration with branch-and-bound into branch-and-cut methods. Many problems in the various application areas discussed in this Handbook have been attacked using these methods.

Cutting plane algorithms for general integer programming problems were first proposed by Gomory (1963) (see §4.1). Unfortunately, the cutting planes proposed by Gomory did not appear to be very strong, leading to slow convergence of these algorithms, so the algorithms were neglected for many years. The development of polyhedral theory and the consequent introduction of strong, problem specific cutting planes led to a resurgence of cutting plane methods

[1]To appear in the *Handbook of Applied Optimization*, Oxford University Press, 2000.
[2]Research supported in part by NSF Grant CCR–9901822

in the 1980's, and cutting plane methods are now the method of choice for a wide variety of problems. Perhaps the best known branch-and-cut algorithms are those that have been used to solve the *traveling salesman problem* (TSP) (see §4.2). This approach is able to solve and prove optimality of far larger instances than other methods. Two papers that describe some of this research and also serve as good introductions to the area of branch-and-cut algorithms are Grötschel and Holland (1991); Padberg and Rinaldi (1991). A more recent work on the branch-and-cut approach to the TSP is Applegate *et al.* (1994). The successful research on the TSP is based upon the use of polyhedral theory to find strong cutting planes.

Branch-and-cut methods have also been used to solve other combinatorial optimization problems, again through the exploitation of strong cutting planes arising from polyhedral theory. Problems attacked recently with cutting plane or branch-and-cut methods include the linear ordering problem, maximum cut problems, scheduling problems, network design problems, packing problems, the maximum satisfiability problem, biological and medical applications, and finding maximum planar subgraphs. Recent surveys include Caprara and Fischetti (1997); Jünger *et al.* (1995).

Branch-and-cut methods for general integer programming problems are also of great interest (see, for example, Balas *et al.*, 1996a; Ceria *et al.*, 1998; Cordier *et al.*, 1997; Crowder *et al.*, 1983; Jünger and Thienel, 1998; Nemhauser *et al.*, 1994). It is usually not possible to efficiently solve a general integer programming problem using just a cutting plane approach, and it is therefore necessary to also branch, resulting in a branch-and-cut approach. A pure branch-and-bound approach can be sped up considerably by the employment of a cutting plane scheme, either just at the top of the tree, or at every node of the tree, because the cutting planes lead to a considerable reduction in the size of the tree.

For general problems, the specialized facets used when solving a specific combinatorial optimization problem are not available. Useful families of general inequalities include cuts based on knapsack problems (Crowder *et al.*, 1983), Gomory cutting planes (Gomory, 1963; Balas *et al.*, 1996b), and lift and project cutting planes (Balas *et al.*, 1996a). Cutting planes and polyhedral theory are discussed in more detail in §4. Before that, an example of a branch-and-cut algorithm is given in §2, and an algorithm is outlined in §3.

The software packages MINTO (Nemhauser *et al.*, 1994) and ABACUS (Jünger and Thienel, 1998) implement branch-and-cut algorithms to solve integer programming problems. The packages use standard linear programming solvers to solve the relaxations and they have a default implementation available. They also offer the user many options, including how to add cutting planes and how to branch. The commercial packages CPLEX and XPRESS-MP have also incorporated cutting plane generation into their branch-and-bound algorithms. Many refinements are required for an efficient implementation of a branch-and-cut algorithm and some of these are detailed in §5.

Nemhauser and Wolsey (1988) and Wolsey (1998) provide excellent and detailed descriptions of cutting plane algorithms and the other material in this entry, as well as other aspects of integer programming. Schrijver (1986) is an excellent source of additional material.

One aspect of a branch-and-cut approach that should not be overlooked is that it can be used to provide bounds. In particular, if we are minimizing but we are unable to prove optimality, a lower bound on the optimal value can be deduced from the algorithm, which can be used to provide a guarantee on the distance from optimality. Therefore, for large and/or hard problems, branch-and-cut can be used in conjunction with heuristics or metaheuristics to obtain a good (possibly optimal) solution and also to indicate how far from optimality this solution may be.

2 A simple example

The integer programming problem

$$\begin{array}{lrcrcll} \min \quad z := & -6x_1 & - & 5x_2 & & & \\ \text{subject to} & 3x_1 & + & x_2 & \leq & 11 & \\ & -x_1 & + & 2x_2 & \leq & 5 & \\ & & & x_1, x_2 & \geq & 0, & \text{integer.} \end{array} \qquad (Eg0)$$

is illustrated in Figure 1. The feasible integer points are marked. The *linear programming relaxation* (or *LP relaxation*) is obtained by ignoring the integrality restrictions and is indicated by the polyhedron contained in the solid lines.

A branch-and-cut approach first solves the linear programming relaxation, giving the point $(2\frac{3}{7}, 3\frac{5}{7})$, with value $-33\frac{1}{7}$. There is now a choice: should the LP relaxation be improved by adding a cutting plane, for example, $x_1 + x_2 \leq 5$, or should the problem be divided into two by splitting on a variable?

If the algorithm splits on x_1, two new problems are obtained:

$$\begin{array}{lrcrcll} \min \quad z := & -6x_1 & - & 5x_2 & & & \\ \text{subject to} & 3x_1 & + & x_2 & \leq & 11 & \\ & -x_1 & + & 2x_2 & \leq & 5 & \\ & \mathbf{x_1} & & & \geq & \mathbf{3} & \\ & & & x_1, x_2 & \geq & 0, & \text{integer.} \end{array} \qquad (Eg1)$$

and

$$\begin{array}{lrcrcll} \min \quad z := & -6x_1 & - & 5x_2 & & & \\ \text{subject to} & 3x_1 & + & x_2 & \leq & 11 & \\ & -x_1 & + & 2x_2 & \leq & 5 & \\ & \mathbf{x_1} & & & \leq & \mathbf{2} & \\ & & & x_1, x_2 & \geq & 0, & \text{integer.} \end{array} \qquad (Eg2)$$

The optimal solution to the original problem will be the better of the solutions to these two subproblems. The solution to the linear programming relaxation of $(Eg1)$ is $(3, 2)$, with value -28. This solution is integral, so it solves $(Eg1)$, and becomes the incumbent best known feasible solution. The LP relaxation of

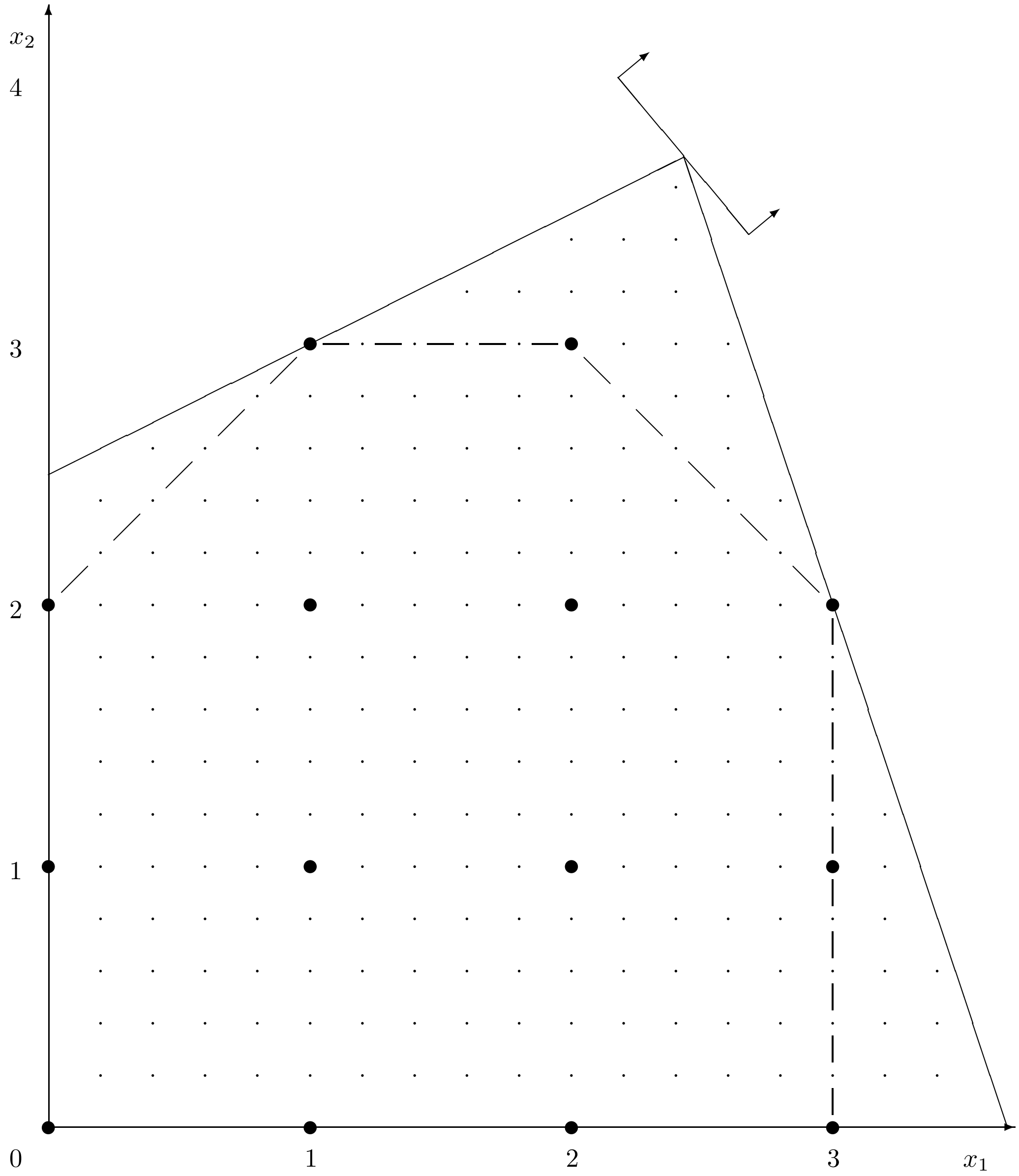

Figure 1: A two dimensional integer programming problem

$(Eg2)$ has optimal solution $(2, 3.5)$, with value -29.5. This point is nonintegral, so it does not solve $(Eg2)$, and it must be attacked further.

Assume the algorithm uses a cutting plane approach and adds the inequality $2x_1 + x_2 \leq 7$ to $(Eg2)$. This is a valid inequality, in that it is satisfied by every integral point that is feasible in $(Eg2)$. Further, this inequality is violated by $(2, 3.5)$, so it is a cutting plane. The resulting subproblem is

$$\begin{array}{llrcrcl}
\min & z := & -6x_1 & - & 5x_2 & & \\
\text{subject to} & & 3x_1 & + & x_2 & \leq & 11 \\
& & -x_1 & + & 2x_2 & \leq & 5 \\
& & x_1 & & & \leq & 2 \\
& & \mathbf{2x_1} & + & \mathbf{x_2} & \leq & \mathbf{7} \\
& & & & x_1, x_2 & \geq & 0, \text{ integer.}
\end{array} \qquad (Eg3)$$

The LP relaxation of $(Eg3)$ has optimal solution $(1.8, 3.4)$, with value -27.8. Notice that the optimal value for this modified relaxation is larger than the value of the incumbent solution. The value of the optimal integral solution to the second subproblem must be at least as large as the value of the relaxation. Therefore, the incumbent solution is better than any feasible integral solution for $(Eg3)$, so it actually solves the original problem.

The progress of the algorithm is illustrated in Figure 2.

Of course, there are several issues to be resolved with this algorithm. These include the major questions of deciding whether to branch or to cut (§5.5) and deciding how to branch and how to generate cutting planes (§4). Notice that the cutting plane introduced in the second subproblem is not valid for the first subproblem. This inequality can be modified to make it valid for the first subproblem by using a *lifting* technique, which is discussed in §5.6.

3 Outline of an algorithm

We regard the mixed integer linear programming problem

$$\begin{array}{lrcl}
\min & c^T x & & \\
\text{subject to} & Ax & \leq & b \\
& x & \geq & 0 \\
& x_i & & \text{integer, } i = 1, \ldots, p.
\end{array} \qquad (ILP)$$

as our standard form, where x and c are n-vectors, b is an m-vector, and A is an $m \times n$ matrix. The first p variables are restricted to be integer, and the remainder may be fractional. If $p = n$ then this is an integer programming problem. If a variable is restricted to take the values 0 or 1 then it is a binary variable. If all variables are binary then the problem is a binary program. There is no loss of generality with restricting attention to such a format.

A branch-and-cut algorithm is outlined in Figure 3. Notice that L is the set of active nodes in the branch-and-cut tree. The value of the best known feasible point for (ILP) is $\bar{z}$, which provides an upper bound on the optimal

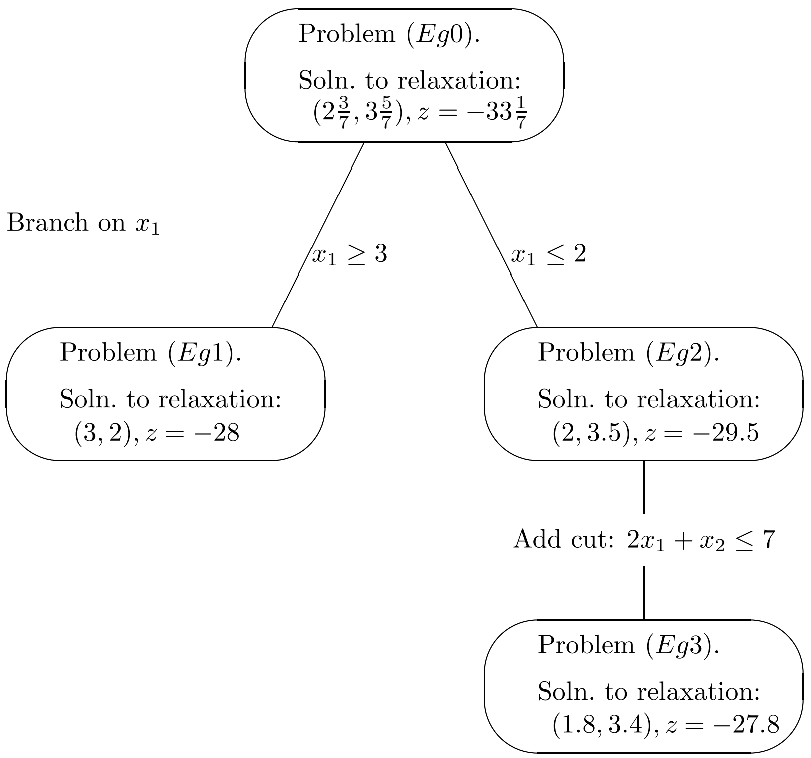

Figure 2: Progress of branch-and-cut on the two dimensional integer programming problem

1. *Initialization*: Denote the initial integer programming problem by ILP^0 and set the active nodes to be $L = \{ILP^0\}$. Set the upper bound to be $\bar{z} = +\infty$. Set $\underline{z_l} = -\infty$ for the one problem $l \in L$.

2. *Termination*: If $L = \emptyset$, then the solution x^* which yielded the incumbent objective value $\bar{z}$ is optimal. If no such x^* exists (i.e., $\bar{z} = +\infty$) then ILP is infeasible.

3. *Problem selection*: Select and delete a problem ILP^l from L.

4. *Relaxation:* Solve the linear programming relaxation of ILP^l. If the relaxation is infeasible, set $\underline{z_l} = +\infty$ and go to Step 6. Let $\underline{z_l}$ denote the optimal objective value of the relaxation if it is finite and let x^{lR} be an optimal solution; otherwise set $\underline{z_l} = -\infty$.

5. *Add cutting planes*: If desired, search for cutting planes that are violated by x^{lR}; if any are found, add them to the relaxation and return to Step 4.

6. *Fathoming and Pruning*:

   (a) If $\underline{z_l} \geq \bar{z}$ go to Step 2.

   (b) If $\underline{z_l} < \bar{z}$ and x^{lR} is integral feasible, update $\bar{z} = \underline{z_l}$, delete from L all problems with $\underline{z_l} \geq \bar{z}$, and go to Step 2.

7. *Partitioning*: Let $\{S^{lj}\}_{j=1}^{j=k}$ be a partition of the constraint set S^l of problem ILP^l. Add problems $\{\text{ILP}^{lj}\}_{j=1}^{j=k}$ to L, where ILP^{lj} is ILP^l with feasible region restricted to S^{lj} and $\underline{z_{lj}}$ for $j = 1, \ldots, k$ is set to the value of $\underline{z_l}$ for the parent problem l. Go to Step 2.

Figure 3: A general branch-and-cut algorithm

value of (ILP). Further, $\underline{z_l}$ is a lower bound on the optimal value of the current subproblem under consideration. The value of the LP relaxation of the subproblem can be used to update $\underline{z_l}$. In some situations, a very large number of violated cutting planes are found in Step 5, in which case it is common to sort the cutting planes somehow (perhaps by violation), and add just a subset. The subproblems formed in Step 7 are called child subproblems, with the previous problem ILP^l being the parent subproblem. Usually, the partitioning takes the form of a variable disjunction: $x_i \leq a$ versus $x_i \geq a+1$ for some variable x_i and integer a, as in the example. Other choices are possible, and they are discussed more in the branch-and-bound Chapter.

The relaxations can be solved using any method for linear programming problems. Typically, the initial relaxation is solved using the simplex method. Subsequent relaxations are solved using the dual simplex method, since the dual solution for the relaxation of the parent subproblem is still feasible in the relaxation of the child subproblem. Further, when cutting planes are added in Step 5, the current iterate is still dual feasible, so again the modified relaxation can be solved using the dual simplex method. It is also possible to use an interior point method, and this can be a good choice if the linear programming relaxations are large — see §5.8 for more discussion of this option.

If the objective function and/or the constraints in (ILP) are nonlinear, the problem can still be attacked with a branch-and-cut approach. For more information about such problems, see the Chapter on mixed integer nonlinear programming problems.

4 Polyhedral theory and cutting planes

We say that any inequality $\pi^T x \leq \pi_0$ that is satisfied by all the feasible points of (ILP) is a *valid inequality*. The convex hull of the set of feasible solutions to (ILP) is a polyhedron. Every valid inequality defines a face of this polyhedron, namely the set of all the points in the polyhedron that satisfy $\pi^T x = \pi_0$. A facet is a face of a polyhedron that has dimension one less than the dimension of the polyhedron, and it is necessary to have an inequality that represents each facet in order to have a complete linear inequality description of the polyhedron. If all the facets of the convex hull of the set of integer feasible points are known, then the integer problem can be solved as a linear programming problem by minimizing the objective function over this convex hull. Unfortunately, it is not easy to obtain such a description. In fact, for an NP-Complete problem, such a description must contain an exponential number of facets, unless P=NP.

In the example above, the convex hull of the set of feasible integer points has dimension 2, and all of the dashed lines represent facets. The valid inequality $3x_1 + x_2 \leq 11$ represents a face of the convex hull of dimension 0, namely the point $(3, 2)$.

Chvátal-Gomory, strong, and general cutting planes are described in §4.1, §4.2, and §4.3, respectively.

4.1 Chvátal-Gomory cutting planes

Cutting planes can be obtained by first combining together inequalities from the current linear programming relaxation and then exploiting the fact that the variables must be integral. This process is known as *integer rounding*, and the cutting planes generated are known as *Chvátal-Gomory cutting planes*. Integer rounding was described implicitly by Gomory (1963), and described explicitly by Chvátal (1973).

Consider again the example problem given earlier. The first step is to take a weighted combination of the inequalities. For example,

$$\frac{1}{6}(3x_1 + x_2 \leq 11) + \frac{5}{12}(-x_1 + 2x_2 \leq 5)$$

gives the valid inequality for the relaxation:

$$\frac{1}{12}x_1 + x_2 \leq 3\frac{11}{12}.$$

Since all the variables are constrained to be nonnegative, rounding down the left hand side of this inequality will only weaken it, giving $x_2 \leq 3\frac{11}{12}$, also valid for the LP relaxation. In any feasible solution to the integer programming problem, the left hand side of this inequality must take an integer value. Therefore, the right hand side can be rounded down to give the following valid inequality for the integer programming problem:

$$x_2 \leq 3.$$

Gomory originally derived constraints of this form directly from the optimal simplex tableau. These constraints are then added to the tableau and the modified relaxation is solved. For example, after introducing slack variables x_3 and x_4, the optimal simplex tableau for the example problem can be written

$$\begin{array}{llcccccccl}
\text{min} & & & & 2\frac{3}{7}x_3 & + & 1\frac{2}{7}x_4 & =: & z + 33\frac{1}{7} \\
\text{subject to} & x_1 & & + & \frac{2}{7}x_3 & - & \frac{1}{7}x_4 & = & 2\frac{3}{7} \\
 & & x_2 & + & \frac{1}{7}x_3 & + & \frac{3}{7}x_4 & = & 3\frac{5}{7} \\
 & & & & \multicolumn{3}{c}{x_i \geq 0,\ i = 1, \ldots, 4} & &
\end{array}$$

Since x_1 and x_2 are constrained to be integer, it follows that z, x_3, and x_4 must also all be integer. For each row, the fractional parts of the left hand side and the right hand side must be equal. Thus, the objective function row of this tableau indicates that the constraint

$$\frac{3}{7}x_3 + \frac{2}{7}x_4 \geq \frac{1}{7} \tag{1}$$

must be satisfied. This constraint can be added to the tableau and the modified relaxation solved using the dual simplex method. The constraint can be written in terms of the original variables:

$$\frac{1}{7} \leq \frac{3}{7}x_3 + \frac{2}{7}x_4 = \frac{3}{7}(11 - 3x_1 - x_2) + \frac{2}{7}(5 + x_1 - 2x_2) = 6\frac{1}{7} - x_1 - x_2,$$

or, rearranging,

$$x_1 + x_2 \leq 6.$$

If this constraint is added to the relaxation, the optimal value is $z = -32.5$, achieved at $x = (2.5, 3.5)$. The Gomory cutting plane generated at this new point can be written $2x_1 + x_2 \leq 8$. Adding this constraint gives an optimal point of $x = (2.2, 3.6)$ with value $z = -31.2$, and the process can be iterated further.

Cutting planes can be generated from any constraint where the corresponding basic variable is fractional. The two constraint rows of the tableau given above imply the constraints

$$\frac{2}{7}x_3 + \frac{6}{7}x_4 \quad \geq \quad \frac{3}{7} \tag{2}$$

$$\frac{1}{7}x_3 + \frac{3}{7}x_4 \quad \geq \quad \frac{5}{7}, \tag{3}$$

which are equivalent to

$$\begin{aligned} 2x_2 &\leq 7 \\ x_2 &\leq 3, \end{aligned}$$

respectively.

Every valid inequality for the convex hull of the set of feasible points for (ILP) can be derived by repeatedly applying the Chvátal-Gomory rounding procedure (Chvátal, 1973). If a cutting plane is always generated from the first possible row then Gomory's cutting plane algorithm will solve an integer program in a finite number of iterations (Gomory, 1963). Balas *et al.* (1996b) have shown that the algorithm can be made competitive with other methods if certain techniques are used, such as adding many Chvátal-Gomory cuts at once. In the example, adding all three of the cuts (1)–(3) would give an optimal solution of $x = (2\frac{2}{3}, 3)$ with value $z = -31$; the Gomory cuts that would next be generated are equivalent to $x_1 + x_2 \leq 5$ and $2x_1 + x_2 \leq 8$, and adding these constraints would lead to the optimal solution $x = (3, 2)$, $z = -28$. Gomory cuts can contain a large number of nonzeroes, so care is required to ensure that the LP relaxation does not become very hard with large memory requirements. The cuts are generated directly from the basis inverse, so care must also be taken to avoid numerical difficulties.

Gomory cutting planes can also be derived for mixed integer linear programming problems. See, for example, Nemhauser and Wolsey (1988); Wolsey (1998) for more details.

4.2 Strong cutting planes from polyhedral theory

The resurgence of interest in cutting plane algorithms in the 1980's was due to the development of polyhedral combinatorics and the consequent implementation of cutting plane algorithms that used facets of the convex hull of integral feasible points as cuts. Typically, a partial polyhedral description of the convex hull of the set of integer feasible points is determined theoretically. This description will usually contain families of facets of certain types. *Separation routines* for these families can often be developed; such a routine will take as input a point (for example, the optimal solution to the LP relaxation), and return as output violated constraints from the family, if any exist.

In the remainder of this subsection, we illustrate the use of this approach through consideration of the *traveling salesman problem* (TSP). In this problem, a set of cities is provided along with distances between the cities. A route that visits each city exactly once and returns to the original city is called a tour. It

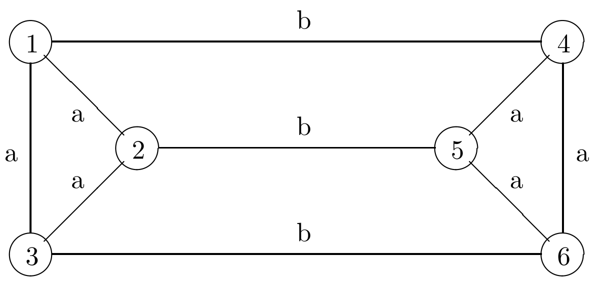

Figure 4: A traveling salesman example

is desired to choose the shortest tour. One application of the TSP is in printed circuit board (PCB) production: a PCB needs holes drilled in certain places to hold electronic components such as resistors, diodes, and integrated circuits. These holes can be regarded as the cities, and the objective is to minimize the total distance traveled by the drill.

The traveling salesman problem can be represented on a graph, $G = (V, E)$, where V is the set of vertices (or cities) and E is the set of edges (or links between the cities). Each edge $e \in E$ has an associated cost (or length) c_e. If the incidence vector x is defined by

$$x_e = \begin{cases} 1 & \text{if edge } e \text{ is used} \\ 0 & \text{otherwise} \end{cases}$$

then the traveling salesman problem can be formulated as

$$\begin{array}{ll} \min & \sum c_e x_e \\ \text{subject to} & x \text{ is the incidence vector of a tour.} \end{array}$$

The incidence vector of a tour must satisfy the *degree constraints* that the sum of the edge variables must be two at each vertex, giving the following relaxation:

$$\begin{array}{lll} \min & \sum c_e x_e & \\ \text{s.t.} & \sum_{e \in \delta(v)} x_e = 2 & \text{for all vertices } v \\ & x_e = 0 \text{ or } 1 & \text{for all edges } e, \end{array} \qquad (TSP1)$$

where $\delta(v)$ denotes the set of all edges incident to vertex v. All tours are feasible in this formulation, but it also allows infeasible solutions corresponding to *subtours*, consisting of several distinct unconnected loops. Consider, for example the graph shown in Figure 4, where the labeled edge lengths are $a = 1$, $b = 2$, and unshown edges have length 10. The point $x_{12} = x_{23} = x_{13} = x_{45} = x_{46} = x_{56} = 1$, $x_{ij} = 0$ for all other edges, solves $(TSP1)$.

Any tour must use two of the edges between the set of vertices $\{1, 2, 3\}$ and the set of vertices $\{4, 5, 6\}$. In general, *subtour elimination constraints* can be

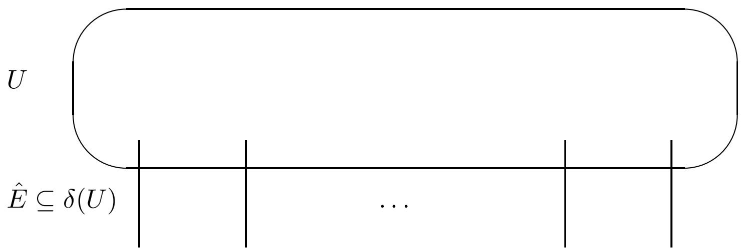

Figure 5: Graph for a 2-matching inequality

added to the relaxation. These take the form

$$\sum_{e \in \delta(U)} x_e \geq 2 \tag{4}$$

for every subset $U \subseteq V$ with cardinality $2 \leq\mid U \mid\leq \frac{|V|}{2}$, where $\delta(U)$ denotes the set of edges with exactly one endpoint in U. Any feasible solution to the relaxation given above which also satisfies the subtour elimination constraints must be the incidence vector of a tour. All of these inequalities are facets of the convex hull of the set of incidence vectors of tours. Now, the number of subtour elimination constraints is exponential in the number of cities, so the subtour elimination constraints are added as cutting planes as needed.

The degree constraints and the subtour elimination constraints, together with the simple bounds $0 \leq x_e \leq 1$, are still not sufficient to describe the convex hull of the set of incidence vectors of tours. Consider again the graph shown in Figure 4, where now the edge lengths are $a = 2$ and $b = 1$. All edges that are not shown have length 10. The point $x_{12} = x_{23} = x_{13} = x_{45} = x_{46} = x_{56} = 0.5$, $x_{14} = x_{25} = x_{36} = 1$ is feasible in $(TSP1)$ and satisfies (4). However, this point is not in the convex hull of the set of incidence vectors of tours. It violates the *2-matching inequality*

$$x_{12} + x_{23} + x_{13} + x_{14} + x_{25} + x_{36} \leq 4. \tag{5}$$

To obtain the general form of this inequality, consider Figure 5. Here, U is a subset of the vertices and $\hat{E}$ is an odd set of disjoint edges, each with exactly one endpoint in U. Let $E(U)$ be the set of edges that have both endpoints in U. A tour can use at most $\mid U \mid -1$ of the edges in $E(U)$, in which case it can use only two of the edges from $\hat{E}$. If more edges are used from $\hat{E}$, then fewer edges must be used from $E(U)$. The general form of constraint (5) is

$$\sum_{e \in E(U)} x_e + \sum_{e \in \hat{E}} x_e \leq\mid U \mid + \frac{\mid \hat{E} \mid -1}{2}. \tag{6}$$

This is a facet defining inequality, provided $\hat{E}$ contains at least three edges.

It can be generalized further to give the family of *generalized comb inequalities* (Nemhauser and Wolsey, 1988), which are also facet defining.

Many other families of valid inequalities have been discovered. Branch-and-cut methods search for violated inequalities from these families. These are the most successful methods for solving large instances of the traveling salesman problem (Applegate *et al.*, 1994; Grötschel and Holland, 1991; Padberg and Rinaldi, 1991) and similar ideas have been used to solve a variety of problems (Caprara and Fischetti, 1997; Jünger *et al.*, 1995; Nemhauser and Wolsey, 1988).

4.3 Alternative general cutting planes

A knapsack problem is an integer programming problem with just one linear inequality constraint. A general integer programming problem can be regarded as the intersection of several knapsack problems, one for each constraint. This observation was used in the breakthrough paper by Crowder *et al.* (1983) to solve general integer programming problems. The approach consists of finding facets and strong cutting planes for the knapsack problem and adding these constraints to the LP relaxation of the integer program as cutting planes. These inequalities have been extended to knapsacks with general integer variables and one continuous variable (Ceria *et al.*, 1998) and to binary problems with generalized upper bounds.

Another family of useful inequalities are *lift-and-project* or *disjunctive inequalities*. These were originally introduced by Balas, and it is only in the last few years that the value of these cuts has become apparent for general integer programming problems (Balas *et al.*, 1996a). Given the feasible region for a binary programming problem $S := \{x : Ax \leq b, x_i = 0, 1\ \forall i\}$, each variable can be used to generate a set of disjunctive inequalities. Let $S_j^0 := \{x : Ax \leq b, 0 \leq x_i \leq 1\ \forall i, x_j = 0\}$ and $S_j^1 := \{x : Ax \leq b, 0 \leq x_i \leq 1\ \forall i, x_j = 1\}$. Then $S \subseteq S_j^0 \cup S_j^1$, so valid inequalities for S can be generated by finding valid inequalities for the convex hull of $S_j^0 \cup S_j^1$. These inequalities are generated by solving linear programming problems. Because of the expense, the cuts are usually only generated at the root node. Nonetheless, they can be very effective computationally.

Other general cutting planes have been developed. Several families and routines for identifying violated inequalities are described and investigated computationally by Cordier *et al.* (1997).

These alternative general cutting planes are not usually strong enough on their own to solve an integer programming problem, and they are most successfully employed in branch and cut algorithms for integer programming.

5 Techniques for a good implementation

Many refinements to the basic algorithm are necessary to get the best possible performance out of a branch-and-cut code. These include using reduced costs to eliminate variables (§5.1), working with a subset of the variables and then adding in the omitted variables later if necessary (§5.2), using primal heuristics to generate good solutions so that nodes can be pruned by bounds (§5.3), preprocessing the problem (§5.4), maintaining an appropriate balance between cutting and branching (§5.5), and strengthening cuts through lifting (§5.6). Further details are discussed in §5.7 and the possibility of using an interior point method to solve the linear programming relaxations is discussed in §5.8.

5.1 Fixing variables

Nonbasic variables can be guaranteed to take their current values in an optimal solution to the integer programming problem if their reduced costs are sufficiently large. For example, if the binary variable x_j equals zero in the optimal solution to an LP relaxation and the reduced cost of this variable is r_j, then any feasible point in the relaxation with $x_j = 1$ must have value at least $\underline{z}+r_j$, where $\underline{z}$ is the optimal value of the relaxation. The value $\bar{z}$ of the best known feasible integral solution provides an upper bound on the optimal value of the integer program, so we must have $x_j = 0$ if $r_j > \bar{z} - \underline{z}$. Similar tests can be derived for nonbasic variables at their upper bounds. It is also possible to fix variables when an interior point method is used to solve the relaxations (Mitchell *et al.*, 1998).

Once some variables have been fixed in this manner, it is often possible to fix further variables using logical implications. For example, in a traveling salesman problem, if x_e has been set equal to one for two edges incident to a particular vertex, then all other edges incident to that vertex can have their values fixed to zero.

In order to fully exploit the fixing of variables within the branching process, *parent node reconstruction* (Padberg and Rinaldi, 1991) is performed as follows. Once a parent node has been selected, it is not immediately divided into two children, but is solved again using the cutting plane algorithm. When the cutting plane procedure terminates, the optimal reduced cost vector has been reconstructed and this is used to perform variable fixing.

5.2 Column generation

For many problems, including the traveling salesman problem, it is impractical to work explicitly with all the variables. Thus, the typical approach for the TSP is to find the ten (say) closest neighbours for each vertex and work only with the corresponding edges. In order to verify optimality, it is necessary to check at the end that none of the omitted edges are actually necessary. In linear programming terms, this requires checking whether a column should be added to the relaxation corresponding to the omitted variable. The reduced cost of this column can be found from the values of the dual variables: the column should be added if the extra dual constraint would be violated by the optimal dual solution to the current relaxation. If a large number of columns need to be added, it is usual to just add a subset of them, resolve, and check whether more need to be added.

5.3 Primal heuristics

In the example problem, the existence of a good incumbent solution made it possible to prune ($Eg3$) by bounds. In many cases, it takes many stages until the solution to a relaxation is integral. Therefore, it is often useful to have good heuristics for converting the fractional solution of a relaxation into a good integral solution that can be used to prune other subproblems. Primal heuristics that use the solution to the linear programming relaxation are discussed further in the Chapter on branch-and-bound algorithms. Branch-and-cut can also be combined with other heuristics such as local search, and also with metaheuristics.

5.4 Preprocessing

A very important component of a practical branch-and-cut algorithm is preprocessing to eliminate unnecessary constraints, determine any fixed variables, and simplify the problem in other ways. Preprocessing techniques are discussed in the Chapter on branch-and-bound algorithms.

5.5 When to add cutting planes

Usually, there comes a point at which the cutting plane loop in Steps 4 and 5 tails off, that is, the solution to one relaxation is not much better than the solutions to recent relaxations. The algorithm should then proceed to Step 6. It is believed that tailing off is a function of lack of knowledge about the polyhedral structure of the relaxation, rather than a fundamental weakness of the cutting plane approach (Padberg and Rinaldi, 1991).

The computational overhead of searching for cutting planes can be prohibitive. Therefore, it is common to not search at some nodes of the tree. Alternatives include searching at every eighth node, say, or at every node at a depth of a multiple of eight in the tree. In some implementations, a fixed number of rounds of cutting plane searching is performed at a node, with perhaps several rounds performed at the root node, and fewer rounds performed lower in the tree.

The *cut-and-branch* variant adds cutting planes only at the root node of the tree. Usually, an implementation of such a method will expend a great deal of effort on generating cutting planes, requiring time far greater than just solving the relaxation at the root. With cut-and-branch, all generated cuts are valid throughout the tree. Cut-and-branch is an excellent technique for many general integer programs, but it lacks the power of branch-and-cut for some hard problems. See Cordier *et al.* (1997) for more discussion of the relative computational performance of cut-and-branch and branch-and-cut.

5.6 Lifting cuts

A cut added at one node of the branch-and-cut tree may well not be valid for another subproblem. Of course, it is not necessary to add the cut at any other node, in which case the cut is called a *local* cut. This cut will then only affect the current subproblem and its descendants. The drawback to such an approach is in the potential memory requirement of needing to store a different version of the problem for each node of the tree. In order to make a cut valid throughout the tree (or *global*), it is necessary to *lift* it. Lifting can be regarded as a method of rotating a constraint.

For binary problems, the inequality generated at a node in the tree will generally only use the variables that are not fixed at that node. If the inequality

$$\sum_{j \in J} a_j x_j \leq h \text{ for some subset } J \subseteq \{1, \ldots, n\}$$

is valid at a node where x_i has been fixed to zero, the lifted inequality takes the form

$$\sum_{j \in J} a_j x_j + \alpha_i x_i \leq h$$

for some scalar α_i. This scalar should be maximized in order to make the inequality as strong as possible. Now, maximizing α_i requires solving another integer program where x_i is fixed at one, so it may be necessary to make an approximation and underestimate α_i. This process has to be applied successively to each variable that has been fixed at the node. The order in which the variables are examined may well affect the final inequality, and other valid inequalities can be obtained by lifting more than one variable at a time. See Ceria *et al.* (1998) for discussion of lifting in the case of general mixed integer linear programming problems.

5.7 Implementation details

Many details of tree management can be found in the Chapter on branch-and-bound algorithms. These include node selection, branching variable selection, and storage requirements, among other issues. Typically, a branch-and-bound algorithm stores the solution to a node as a list of the indices of the basic variables. If cuts are added locally then it is necessary to store a representation of the constraint set for each active node.

Many branch-and-cut implementations use a *pool of cuts* (Padberg and Rinaldi, 1991), which is a set of constraints that have been generated earlier and either not included in the relaxation or subsequently dropped because they no longer appeared to be active. These cuts can be checked quickly for violation before more involved separation routines are invoked. The pool of cuts also makes it possible to reconstruct the parent node more efficiently.

5.8 Solving large problems

The difficulty of a particular integer programming problem is not purely a function of the size of the problem. For example, there are problems in the standard MIPLIB test set (Bixby *et al.*, 1998) with just a few hundred variables that prove resistant to standard solution approaches, because of an explosion in the size of the tree.

For some problems, difficulties are caused by the size of the LP relaxation, and *interior point methods* may be useful in such cases. Interior point methods are superior to simplex methods for many linear programming problems with thousands of variables. For large integer programs, the first relaxation at the top node of the tree can be solved using an interior point method, and subsequent relaxations can be solved using the (dual) simplex method. For some problems, the relaxations are just too large to be handled with a simplex method, so interior point methods are used throughout the algorithm. Interior point methods handle degeneracy better than the simplex method. Therefore, for example, the branch-and-cut solver described by Applegate *et al.* (1994) occasionally uses an interior point method to handle some subproblems. Restarting is harder with an interior point method than with a simplex method when the relaxation is only slightly changed, so various techniques are needed for a successful interior point cutting plane algorithm. The use of interior point methods in branch-and-cut algorithms is surveyed by Mitchell *et al.* (1998). Interior point branch-and-bound methods are discussed in the Chapter on branch-and-bound.

One way to enable the solution of far larger problems is to use a *parallel computer*. The nature of branch-and-cut and branch-and-bound algorithms makes it possible for them to exploit coarse grain parallel computers efficiently: typically, a linear programming relaxation is solved on a node of the computer. It is possible to use one node to manage the distribution of linear programs to nodes. Alternatively, methods have been developed where a common data structure is maintained and all nodes access this data structure to obtain a relaxation that requires solution. It is also possible to generate cutting planes in parallel.

6 Conclusions

Branch-and-cut methods have proven to be a very successful approach for solving a wide variety of integer programming problems. In contrast with metaheuristics, they can guarantee optimality. For a structured integer programming problem, the branch-and-cut method should be tailored to the problem with an appropriate selection of cutting planes. These methods are also useful for general integer programming problems, where families of general cutting planes are used. Other important aspects of successful implementations have also been discussed in this chapter. The area of branch-and-cut algorithms is constantly evolving, and it promises to become even more important with the exploitation of faster computers and parallel computing.

References

D. Applegate, R. Bixby, V. Chvátal, and W. Cook. The traveling salesman problem. Technical report, DIMACS, Rutgers University, New Brunswick, NJ, 1994.

E. Balas, S. Ceria, and G. Cornuéjols. Mixed 0–1 programming by lift-and-project in a branch-and-cut framework. *Management Science*, 42:1229–1246, 1996.

E. Balas, S. Ceria, G. Cornuéjols, and N. Natraj. Gomory cuts revisited. *Operations Research Letters*, 19:1–9, 1996.

R. E. Bixby, S. Ceria, C. M. McZeal, and M. W. P. Savelsbergh. An updated mixed integer programming library: MIPLIB 3.0. *Optima*, 58:12–15, 1998. Problems available at *http://www.caam.rice.edu/bixby/miplib/miplib.html*.

A. Caprara and M. Fischetti. Branch and cut algorithms. In M. Dell'Amico, F. Maffioli, and S. Martello, editors, *Annotated Bibliographies in Combinatorial Optimization*, chapter 4. John Wiley, 1997.

S. Ceria, C. Cordier, H. Marchand, and L. A. Wolsey. Cutting plane algorithms for integer programs with general integer variables. *Mathematical Programming*, 81:201–214, 1998.

V. Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, 4:305–337, 1973.

C. Cordier, H. Marchand, R. Laundy, and L. A. Wolsey. bc-opt: A branch-and-cut code for mixed integer programs. Technical report, CORE, Université Catholique de Louvain, Louvain-la-Neuve, Belgium, October 1997.

H. P. Crowder, E. L. Johnson, and M. Padberg. Solving large-scale zero-one linear programming problems. *Operations Research*, 31:803–834, 1983.

R. E. Gomory. An algorithm for integer solutions to linear programs. In R. L. Graves and P. Wolfe, editors, *Recent Advances in Mathematical Programming*, pages 269–302. McGraw-Hill, New York, 1963.

M. Grötschel and O. Holland. Solution of large-scale travelling salesman problems. *Mathematical Programming*, 51(2):141–202, 1991.

M. Jünger, G. Reinelt, and S. Thienel. Practical problem solving with cutting plane algorithms in combinatorial optimization. In *Combinatorial Optimization: DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 20, pages 111–152. AMS, 1995.

M. Jünger and S. Thienel. Introduction to ABACUS — A Branch-And-CUt System. *Operations Research Letters*, 22:83–95, 1998.

J. E. Mitchell, P. P. Pardalos, and M. G. C. Resende. Interior point methods for combinatorial optimization. In D.-Z. Du and P. Pardalos, editors, *Handbook of Combinatorial Optimization*, volume 1, pages 189–297. Kluwer Academic Publishers, 1998.

G. L. Nemhauser, M. W. P. Savelsbergh, and G. C. Sigismondi. MINTO, a Mixed INTeger Optimizer. *Operations Research Letters*, 15:47–58, 1994.

G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley, New York, 1988.

M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33(1):60–100, 1991.

A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley, Chichester, 1986.

L. A. Wolsey. *Integer Programming*. John Wiley, New York, 1998.