

BRANCH-AND-PRICE: INTEGER PROGRAMMING WITH COLUMN GENERATION, *BP*

Branch-and-price is a generalization of *linear programming* (LP) based *branch-and-bound* specifically designed to handle *integer programming* (IP) formulations that contain a huge number of variables. The basic idea of branch-and-price is simple. Columns are left out of the *LP relaxation* because there are too many columns to handle efficiently and most of them will have their associated variable equal to zero in an optimal solution anyway. Then to check the optimality of an LP solution, a subproblem, called the *pricing problem*, is solved to try to identify columns with a profitable *reduced cost*. If such columns are found, the LP is reoptimized. Branching occurs when no profitable columns are found, but the LP solution does not satisfy the integrality conditions. Branch-and-price applies *column generation* at every node of the branch-and-bound tree.

There are several reasons for considering IP formulations with a huge number of variables.

- A compact formulation of an IP may have a weak LP relaxation. Frequently the relaxation can be tightened by a reformulation that involves a huge number of variables.
- A compact formulation of an IP may have a symmetric structure that causes branch-and-bound to perform poorly because the problem barely changes after branching. A reformulation with a huge number of variables can eliminate this symmetry.
- Column generation provides a *decomposition* of the problem into master and subproblems. This decomposition may have a natural interpretation in the contextual

linear programming

branch-and-bound

integer programming

LP relaxation

pricing problem

reduced cost

column generation

decomposition

generalized assignment problem

setting allowing for the incorporation of additional important constraints or nonlinear cost functions.

- A formulation with a huge number of variables may be the only choice.

At first glance, it may seem that branch-and-price involves nothing more than combining well-known ideas for solving linear programs by column generation with branch-and-bound. However, it is not that straightforward. There are fundamental difficulties in applying column generation techniques for linear programming in integer programming solution methods. These include:

- Conventional integer programming branching on variables may not be effective because fixing variables can destroy the structure of the pricing problem.
- Column generation often converges slowly and solving the LPs to optimality may become computationally prohibitive.

We illustrate the concepts of branch-and-price and the difficulties that may arise by means of an example.

In the *generalized assignment problem* (GAP) the objective is to find a maximum profit assignment of m tasks to n machines such that each task is assigned to precisely one machine subject to capacity restrictions on the machines. For reasons that will become apparent later, we will consider separately the two cases of nonidentical and identical machines.

Nonidentical Machines

The natural integer programming formulation of GAP is

$$\max \sum_{1 \leq i \leq m} \sum_{1 \leq j \leq n} p_{ij} z_{ij}$$

subject to

$$\begin{aligned} \sum_{1 \leq j \leq n} z_{ij} &= 1 & i = 1, \dots, m, \\ \sum_{1 \leq i \leq m} w_{ij} z_{ij} &\leq d_j & j = 1, \dots, n, \\ z_{ij} &\in \{0, 1\} & i = 1, \dots, m, j = 1, \dots, n, \end{aligned}$$

where p_{ij} is the profit associated with assigning task i to machine j , w_{ij} is the amount of the capacity of machine j used by task i , d_j is the capacity of machine j , and z_{ij} is a 0-1 variable indicating whether task i is assigned to machine j .

An alternative formulation of GAP in terms of columns representing feasible assignments of tasks to machines is

$$\max \sum_{j \leq 1 \leq n} \sum_{1 \leq k \leq K_j} \left(\sum_{1 \leq i \leq m} p_{ij} y_{ik}^j \right) \lambda_k^j$$

subject to

$$\begin{aligned} \sum_{1 \leq j \leq n} \sum_{1 \leq k \leq K_j} y_{ik}^j \lambda_k^j &= 1 & i = 1, \dots, m, \\ \sum_{1 \leq k \leq K_j} \lambda_k^j &= 1 & j = 1, \dots, n, \\ \lambda_k^j &\in \{0, 1\} & j = 1, \dots, n, k = 1, \dots, K_j, \end{aligned}$$

where the first m entries of a column, given by $y_k^j = (y_{1k}^j, y_{2k}^j, \dots, y_{mk}^j)$, satisfy the knapsack constraint

$$\begin{aligned} \sum_{1 \leq i \leq m} w_{ij} x_i &\leq d_j, \\ x_i &\in \{0, 1\} & i = 1, \dots, m, \end{aligned}$$

and where K_j denotes the number of feasible solutions to the above knapsack constraint. The first set of constraints ensures that each task is assigned to a machine, and the second set of constraints, the convexity constraints, ensures that exactly one feasible assignment of tasks to machines is selected for each machine. This is in fact the formulation that is obtained when we apply *Dantzig-Wolfe decomposition* to the natural formulation of GAP with the assignment constraints defining the master problem and the

Dantzig-Wolfe decomposition
convex combinations
simplex method
dual price

machine capacity constraints defining the sub-problems.

The reason for considering this alternative formulation of GAP is that the LP relaxation of the master problem is tighter than the LP relaxation of the natural formulation because certain fractional solutions are eliminated. Namely, all fractional solutions that are not *convex combinations* of 0-1 solutions to the knapsack constraints.

Unfortunately, the LP relaxation of the master problem cannot be solved directly due to the exponential number of columns. However, the LP relaxation of a restricted version of the master problem that considers only a subset of the columns can be solved directly using, for instance, the *simplex method*. Furthermore, if the reduced costs of all the columns that were left out are nonnegative, then the LP solution obtained is also optimal for the LP relaxation of the unrestricted master problem. To check whether there exist a column with positive reduced cost we solve the pricing problem

$$\max_{1 \leq j \leq n} \{z(KP_j) - v_j\},$$

where v_j is the optimal *dual price* from the solution to the LP relaxation of the restricted master problem associated with the convexity constraint of machine j , and $z(KP_j)$ is the value of the optimal solution to the knapsack problem

$$\max \sum_{1 \leq i \leq n} (p_{ij} - u_i) x_i$$

subject to

$$\sum_{1 \leq i \leq n} w_{ij} x_i \leq d_j$$

$$x_i^j \in \{0, 1\} \quad i \in \{1, \dots, n\}$$

with u_i being the optimal dual price from the solution to the LP relaxation of the restricted master problem associated with the assignment constraint of task i . If the optimal value of the pricing problem is positive, we have identified a column with positive reduced cost. In that case,

we add the column to the restricted master problem and reoptimize.

The LP relaxation of the master problem solved by column generation may not have an integral optimal solution and applying a standard branch-and-bound procedure to the master problem over the existing columns is unlikely to find an optimal, or good, or even feasible solution to the original problem. Therefore it may be necessary to generate additional columns in order to solve the linear programming relaxations of the master problem at non-root nodes of the search tree.

Standard branching on the λ -variables creates a problem along a branch where a variable has been set to zero. Recall that y_k^j represents a particular solution to the j th knapsack problem. Thus $\lambda_k^j = 0$ means that this solution is excluded. However, it is possible (and quite likely) that the next time the knapsack problem for the j th machine is solved the optimal solution is precisely the one represented by y_k^j . In that case, it would be necessary to find the second best solution to the knapsack problem. At depth l in the branch-and-bound tree we may need to find the l^{th} best solution, which is very hard. Fortunately, there is a simple remedy to this difficulty. Instead of branching on the λ 's in the master problem, we use a branching rule that corresponds to branching on the original variables z_{ij} . When $z_{ij} = 1$, all existing columns in the master that don't assign task i to machine j are deleted and task i is permanently assigned to machine j , i.e., variable x_i is fixed to 1 in the j th knapsack. When $z_{ij} = 0$, all existing columns in the master that assign job i to machine j are deleted and task i cannot be assigned to machine j , i.e., variable x_i is removed from the j th knapsack. Note that each of the knapsack problems contains one fewer variable after the branching has been done.

Observe that the branching scheme discussed above is specific to the GAP. This is typical of branch-and-price algorithms. Each problem requires its own "problem-specific" branching scheme.

tailing-off

Lagrangian relaxation

In practice, one of the computational difficulties encountered when applying branch-and-price is the so-called *tailing-off* effect of the column generation, i.e., the large number of iterations needed to prove the optimality of the LP solution. Potentially, this may happen at every node of the search tree. Also, the pricing problem that needs to be solved at each column generation iteration may be difficult and time consuming. Fortunately, the branch-and-bound framework has some inherent flexibility that can be exploited effectively in branch-and-price algorithms. Branch-and-bound is an enumeration scheme that is enhanced by fathoming based on bound comparisons. To control the size of the branch-and-bound tree it is best to work with strong bounds; however, the method will work with any bound. Therefore, instead of solving the linear program to optimality, i.e., generating columns as long as profitable columns exist, we can choose to prematurely end the column generation process and work with bounds on the final LP value.

Again, consider the alternative formulation of GAP. By dualizing the assignment constraints, we obtain the following *Lagrangian relaxation*, which provides an upper bound on the value of the LP for any vector u .

$$\begin{aligned} \max \quad & \sum_{1 \leq j \leq n} \sum_{1 \leq k \leq K_j} \left(\sum_{1 \leq i \leq m} p_{ij} y_{ik}^j \right) \lambda_k^j + \\ & \sum_{1 \leq i \leq m} u_i \left(1 - \sum_{1 \leq j \leq n} \sum_{1 \leq k \leq K_j} y_{ik}^j \lambda_k^j \right) \end{aligned}$$

subject to

$$\sum_{1 \leq k \leq K_j} \lambda_k^j = 1 \quad j = 1, \dots, n,$$

$$\lambda_k^j \in \{0, 1\} \quad j = 1, \dots, n, \quad k = 1, \dots, K_j.$$

After some algebraic manipulations, we obtain

$$\begin{aligned} & \sum_{1 \leq i \leq m} u_i + \\ & \sum_{1 \leq j \leq n} \max_{1 \leq k \leq K_j} \left(\sum_{1 \leq i \leq m} (p_{ij} - u_j) y_{ik}^j \right) \lambda_k^j \end{aligned}$$

subject to

$$\sum_{1 \leq k \leq K_j} \lambda_k^j = 1 \quad j = 1, \dots, n,$$

$$\lambda_k^j \in \{0, 1\} \quad j = 1, \dots, n, \quad k = 1, \dots, K_j,$$

which is equivalent to

$$\sum_{1 \leq i \leq m} u_i + \sum_{1 \leq j \leq n} z(KP_j).$$

This shows that after solving the pricing problem, we have all the information necessary to compute an upper bound on the value of the final LP solution. Therefore, after every column generation iteration, we may decide to prematurely end the column generation process if the value of the LP solution to the current restricted master problem, which provides a lower bound on the final LP value, and this upper bound are sufficiently close.

Identical Machines

This is a special case of the problem with non-identical machines and therefore the methodology described above applies. However, we need only one subproblem since all of the machines are identical, which implies that the λ_k^j can be aggregated by defining $\lambda_k = \sum_j \lambda_k^j$ and that the convexity constraints can be combined into a single constraint $\sum_{1 \leq k \leq K} \lambda_k = n$ where λ_k is restricted to be integer. In some cases the aggregated constraint will become redundant and can be deleted altogether. An example of this is when the objective is to minimize $\sum \lambda_k$, i.e., the number of machines needed to process all the tasks. Note that this special case of GAP is equivalent to a 0-1 *cutting stock problem*.

A much more important issue here concerns symmetry, which causes branching on the original variables to perform very poorly. With identical machines, there are an exponential number of solutions that differ only by the names of the machines, i.e. by swapping the assignments of 2 machines we get 2 solutions that are the same but have different values for the variables. This statement is true for fractional as well as 0 - 1 solutions. The implication is that when a fractional solution is excluded at some node of cutting stock problem

the tree, it pops up again with different variable values somewhere else in the tree. In addition, the large number of alternate optima dispersed throughout the tree renders pruning by bounds nearly useless.

The remedy here is a different branching scheme that works directly on the master problem but focuses on pairs of tasks. In particular, we consider rows of the master with respect to tasks r and s . Branching is done by dividing the solution space into one set in which r and s appear together, in which case they can be combined into one task when solving the knapsack, and into another set in which they must appear separately, in which case a constraint $x_r + x_s \leq 1$ is added to the knapsack. Note that the structure of the subproblems is no longer the same on the different branches.

Most of the material presented above is based on [3], in which the term branch-and-price was first introduced, and [1], in which the concepts of branch-and-price are covered in much more detail. Another important source of information on branch-and-price is [4], in which various general branching schemes and bounding schemes are discussed. Routing and scheduling has been a particularly fruitful application area of branch-and-price, see Desrosiers et al. [2] for a survey of these results.

References

- [1] BARNHART, C., JOHNSON, E.L., NEMHAUSER, G.L., SAVELSBERGH, M.W.P., AND VANCE, P.H.: ‘Branch-and-Price: Column Generation for Solving Integer Programs’, *Operations Research* **46** (1998), 316–329.
- [2] DESROSIERS, J., DUMAS, Y., SOLOMON, M.M., AND SOUMIS, F.: ‘Time constrained routing and scheduling’, *Handbooks in Operations Research and Management Science, Volume 8: Network Routing*, in M.E. BALL, T.L MAGNANTI, C. MONMA, AND G.L. NEMHAUSER (eds.). Elsevier, Amsterdam, 1995, pp. 35–140.
- [3] SAVELSBERGH, M.W.P.: ‘A Branch-and-Price Algorithm for the Generalized Assignment Problem’, *Operations Research* **6** (1997), 831–841.
- [4] VANDERBECK, F., AND WOLSEY, L.A.: ‘An Exact Algorithm for IP Column Generation’, *Operations Research Letters* **19** (1996), 151–160.

Martin W.P. Savelsbergh

School of Industrial and Systems Engineering

Georgia Institute of Technology

Atlanta, GA 30332-0205

E-mail address:

`martin.savelsbergh@isye.gatech.edu`

AMS1991 Subject Classification: 68Q99.

Key words and phrases: optimization, integer programming, decomposition, column generation.