

An Introduction to Branch-and-Price Part II: Algorithms and Applications

Adam N. Letchford

Department of Management Science
Lancaster University

September 2011

Outline

- 1 Solving the LP relaxation ('pricing')
- 2 Solving the ILP ('branch-and-price')
- 3 Example I: Generalised Assignment Problem
- 4 Example II: Capacitated Facility Location Problem
- 5 Example III: Cutting Stock Problem

Quick recap

- One way to strengthen an ILP formulation of a COP is by adding a huge number of *rows* (constraints). To solve the resulting ILP, we can use *branch-and-cut*.
- Another way to strengthen the ILP formulation is to add a huge number of *columns* (variables). We then have to use *branch-and-price*.
- Branch-and-price works best when the COP has a natural ILP formulation with a 'block-angular' constraint matrix.

Quick recap (cont.)

The master problem (in the general case) takes the form:

$$\begin{aligned} \min \quad & \tilde{c} \cdot \lambda \\ \text{s.t.} \quad & \tilde{C}\lambda \geq d \\ & \sum_{k=1}^t \lambda_k = 1 \\ & \lambda \in \{0, 1\}^t. \end{aligned}$$

There is one λ variable for each vertex of P_I^1 .

Solving the LP relaxation

The LP relaxation of the master problem looks like this:

$$\begin{aligned} \min \quad & \tilde{c} \cdot \lambda \\ \text{s.t.} \quad & \tilde{C}\lambda \geq d \\ & \sum_{k=1}^t \lambda_k = 1 \\ & \lambda \in \mathbb{R}_+^t. \end{aligned}$$

Obviously we can't solve this LP with the standard simplex method!

Solving the LP relaxation (cont.)

We use the following *column generation* approach:

- 1 Start with a ‘restricted’ master problem (RMP) containing a small subset of the columns.
- 2 Solve the LP relaxation of the RMP by the simplex method.
- 3 Check if any columns *not* in the RMP have a negative reduced cost.
- 4 If no such columns exist, stop. We have found an optimal solution to the LP relaxation of the master problem.
- 5 Otherwise, add one or more columns to the RMP.
- 6 Re-optimize the LP via primal simplex and return to 3.

Solving the LP relaxation (cont.)

The problem of checking whether new columns need to be generated is called the *pricing problem*.

From duality theory, we can solve the pricing problem by solving the 0-1 LP:

$$\min \{ (c - C^T \pi) \cdot x : x \in P_I^1 \},$$

where π is the vector of dual prices for the constraints $\tilde{C}\lambda \geq d$.

By assumption, this is an 'easy' 0-1 LP to solve.

If the cost of the solution is less than the dual price of the convexity constraint, the column has a negative reduced cost.

Solving the LP relaxation (cont.)

- If the 0-1 LP has block-angular structure, then we solve one pricing problem for each block.
- That means solving one very small and easy 0-1 LP for each block.
- If the blocks are identical, then we only need to solve one very small and easy 0-1 LP!

Branch-and-price

- We have established that the LP relaxation of the master problem can be solved using column generation.
- This provides a (typically very strong) lower bound for our original COP.
- If we then want to solve the COP to proven optimality, we have to start branching.
- We price at *each* node of the branch-and-bound tree, just in case more columns of negative reduced cost can be found.
- The name 'branch-and-price' comes from Savelsbergh (1997).

Branch-and-price (cont.)

Very Important!

It is usually best to branch on the original x variables, not on the λ variables.

(After all, if we fixed a λ variable to zero, how could we stop it being generated again in the next pricing iteration?!)

Branch-and-price (cont.)

Just as in the case of branch-and-cut, some extra ‘tricks’ can speed things up. For example:

- Delete non-basic variables periodically to stop the LPs becoming too large.
- Call fast pricing heuristics before resorting to exact pricing.
- Periodically round the fractional solutions to integer ones to obtain upper bounds.

Example I: the GAP

The Generalised Assignment Problem (GAP) is a well-known COP, with many applications.

- We have n jobs and m machines.
- Each job must be assigned to one machine.
- If we process job i on machine j we get a profit p_{ij} ...
- ... but we consume a_{ij} units of a resource.
- Machine j has b_j units of the resource.
- We wish to maximise our total profit.

Example I: the GAP (cont.)

A 0-1 LP formulation of the GAP is:

$$\begin{aligned} \max \quad & \sum_{i=1}^n \sum_{j=1}^m p_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_{i=1}^n a_{ij} x_{ij} \leq b_j \quad (j = 1, \dots, m) \end{aligned} \quad (1)$$

$$\begin{aligned} & \sum_{j=1}^m x_{ij} = 1 \quad (i = 1, \dots, n) \quad (2) \\ & x \in \{0, 1\}^{n \times m}. \end{aligned}$$

We will call (1) the 'knapsack' constraints and (2) the 'assignment' constraints.

Example I: the GAP (cont.)

At first sight, it looks like the knapsack constraints are 'nasty' and the assignment constraints are 'nice'.

Example I: the GAP (cont.)

At first sight, it looks like the knapsack constraints are 'nasty' and the assignment constraints are 'nice'.

In fact, it is best to take the opposite point of view (Savelsbergh, 1997)!

Example I: the GAP (cont.)

At first sight, it looks like the knapsack constraints are 'nasty' and the assignment constraints are 'nice'.

In fact, it is best to take the opposite point of view (Savelsbergh, 1997)!

WHY?

Example I: the GAP (cont.)

At first sight, it looks like the knapsack constraints are 'nasty' and the assignment constraints are 'nice'.

In fact, it is best to take the opposite point of view (Savelsbergh, 1997)!

WHY?

Because the constraint matrix has block-angular structure! If we delete the assignment constraints, we have one block for each machine.

Example I: the GAP (cont.)

In the branch-and-price algorithm, the pricing subproblem for machine j takes the form:

$$\begin{aligned} \max \quad & \sum_{i=1}^n (p_{ij} - \pi_i) x_i \\ \text{s.t.} \quad & \sum_{i=1}^n a_{ij} x_i \leq b_j \\ & x \in \{0, 1\}^n. \end{aligned}$$

where π_i is the dual price for the i th assignment constraint.

This is a 0-1 knapsack problem. Although the 0-1 knapsack problem is \mathcal{NP} -hard, it can be solved quickly in practice (e.g., by dynamic programming).

Example II: a CFLP

Capacitated Facility Location Problems (CFLPs) are another well-known and important class of COPs.

In one version of the problem:

- We have n customers and m potential locations for facilities.
- Customer i has a demand a_i .
- Facility j , if built, will cost c_j and have capacity b_j .
- If customer i is serviced by location j we will gain a profit p_{ij} .
- We wish to maximise our total profit.

Example II: a CFLP (cont.)

A 0-1 LP formulation of the CFLP is:

$$\max \quad \sum_{i=1}^n \sum_{j=1}^m p_{ij} x_{ij} - \sum_{j=1}^m c_j y_j$$

$$\text{s.t.} \quad \sum_{i=1}^n a_i x_{ij} \leq b_j y_j \quad (j = 1, \dots, m) \quad (3)$$

$$\sum_{j=1}^m x_{ij} = 1 \quad (i = 1, \dots, n) \quad (4)$$

$$x \in \{0, 1\}^{n \times m}$$

$$y \in \{0, 1\}^m.$$

As before, we have 'knapsack' constraints (3) and 'assignment' constraints (4). But we now have the complication of the y variables.

Example II: a CFLP (cont.)

As before, we view the assignment constraints as the 'nasty' ones.

The pricing subproblem for location j takes the form:

$$\begin{aligned} \max \quad & \sum_{i=1}^n (p_{ij} - \pi_i) x_i - c_j y_j \\ \text{s.t.} \quad & \sum_{i=1}^n a_i x_i \leq b_j y_j \\ & x \in \{0, 1\}^n \\ & y_j \in \{0, 1\}. \end{aligned}$$

This is trivial when $y_j = 0$. When $y_j = 1$, it reduces to the 0-1 knapsack problem.

Example III: the CSP

The Cutting Stock Problem (CSP) is another well-known and important COP.

- We have an unlimited (or large) number of rolls of paper available, each of width W .
- We have n orders from customers.
- Customer i has ordered d_i strips of paper of width w_i .
- We wish to minimise the number of rolls used.

This was the first COP to be tackled with Dantzig-Wolfe decomposition (Gilmore & Gomory, 1961, 1963).

Example III: the CSP (cont.)

Let m be an upper bound on the number of rolls used. (Perhaps found by some simple heuristic.)

Here is an ILP formulation:

$$\begin{aligned} \min \quad & \sum_{j=1}^m y_j \\ \text{s.t.} \quad & \sum_{i=1}^n w_i x_{ij} \leq W y_j \quad (j = 1, \dots, m) \end{aligned} \quad (5)$$

$$\sum_{i=1}^n x_{ij} \geq d_i \quad (i = 1, \dots, n) \quad (6)$$

$$x \in \mathbb{Z}_+^{n \times m}.$$

$$y \in \{0, 1\}^n.$$

As usual, we view (5) as the 'nice' constraints and (6) as the 'nasty' ones.

Example III: the CSP (cont.)

Some observations:

- We now have some general-integer variables as well as binary ones.
- The 'nasty' constraints now have a right-hand side greater than one.
- The ILP has block-angular structure (one block per roll).

More importantly:

- The blocks are identical!

This is so since each roll of paper has the same width.

Example III: the CSP (cont.)

The master problem takes a relatively simple form:

$$\begin{aligned} \min \quad & \sum_{k=1}^t \lambda_k \\ \text{s.t.} \quad & \sum_{k=1}^t r_{ik} \lambda_k \geq d_i \quad (i = 1, \dots, n) \\ & \lambda \in \mathbb{Z}_+^t. \end{aligned}$$

Here, t is the number of 'cutting patterns' (ways in which a single roll of paper can be cut), and r_{ik} is the number of strips of width w_i that are cut in cutting pattern k .

Example III: the CSP (cont.)

We only need to solve *one* pricing subproblem, of the form:

$$\begin{aligned} \max \quad & \sum_{i=1}^n \pi_i x_i \\ \text{s.t.} \quad & \sum_{i=1}^n w_i x_i \leq W \\ & x \in \mathbb{Z}_+^n. \end{aligned}$$

If the optimal solution has a profit larger than 1, we have found a column with negative reduced cost.

This is a *general-integer* knapsack problem. It is \mathcal{NP} -hard, but can be solved quickly in practice by dynamic programming.

Some other applications of branch-and-price

- Crew scheduling (Desrochers & Soumis, 1989)
- Vehicle routing with time windows (Desrosiers *et al.*, 1992)
- Multi-item lot sizing (Vanderbeck, 1994)
- Job grouping in flexible manufacturing systems (Crama & Oerlemans, 1994)
- Graph colouring (Mehrotra & Trick, 1996)
- Multi-commodity flows (Barnhart *et al.*, 1997)
- Bin packing (Valério de Carvalho, 1999)
- Scheduling on parallel machines (van den Akker *et al.*, 1999)
- Airline schedule generation (Erdmann *et al.*, 2001)
- Real-time vehicle dispatching (Krumke *et al.*, 2002)

Concluding remarks

- Branch-and-cut and branch-and-price are two complementary methods for solving \mathcal{NP} -hard COPs.
- Branch-and-cut works best if the polyhedron is easy to analyse.
- Branch-and-price works best if the problem has a natural ILP formulation with block-angular structure.
- Researchers are now looking at combining both row and column generation, leading to *branch-cut-and-price* algorithms!