BRANCH-AND-BOUND METHODS FOR INTEGER PROGRAMMING, *Branch-and-Bound*

Overview.

An *integer programming problem* (IP) is an optimization problem in which some or all of the variables are restricted to take on only integer values. The exposition presented here will focus on the case in which the objective and constraints of the optimization problem are defined via linear functions. In addition, for simplicity, it will be assumed that all of the variables are restricted to be nonnegative integer valued. Thus, the mathematical formulation of the problem under consideration can be stated as:

$$\begin{array}{lll} \text{maximize} & c^T x & \\ \text{subject to} & Ax \le b & \\ & x \in Z_+^n & \end{array} \qquad \text{(IP)}$$

where $A \in \Re^{m \times n}$, $b \in \Re^m$ and $c \in \Re^n$. For notational convenience, let S denote the constraint set of problem (IP); i.e.,

$$S := \{x \in Z_+^n : Ax \le b\}.$$

The classical approach to solving integer programs is branch-and-bound [41]. The branch-and-bound method is based on the idea of iteratively partitioning the set S (branching) to form subproblems of the original integer program. Each subproblem is solved — either exactly or approximately — to obtain an upper bound on the subproblem objective value. The driving force behind the branch-and-bound approach lies in the fact that if an upper bound for the objective value of a given subproblem is less than the objective value of a known integer feasible solution (e.g., obtained by solving some other subproblem) then the optimal solution of the original integer program cannot lie in the subset of S associated with the given subproblem. Hence, the upper bounds on subproblem objective values are, in essence, used to construct a proof of optimality without exhaustive search.

integer programming problem
linear programming relaxation
incumbent objective value

One concept that is fundamental to obtaining upper bounds on subproblem objective values is that of problem relaxation. A *relaxation* of the optimization problem

$$\max\{c^T x : x \in S\}$$

is an optimization problem

$$\max\{c_R^T x : x \in S_R\},$$

where $S \subseteq S_R$ and $c^T x \le c_R^T x$ for all $x \in S$. Clearly, solving a problem relaxation provides an upper bound on the objective value of the underlying problem. Perhaps the most common relaxation of problem (IP) is the *linear programming relaxation* formed by relaxing the integer restrictions and enforcing appropriate bound conditions on the variables; i.e., $c_R = c$ and $S_R = \{x \in \Re^n : Ax \le b,\ l \le x \le u\}$.

A formal statement of a general branch-and-bound algorithm [50] is presented in Figure 1. The notation L is used to denote the list of active subproblems $\{\text{IP}^i\}$, where $\text{IP}^0 = \text{IP}$ denotes the original integer program. The notation $\bar{z}_i$ denotes an upper bound on the optimal objective value of IP^i, and $\underline{z}_{ip}$ denotes the *incumbent objective value* (i.e., the objective value corresponding to the current best integral feasible solution to IP).

Figure 1.
General Branch-and-Bound Algorithm

1. (*Initialization*): Set $L = \{\text{IP}^0\}, \bar{z}_0 = +\infty$, and $\underline{z}_{ip} = -\infty$.
2. (*Termination*): If $L = \emptyset$, then the solution x^* which yielded the incumbent objective value $\underline{z}_{ip}$ is optimal. If no such x^* exists (i.e., $\underline{z}_{ip} = -\infty$) then IP is infeasible.
3. (*Problem selection and relaxation*): Select and delete a problem IP^i from L. Solve a relaxation of IP^i. Let z_i^R denote the optimal objective value of the relaxation, and let x^{iR} be an optimal solution if one exists. (Thus, $z_i^R = c^T x^{iR}$, or $z_i^R = -\infty$.)
4. (*Fathoming and Pruning*):
   (a) If $z_i^R \le \underline{z}_{ip}$ go to Step 2.

   (b) If $z_i^R > \underline{z}_{ip}$ and x^{iR} is integral feasible, update $\underline{z}_{ip} = z_i^R$. Delete from L all problems with $\bar{z}_i \leq \underline{z}_{ip}$. Go to Step 2.

5. (*Partitioning)*: Let $\{S^{ij}\}_{j=1}^{j=k}$ be a partition of the constraint set S^i of problem IP^i. Add problems $\{\text{IP}^{ij}\}_{j=1}^{j=k}$ to L, where IP^{ij} is IP^i with feasible region restricted to S^{ij} and $\bar{z}_{ij} = z_i^R$ for $j = 1, \ldots, k$. Go to Step 2.

The actual implementation of a branch-and-bound algorithm is typically viewed as a *tree search*, where the problem at the root node of the tree is the original IP. The tree is constructed in an iterative fashion with new nodes formed by *branching* on an existing node for which the optimal solution of the relaxation is fractional (i.e., some of the integer restricted variables have fractional values). Typically, two child nodes are formed by selecting a fractional valued variable and adding appropriate constraints in each child subproblem to ensure that the associated constraint sets do not include solutions for which this chosen branching variable assumes the same fractional value.

The phrase *fathoming a node* is used in reference to criteria that imply that a node need not be explored further. As indicated in Step 4, these criteria include:

(a) the objective value of the subproblem relaxation at the node is less than or equal to the incumbent objective value; and
(b) the solution for the subproblem relaxation is integer valued.

Note that (a) includes the case when the relaxation is infeasible, since in that case its objective value is $-\infty$. Condition (b) provides an opportunity to *prune* the tree; effectively fathoming nodes for which the objective value of the relaxation is less than or equal to the updated incumbent objective value. The tree search ends when all nodes are fathomed.

A variety of strategies have been proposed for intelligently selecting branching variables, for problem partitioning, and for selecting nodes to process. However, no single collection of strategies stands out as being best in all cases. In the remainder of this article, some of the strategies that have been implemented or proposed are summarized. An illustrative example is presented. Some of the related computational strategies – *preprocessing and reformulation*, *heuristic procedures*, and the concept of *reduced-cost fixing* – which have proved to be highly effective in branch-and-bound implementations are considered. Finally, there is a discussion of recent linear programming based branch-and-bound algorithms that have employed *interior-point methods* for the subproblem relaxation solver, which is in contrast to using the more traditional simplex-based solvers.

Though branch-and-bound is a classic approach for solving integer programs, there are practical limitations to its success in applications. Often integer feasible solutions are not readily available, and node pruning becomes impossible. In this case, branch-and-bound fails to find an optimal solution due to memory explosion as a result of excessive accumulation of active nodes. In fact, general integer programs are *NP-hard*; and consequently, as of this writing, there exists no known *polynomial-time algorithm* for solving general integer programs [30].

In 1983, a breakthrough in the computational possibilities of branch-and-bound came as a result of the research by Crowder, Johnson, and Padberg. In their paper [22], cutting planes were

tree search
branching
fathoming a node
prune
preprocessing and reformulation
heuristic procedures
reduced-cost fixing
interior-point methods
NP-hard
polynomial-time algorithm

added at the root node to strengthen the LP formulation before branch-and-bound was called. In addition, features such as reduced-cost fixing, heuristics and preprocessing were added within the tree search algorithm to facilitate the solution process. Readers are referred to the article on **cutting planes** in this encyclopedia for details on cutting plane applications to integer programming.

Most commercial integer programming solvers use a branch-and-bound algorithm with linear programming relaxations. Unless otherwise mentioned, the descriptions of the strategies discussed herein are based on using the linear programming relaxation.

Readers are referred to Nemhauser and Wolsey's textbook on Integer and Combinatorial Optimization [50] for other references not included in this paper. The text [53] also includes useful material about branch-and-bound.

Partitioning Strategies.

When linear programming relaxation is employed, partitioning is done via addition of linear constraints. Typically, two new nodes are formed on each division. Suppose x^R is an optimal solution to the relaxation of a branch-and-bound node. Common partitioning strategies include:

- *Variable Dichotomy* [23]. If x_j^R is fractional, then two new nodes are created, one with the simple bound $x_j \leq \lfloor x_j^R \rfloor$ and the other with $x_j \geq \lceil x_j^R \rceil$; where $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ denote the floor and the ceiling of a real number. In particular, if x_j is restricted to be binary, then the branching reduces to fixing $x_j = 0$ and $x_j = 1$, respectively. One advantage of simple bounds is that they maintain the size of the basis among branch-and-bound nodes, since the simplex method can be implemented to handle both upper and lower bounds on variables without explicitly increasing the dimensions of the basis.
- *Generalized-Upper-Bound (GUB) Dichotomy* [8]. If the constraint $\sum_{j \in Q} x_j = 1$ is present in the original integer program, and $x_i^R, i \in Q$ are fractional, one can partition $Q = Q_1 \cup Q_2$ such that $\sum_{j \in Q_1} x_j^R$ and $\sum_{j \in Q_2} x_j^R$ are approximately of equal value. Then two branches can be formed by setting $\sum_{j \in Q_1} x_j = 0$, and $\sum_{j \in Q_2} x_j = 0$ respectively.
- *Multiple branches for bounded integer variable.* If x_j^R is fractional, and $x_j \in \{0, \ldots, l\}$, then one can create $l + 1$ new nodes, with $x_j = k$ for node k, $k = 0, \ldots, l$. This idea was proposed in the first branch-and-bound algorithm by Land and Doig [41], but currently is not commonly used.

Branching Variable Selection.

During the partitioning process, branching variables must be selected to help create the children nodes. Clearly the choice of a branching variable affects the running time of the algorithm. Many different approaches have been developed and tested on different types of integer programs. Some common approaches are listed below:

- *Most/Least Infeasible Integer Variable.* In this approach, the integer variable whose fractional value is farthest/closest from/to an integral value is chosen as the branching variable.
- *Driebeck-Tomlin Penalties* [25, 59]. Penalties give a lower bound on the degradation of the objective value for branching each direction from a given variable. The penalties are the cost of the dual pivot needed to remove the fractional variable from the basis. If many pivots are required to restore primal feasibility, these penalties are not very informative. The *up penalty*, when forcing the value of the kth basic variable up, is

$$u_k = \min_{j: a_{kj} < 0} \frac{(1 - f_k)\bar{c}_j}{-a_{kj}}$$

where f_k is the fractional part of x_k, $\bar{c}_j$ is the reduced cost of variable x_j, and the a_{kj} are the transformed matrix coefficients

cutting planes
Variable Dichotomy
Generalized-Upper-Bound (GUB) Dichotomy
Driebeck-Tomlin Penalties

from the kth row of the optimal dictionary for the LP relaxation. The *down penalty* d_k is calculated as

$$d_k = \min_{j:a_{kj}>0} \frac{f_k \bar{c}_j}{a_{kj}}.$$

Once the penalties have been computed, a variety of rules can be used to select the branching variable (e.g., $\max_k \max(u_k, d_k)$, or $\max_k \min(u_k, d_k)$). A penalty can be used to eliminate a branch if the LP objective value for the parent node minus the penalty is worse than the incumbent integer solution. Penalties are out of favor because their cost is considered too high for their benefit.

- *Pseudo-Cost Estimate*. Pseudo-costs provide a way to estimate the degradation to the objective value by forcing a fractional variable to an integral value. The technique was introduced in 1970 by Benichou *et al* [10]. They attempt to reflect the total cost, not just the cost of the first pivot, as with penalties. Once a variable x_k is labeled as a candidate branching variable, the pseudo-costs are computed as:

$$U_k = \frac{\bar{z}_k - z_k^u}{1-f_k}, \text{ and } D_k = \frac{\bar{z}_k - z_k^d}{f_k}$$

where $\bar{z}_k$ is the objective value of the parent, z_k^u is the objective value resulting from forcing up, and z_k^d is the objective value from forcing down. (If the subproblem is infeasible, the associated pseudo-cost is not calculated.) If a variable has been branched upon repeatedly, an average may be used.

The branching variable is chosen as that with the maximum degradation, where the degradation is computed as: $D_k f_k + U_k(1-f_k)$. Pseudo-costs are not considered to be beneficial on problems where there is a large percentage of integer variables.

- *Pseudo-Shadow Prices*. Similar to pseudo-costs, pseudo-shadow prices estimate the total cost to force a variable to an integral value. Up and down pseudo-shadow prices for each constraint and pseudo-shadow prices for each integer variable are specified by the user or given an initial value. The degradation in the objective function for forcing an integer variable x_k up or down to an integral value can be estimated. The branching variable is chosen using criteria similar to penalties and pseudo-costs. See [27, 42] for precise mathematical formulations on this approach.
- *Strong Branching*. This branching strategy arose in connection with research on solving difficult instances of the traveling salesman problem and general mixed 0/1 integer programming problems [2, 12, 13]. Applied to 0/1 integer programs within a simplex-based branch-and-cut setting, strong branching works as follows. Let N and K be positive integers. Given the solution of some linear programming relaxation, make a list of N binary variables that are fractional and closest to 0.5 (if there are fewer than N fractional variables, take all fractional variables). Suppose that I is the index set of this list. Then, for each $i \in I$, fix x_i first to 0 and then to 1 and perform K iterations (starting with the *optimal basis* for the LP relaxation of the current node) of the dual simplex method with steepest-edge pricing. Let L_i, U_i, $i \in I$, be the objective values that result from these simplex runs, where L_i corresponds to fixing x_i to 0 and U_i to fixing it to 1. A branching variable can be selected based on the best weighted-sum of these two values.
- *Priorities Selection*. Variables are selected based on their priorities. Priorities can be user-assigned, or based on objective function coefficients, or on pseudo-costs.

Node Selection.

Given a list of active problems, one has to decide which subproblem should be selected to be examined next. This in turn will affect the

Pseudo-Cost Estimate
Pseudo-Shadow Prices
Strong Branching
optimal basis

possibilities of improving the incumbent, the chance of node fathoming, and the total number of problems needed to be solved before optimality is achieved. Below, various strategies given in [7, 10, 11, 20, 27, 29, 31, 35, 49] are presented.

- *Depth-First-Search with Backtracking.* Choose a child of the previous node as the next node; if it is pruned, choose the other child. If this node is also pruned, choose the most recently created unexplored node, which will be the other child node of the last successful node.
- *Best-Bound.* Among all unexplored nodes, choose the one which has the best LP objective value. In the case of maximization, the node with the largest LP objective value will be chosen. The rationale is that since nodes can only be pruned when the relaxation objective value is less than the current incumbent objective value, the node with largest LP objective value cannot be pruned, since the best objective value corresponding to an integer feasible solution cannot exceed this largest value.
- *Sum of Integer Infeasibilities.* The sum of infeasibilities at a node is calculated as
$$s = \sum_j \min(f_j, 1 - f_j).$$
Choose the node with either maximum or minimum sum of integer infeasibilities.
- *Best-Estimate using Pseudo-Costs.* This technique was introduced [10] along with the idea of using pseudo-costs to select a branching variable. The individual pseudo-costs can be used to estimate the resulting integer objective value attainable from node k:
$$\epsilon_k = \bar{z}_k - \sum_i \min(D_i f_i, U_i(1 - f_i))$$
where $\bar{z}_k$ is the value of the LP relaxation at node k. The node with the best estimate is chosen.
- *Best-Estimate using Pseudo-Shadow Prices.* Pseudo-shadow prices can also be used to provide an estimate of the resulting integer objective value attainable from the node, and the node with the best estimate can then be chosen.
- *Best Projection* [35, 49]. Choose the node among all unexplored nodes which has the best projection. The projection is an estimate of the objective function value associated with an integer solution obtained by following the subtree starting at this node. It takes into account both the current objective function value and a measure of the integer infeasibility. In particular, the projection p_k associated with node k is defined as
$$p_k = \bar{z}_k - \frac{s_k(\bar{z}_0 - z_{ip})}{s_0},$$
where $\bar{z}_0$ denotes the objective value of the LP at the root node, z_{ip} denotes an estimate of the optimal integer solution, and s_k denotes the sum of the integer infeasibilities at node k. The projection is a weighting between the objective function and the sum of infeasibilities. The weight $(\bar{z}_0 - z_{ip})/s_0$ corresponds to the slope of the line between node 0 and the node producing the optimal integer solution. It can be thought of as the cost to remove one unit of infeasibility. Let n_k be the number of integer infeasibilities at node k. A more general projection formula is to let $w_k = \mu n_k + (1 - \mu)s_k$, where $\mu \in [0, 1]$, and define
$$p_k = \bar{z}_k - \frac{w_k(\bar{z}_0 - z_{ip})}{w_0}.$$

Illustrative Example.

In this section, a two-variable integer program is solved using branch-and-bound. The most infeasible integer variable is used as the branching

Depth-First-Search with Backtracking
Best-Bound
Best Projection

variable, and best-bound is used for node selection. Consider the problem

$$\begin{array}{lrcrcrl} \text{maximize} & 13x_1 & + & 8x_2 & & & \\ \text{subject to} & x_1 & + & 2x_2 & \leq & 10 & \\ & 5x_1 & + & 2x_2 & \leq & 20 & \qquad (\mathrm{IP}^0) \\ & \multicolumn{5}{l}{x_1 \geq 0,\ x_2 \geq 0} \\ & \multicolumn{5}{l}{x_1,\ x_2 \text{ integer}} \end{array}$$

Initially, L consists of just this problem IP^0. The solution to the LP relaxation is $x_1^0 = 2.5$, $x_2^0 = 3.75$, with value $z_0^R = 59.5$. The most infeasible integer variable is x_1, so two new subproblems are created, IP^1 where $x_1 \geq 3$ and IP^2 where $x_1 \leq 2$, and $L = \{\mathrm{IP}^1, \mathrm{IP}^2\}$.

Both problems in L have the same bound 59.5, so assume the algorithm arbitrarily selects IP^1. The optimal solution to the LP relaxation of IP^1 is $x_1^1 = 3$, $x_2^1 = 2.5$, with value $z_1^R = 59$. The most infeasible integer variable is x_2, so two new subproblems of IP^1 are created, IP^3 where $x_2 \geq 3$ and IP^4 where $x_2 \leq 2$, and now $L = \{\mathrm{IP}^2, \mathrm{IP}^3, \mathrm{IP}^4\}$.

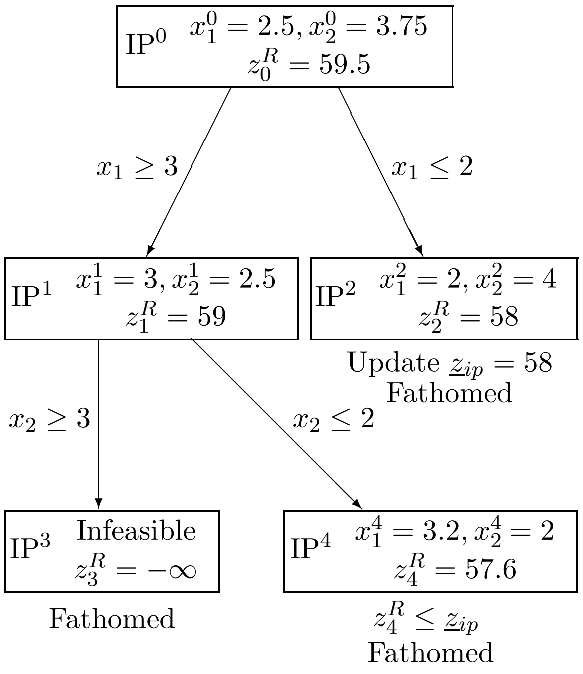
The algorithm next examines IP^2, since this is the problem with the best bound. The optimal solution to the LP-relaxation is $x_1^2 = 2$, $x_2^2 = 4$, with value $z_2^R = 58$. Since x^2 is integral feasible, $\underline{z}_{ip}$ can be updated to 58 and IP^2 is fathomed.

Both of the two problems remaining in L have best bound greater than 58, so neither can yet be fathomed. Since these two subproblems have the same bound 59, assume the algorithm arbitrarily selects IP^3 to examine next. The LP relaxation to this problem is infeasible, since it requires that x satisfy $x_1 \geq 3$, $x_2 \geq 3$ and $5x_1 + 2x_2 \leq 20$ simultaneously. Therefore, $z_3^R = -\infty$, and this node can be fathomed by bounds since $z_3^R \leq \underline{z}_{ip}$.

That leaves the single problem IP^4 in L. The solution to the LP relaxation of this problem is $x_1^4 = 3.2$, $x_2^4 = 2$, with value $z_4^R = 57.6$. Since $z_4^R \leq \underline{z}_{ip}$, this subproblem can also be fathomed by bounds. The set L is now empty, so x^2 is optimal for the integer programming problem IP^0.

The progress of the algorithm is indicated in Figure 2. Each box contains the name of the subproblem, the solution to the LP relaxation, and the value of the solution.

Figure 2. Branch and bound example


IP^0 $x_1^0 = 2.5, x_2^0 = 3.75$ $z_0^R = 59.5$
$x_1 \geq 3$
$x_1 \leq 2$
IP^1 $x_1^1 = 3, x_2^1 = 2.5$ $z_1^R = 59$
IP^2 $x_1^2 = 2, x_2^2 = 4$ $z_2^R = 58$
Update $\underline{z}_{ip} = 58$
Fathomed
$x_2 \geq 3$
$x_2 \leq 2$
IP^3 Infeasible $z_3^R = -\infty$
Fathomed
IP^4 $x_1^4 = 3.2, x_2^4 = 2$ $z_4^R = 57.6$
$z_4^R \leq \underline{z}_{ip}$
Fathomed

Preprocessing and Reformulation.

Problem preprocessing and reformulation has been shown to be a very effective way of improving integer programming formulations prior to and during branch-and-bound [14, 15, 18, 19, 22, 24, 34, 36, 37, 56]. Below, some commonly employed preprocessing techniques are listed. For more details on these procedures, see the references.

1. Removal of empty (all zeros) rows and columns. Detection of implicit bounds and implicit slack variables.
2. Removal of rows dominated by multiples of other rows, including pairs of rows for which the support of one is a subset of the support of the other.
3. Strengthen the bounds within rows by comparing individual variables and coefficients to the right-hand-side. Additional strengthening may be possible for integral variables using rounding.
4. Use variable bounds to determine upper and lower bounds for the left-hand side of a constraint, and compare these bounds to

the right-hand side. Where possible conclude that a constraint is inconsistent, redundant, or forces the fixing of some or all variables in its support. Several of these row-driven operations can be dualized to columns.

5. *Aggregation*: Given an equality constraint where the bound on some variable is implied by the satisfaction of the bounds on the other variables, this variable can be substituted out, and the constraint deleted. Note that free variables always satisfy this condition. Note also that in order to control fill-in (and coefficient growth), not all such substitutions may be desirable. For integral variables, there is the added restriction that they can be eliminated only if their integrality is implied by the integrality of the remaining variables. For integer programming problems, an added advantage of aggregation relative to LP's, is that the reduction in the number of equality constraints increases the relative dimension of the underlying polytope.
6. *Coefficient reduction*: Consider a constraint $\sum_{j \in K} a_j x_j \geq b$ in which all $a_j \geq 0$ and all $x_j \geq 0$. If x_j is a 0/1 variable and $a_j > b$, for some $j \in K$, replace a_j by b. A stronger version of this procedure is possible when the problem formulation involves other constraints of appropriate structure.
7. Logical implications and probing:
   a. *Logical implications*: Choose a binary variable x_k and fix it to 0 or 1. Perform 4. This analysis may yield logical implications such as $x_k = 1$ implies $x_j = 0$, or $x_k = 1$ implies $x_j = 1$, for some other variable x_j. The implied equality is then added as an explicit constraint.
   b. *Probing*: Perform logical implications recursively. An efficient implementation of probing appears to be very difficult. Guignard and Spielberg [34], and Savelsbergh [56] discuss details of computational issues regarding probing.

Heuristics.

Heuristic procedures provide a means for obtaining integer feasible solutions quickly, and can be used repeatedly within the branch-and-bound search tree. A good heuristic — one that produces good integer feasible solutions — is a crucial component in the branch-and-bound algorithm since it provides an upper bound for reduced-cost fixing (see later) at the root, and thus allows reduction in the size of the linear program that must be solved. This in turn may reduce the time required to solve subsequent linear programs at nodes within the search tree. In addition, a good upper bound increases the likelihood of being able to fathom active nodes, which is extremely important when solving large-scale integer programs as they tend to create many active nodes leading to memory explosion.

Broadly speaking, five ideas are commonly used in developing heuristics. The first idea is that of greediness. Greedy algorithms work by successively choosing variables based on best improvement in the objective value. Kruskal's algorithm [39], which is an exact algorithm for finding the minimum-weight spanning tree in a graph, is one of the most well-known **greedy algorithms**. Greedy algorithms have been applied to a variety of problems, including 0/1 knapsack problems [38, 43, 55], uncapacitated facility location problems [40, 58], set covering problems [3, 4], and the traveling salesman problem [54].

A second idea is that of **local search**, which involves searching in a local neighborhood of a given integer feasible solution for a feasible solution with a better objective value. The k-interchange heuristic is a classic example of a

Aggregation
Coefficient reduction
Logical implications
Probing
greedy algorithms
local search

local search heuristic [40, 46, 48]. **Simulated annealing** is another example, but with a bit of a twist. It allows, with a certain probability, updated solutions with less favorable objective values in order to increase the likelihood of escaping from a local optimum [16].

Randomized Enumeration is a third idea that is used to obtain integer feasible solutions. One such method is that of **genetic algorithms**, where the randomness is modeled on the biological mechanisms of evolution and natural selection [33]. Recent work on applying a genetic algorithm to the set covering problem can be found in [9].

Simulated annealing and genetic algorithms are examples of *metaheuristics*: they find local mimima, but they also possess mechanisms for moving away from one local minimum to a better local minimum. Another widely used metaheuristic is **tabu search**. This process makes certain moves tabu, or forbidden, and then finds the best remaining move. More details on tabu search can be found in [32], and in the references contained therein.

The term *primal heuristics* refers to certain LP-based procedures for constructing integral feasible solutions from points that are in some sense good, but fail to satisfy integrality. Typically, these non-integral points are obtained as optimal solutions of LP relaxations. Primal heuristic procedures involve successive variable fixing and rounding (according to rules usually governed by problem structure) and subsequent resolves of the modified primal LP [6, 12, 14, 36, 37].

The fifth general principle is that of *exploiting the interplay between primal and dual solutions*. For example, an optimal or heuristic solution to the dual of an LP relaxation may be used to construct a heuristic solution for the primal IP. Problem dependent criteria based on the generated primal-dual pair may suggest seeking an alternative heuristic solution to the dual, which would then be used to construct a new heuristic solution to the primal. Iterating back-and-forth between primal and dual heuristic solutions would continue until an appropriate termination condition is satisfied [21, 26, 28].

It is not uncommon that a heuristic involves more than one of these ideas. For example, pivot-and-complement is a simplex-based heuristic in which binary variables in the basis are pivoted out and replaced by slack variables. When a feasible integer solution is obtained, the algorithm performs a local search in an attempt to obtain a better integer feasible solution [5]. Obviously, within a branch-and-bound implementation, the structure of the problems that the implementation is targeted at influences the design of an effective heuristic [2, 12, 13, 14, 22, 26, 36, 37, 45].

Continuous Reduced Cost Implications.

Reduced cost fixing is a well-known and important idea in the literature of integer programming [22]. Given an optimal solution to an LP relaxation, the reduced costs $\bar{c}_j$ are nonpositive for all nonbasic variables x_j at lower bound, and nonnegative for all nonbasic variables at their upper bounds. Let x_j be a nonbasic variable in a continuous optimal solution having objective value z_{LP}, and let $\underline{z}_{ip}$ be the objective value associated with an integer feasible solution to (IP). The following are true:

(a) If x_j is at its lower bound in the continuous solution and $z_{LP} - \underline{z}_{ip} \leq -\bar{c}_j$, then there exists an optimal solution to the integer program with x_j at its lower bound.

(b) If x_j is at its upper bound in the continuous solution and $z_{LP} - \underline{z}_{ip} \leq \bar{c}_j$, then there exists an optimal solution to the integer program with x_j at its upper bound.

When reduced-cost fixing is applied to the root node of a branch-and-bound tree, variables which are fixed can be removed from the problem, resulting in a reduction in the size of

Simulated annealing
genetic algorithms
metaheuristics
tabu search
primal heuristics
Reduced cost fixing

the integer program. A variety of studies have examined the effectiveness of reduced-cost fixing within the branch-and-bound tree search [12, 14, 22, 36, 37, 51, 52].

Subproblem Solver.

When linear programs are employed as the relaxations within a branch-and-bound algorithm, it is common to use a simplex-based algorithm to solve each subproblem, using *dual simplex* to reoptimize from the optimal basis of the parent node. This technique of *advanced basis* has been shown to reduce the number of simplex iterations to solve the child node to optimality, and thus speedup the overall computational effort. Recently with the advancement in computational technology, the increase in the size of integer programs, and the success of **interior point methods** to solve large-scale linear programs [1, 47] there are some branch-and-bound algorithms employing interior point algorithms as the linear programming solver [17, 44, 45, 57]. In this case, advanced basis is no longer available and care has to be taken to take advantage of warmstart vectors for the interior point solver so as to facilitate effective computational results. In [44, 45], a description of the ideas of "*advanced warmstart*" and computational results are presented.

References

[1] Andersen, E. D., Gondzio, J., Mészáros, C., and Xu, X.: 'Implementation of interior point methods for large scale linear programming', *Interior Point Methods in Mathematical Programming*, in T. Terlaky (ed.). Kluwer Academic Publishers, 1996, ch. 6.

[2] Applegate, D., Bixby, R.E., Chvátal, V., and Cook, W.: Finding cuts in the TSP (A preliminary report), Tech. Rep. 95-05, DIMACS, Rutgers University, New Brunswick, NJ 08903, 1995.

[3] Baker, E.K.: 'Efficient Heuristic Algorithms for the Weighted Set Covering Problem', *Computers and Operations Research* **8** (1981), 303–310.

[4] Baker, E.K., and Fisher, M.L.: 'Computational Results for Very Large Air Crew Scheduling Problems', *Omega* **19** (1981), 613–618.

[5] Balas, E., and Martin, C.H.: 'Pivot and Complement-A Heuristic for 0/1 Programming', *Management Science* **26** (1980), 86–96.

[6] Baldick, R.: A Randomized Heuristic for Inequality-Constrained Mixed-Integer Programming, Tech. rep., Department of Electrical and Computer Engineering, Worcester Polytechnic Institute, 1992.

[7] Beale, E.M.L.: 'Branch and Bound Methods for Mathematical Programming Systems', *Annals of Discrete Mathematics* **5** (1979), 201–219.

[8] Beale, E.M.L., and Tomlin, J.A.: 'Special Facilities in a General Mathematical Programming System for Nonconvex Problems Using Ordered Sets of Variables', *Proceedings of the Fifth International Conference on Operations Research, J. Lawerence, ed., Tavistock Publications* (1970), 447–454.

[9] Beasley, J.E., and Chu, P.C.: 'A Genetic Algorithm for the Set Covering Problem', *European Journal of Operations Research* **194** (1996), 392–404.

[10] Benichou, M., Gauthier, J.M., Girodet, P., Hehntges, G., Ribiere, G., and Vincent, O.: 'Experiments in Mixed Integer Linear Programming', *Mathematical Programming* **1** (1971), 76–94.

[11] Benichou, M., Gauthier, J.M., Hehntges, G., and Ribiere, G.: 'The efficient solution of large-scale linear programming problems - some algorithmic techniques and computational results', *Mathematical Programming* **13** (1977), 280–322.

[12] Bixby, R.E., Cook, W., Cox, A., and Lee, E. K.: Parallel Mixed Integer Programming, Tech. Rep. CRPC-TR95554, Center for Research on Parallel Computation, Rice University, Houston, Texas, 1995.

[13] Bixby, R.E., Cook, W., Cox, A., and Lee, E. K.: 'Computational experience with parallel mixed integer programming in a distributed environment', *Annals of Operations Research, Special Issue on Parallel Optimization* (1997).

[14] Bixby, R.E., and Lee, E.K.: Solving a Truck Dispatching Scheduling Problem Using Branch-and-Cut, Tech. Rep. Research Report TR93-37, Department of Computational and Applied Mathematics, Rice Univerity, Houston, Texas, 1993, to appear in *Operations Research*.

[15] Bixby, R.E., and Wagner, D.K.: 'A note on Detecting Simple Redundancies in Linear Systems', *Operations Research Letters* **6** (1987), 15–18.

[16] Bonomi, E., and Lutton, J.L.: 'The N-City Traveling Salesman Problem: Statistical Mechanics and the Metropolis Algorithm', *SIAM Review* **26** (1984), 551–568.

dual simplex
advanced basis
interior point methods
advanced warmstart

[17] Borchers, B., and Mitchell, J.E.: Using an interior point method in a branch and bound algorithm for integer programming, Tech. Rep. 195, Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY 12180, March 1991, Revised July 7, 1992.

[18] Bradley, G.H., Hammer, P.L., and Wolsey, L.: 'Coefficient Reduction in 0-1 Variables', *Mathematical Programming* **7** (1975), 263–282.

[19] Brearley, A.L., Mitre, G., and Williams, H. P.: 'Analysis of Mathematical programming problems prior to applying the simplex method', *Mathematical Programming* **5** (1975), 54–83.

[20] Breu, R., and Burdet, C.A.: 'Branch and Bound Experiments in Zero-One Programming', *Mathematical Programming* **2** (1974), 1–50.

[21] Conn, A.R., and Cornuejols, G.: A Projection Method for the Uncapacitated Facility Location Problem, Tech. Rep. 26-86-87, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA, 1987.

[22] Crowder, H., Johnson, E.L., and Padberg, M.: 'Solving Large-scale Zero-one Linear Programming Problem', *Operations Research* **31** (1983), 803–834.

[23] Dakin, R.J.: 'A Tree Search Algorithm for Mixed Integer Programming Problems', *Computer Journal* **8** (1965), 250–255.

[24] Dietrich, B., and Escudero, L.: 'Coefficient Reduction for Knapsack-like constraints in 0/1 programs with variable upper bounds', *Operations Research Letters* **9** (1990), 9–14.

[25] Driebeek, N.J.: 'An Algorithm for the Solution of Mixed Integer Programming Problems', *Management Science* **21** (1966), 576–587.

[26] Erlenkotter, D.: 'A Dual-Based Procedure for Uncapacitated Facility Location', *Operations Research* **26** (1978), 992–1009.

[27] Fenelon, M.: 'Branching Strategies for MIP', *CPLEX* (1991).

[28] Fisher, M.L., and Jaikumer, R.: 'A Generalized Assignment Heuristic for Vehicle Routing', *Networks* **11** (1981), 109–124.

[29] Forrest, J.J., Hirst, J.P.H., and Tomlin, J.A.: 'Practical solution of large mixed integer programming problems with UMPIRE', *Management Science* **20** (1974), 736–773.

[30] Garey, M.R., and Johnson, D.S.: *Computers and Intractability - A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, Oxford, England, 1979.

[31] Gauthier, J.M., and Ribiere, G.: 'Experiments in Mixed Integer Programming Using Pseudo-Costs', *Mathematical Programming* **12** (1977), 26–47.

[32] Glover, F., and Laguna, M.: *Tabu Search*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1997.

[33] Goldberg, D.E.: *Genetic algorithms in search, optimization, and machine learning*, Addison-Wesley, 1989.

[34] Guignard, M., and Spielberg, K.: 'Logical reduction methods in zero-one programming', *Operations Research* **29** (1981), 49–74.

[35] Hirst, J.P.H.: 'Features required in branch and bound algorithms for (0-1) mixed integer programming', *Privately circulated manuscript* (1969).

[36] Hoffman, K.L., and Padberg, M.: 'Improving LP-representations of zero-one linear programs for branch-and-cut', *ORSA Journal on Computing* **3** (1991), 121–134.

[37] Hoffman, K.L., and Padberg, M.: 'Solving airline crew-scheduling problems by branch-and-cut', *Management Science* **39** (1992), 657–682.

[38] Ibarra, O.H., and Kim, C.E.: 'Fast Approximation Algorithms for the Knapsack and Sum of Subset Problems', *Journal of the Association for Computing Machinery* **22** (1975), 463–468.

[39] Kruskal, J.B.: 'On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem', *Proceedings of the American Mathematical Society* **7** (1956), 48–50.

[40] Kuehn, A.A., and Hamburger, M.J.: 'A Heuristic Program for Locating Warehouses', *Management Science* **19** (1963), 643–666.

[41] Land, A.H., and Doig, A.G.: 'An Automatic Method for Solving Discrete Programming Problems', *Econometrica* **28** (1960), 497–520.

[42] Land, A.H., and Powell, S.: 'Computer codes for problems of integer programming', *Annals of Discrete Mathematics* **5** (1979), 221–269.

[43] Lawler, E.L.: 'Fast Approximation Algorithms for the Knapsack Problems', *Mathematics of Operations Research* **4** (1979), 339–356.

[44] Lee, E.K., and Mitchell, J.E.: 'Computational Experience in Nonlinear Mixed Integer Programming': *The Operations Research Proceedings 1996*, Springer-Verlag, 1996, pp. 95–100.

[45] Lee, E.K., and Mitchell, J.E.: 'Computational Experience of an Interior-Point SQP Algorithm in a Parallel Branch-and-Bound Framework': *Proceedings of High Performance Optimization Techniques 1997*, Springer-Verlag, 1997, Technical Report LEC 97-08, Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia.

[46] Lin, S., and Kernighan, B.W.: 'An Effective Heuristic Algorithm for the Traveling Salesman Problem', *Operations Research* **21** (1973), 498–516.

[47] Lustig, I. J., Marsten, R. E., and Shanno, D. F.: 'Interior Point Methods for Linear Programming: Computational State of the Art', *ORSA Journal on Computing* **6(1)** (1994), 1–14, see also the following commentaries and rejoinder.

[48] Manne, A.S.: 'Plant Location underEconomies of Scale-Decentralization and Computation', *Management Science* **11** (1964), 213–235.

[49] Mitra, G.: 'Investigations of some Branch and Bound Strategies for the Solution of Mixed Integer Linear Programs', *Mathematical Programming* **4** (1973), 155–170.

[50] Nemhauser, G.L., and Wolsey, L.A.: *Integer and Combinatorial Optimization*, Wiley, New York, 1988.

[51] Padberg, M., and Rinaldi, G.: 'A branch-and-cut approach to a traveling salesman problem with side constraints', *Management Science* **35** (1989), 1393–1412.

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

[53] Parker, R.G., and Rardin, R.L.: *Discrete Optimization*, Academic Press, San Diego, 1988.

[54] Rosenkrantz, D.J., Stearns, R.E., and Lewis, P.M.: 'An Analysis of Several Heuristics for the Traveling Salesman Problem', *SIAM Journal on Computing* **6** (1977), 563–581.

[55] Sahni, S.: 'Approximate Algorithms for the 0-1 Knapsack Problem', *Journal of Association for Computing Machinery* **22** (1975), 115–124.

[56] Savelsbergh, M. W. P.: 'Preprocessing and probing for mixed integer programming problems', *ORSA Journal on Computing* **6** (1994), 445–454.

[57] Silva, A.de, and Abramson, D.: A parallel interior point method and its application to facility location problems, Tech. rep., School of Computing and Information Technology, Griffith University, Nathan, QLD 4111, Australia, 1995, to appear in *Computational Optimization and Applications*.

[58] Spielberg, K.: 'Algorithms for the Simple Plant Location Problem with some Side-Conditions', *Operations Research* **17** (1969), 85–111.

[59] Tomlin, J.A.: 'An Improved Branch and Bound Method for Integer Programming', *Operations Research* **19** (1971), 1070–1075.

Eva K. Lee
Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta GA
USA
E-mail address: evakylee@isye.gatech.edu

John E. Mitchell
Mathematical Sciences
Rensselaer Polytechnic Institute
Troy NY
USA
E-mail address: mitchj@rpi.edu

AMS 1991 Subject Classification: 90C10, 49M35, 90C11, 90C06.
Key words and phrases: branch-and-bound, integer programs, exact algorithms.