

An Introduction to Branch-and-Cut

Part II: Algorithms

Adam N. Letchford

Department of Management Science
Lancaster University

September 2011

Outline

- 1 Cut-and-branch algorithms
- 2 Branch-and-cut algorithms
- 3 Separation
- 4 Common enhancements
- 5 Brief survey of applications

Quick Recap

Most Combinatorial Optimization Problems (COPs) can be formulated as *Integer Linear Programmes* (ILPs) of the form:

$$\min \{c^T x : Ax \geq b, x \in \mathbb{Z}_+^n\},$$

Strong cutting planes can be derived from an analysis of the *integral hull*:

$$P_I = \text{conv} \{x \in \mathbb{Z}_+^n : Ax \geq b\}.$$

The strongest inequalities are the ones defining *facets* of P_I .

Cut-and-branch algorithms

In the 1970s and 1980s, people experimented with the following approach:

- 1 Solve an initial LP relaxation (by primal simplex).
- 2 Let x^* be the solution. If x^* is integer and feasible, output the optimal solution and stop.
- 3 Search for strong (facet-defining) inequalities violated by x^* . If none are found, go to step 6.
- 4 Add the inequalities to the LP as cutting planes.
- 5 Resolve the LP (by dual simplex) and go to step 2.
- 6 Run branch-and-bound (keeping the cutting planes in the ILP formulation).

Cut-and-branch algorithms (cont.)

- The above approach was popularised by Crowder, Johnson & Padberg (1983).
- They used *lifted cover inequalities* to tackle 0-1 LPs.
- They won a prize for solving ten large-scale 0-1 LPs that previously were thought to be unsolvable.
- The approach has come to be known as *cut-and-branch*.
- It works very well for some problems and is still popular.

Cut-and-branch algorithms (cont.)

Cut-and-branch has one disadvantage:

If our ILP formulation has an exponential number of constraints (as in the TSP) then cut-and-branch might lead to an infeasible solution!

This is because the final solution may be integer, but may violate a constraint that was not generated in the cutting-plane phase.

Cut-and-branch algorithms (cont.)

In the 1980s, Grötschel, Padberg and co-authors used a modified version of cut-and-branch to solve quite large TSPs (up to 300 cities or so):

- 1 Run the cutting-plane algorithm.
- 2 If the solution represents a tour, stop.
- 3 Run branch-and-bound on the enlarged formulation.
- 4 If the integer solution represents a tour, stop.
- 5 Find one or more subtour elimination constraints that are violated by the integer solution. Add them to the ILP formulation and return to step 3.

Branch-and-cut algorithms

- Why not search for cutting planes at every node of the branch-and-bound tree?
- A primitive version of this idea was already applied to the TSP by Hong (1972) and Miliotis (1976).
- Grötschel, Jünger & Reinelt (1984) applied it to the so-called *Linear Ordering Problem*.
- The term *branch-and-cut* was coined by Padberg & Rinaldi (1987, 1991).
- They used it to solve very large TSP instances (up to 2000 cities or so).

Branch-and-cut algorithms (cont.)

Main ingredients of a branch-and-cut algorithm:

- Linear programming solver (capable of both primal and dual simplex).
- Branch-and-bound shell which enables cutting planes to be added at any node of the tree.
- Subroutines that search for strong cutting planes.
- A rule for deciding how to branch.
- A rule for which subproblem to work on next.

Branch-and-cut algorithms (cont.)

Remark

Branch-and-cut is considerably harder to implement than cut-and-branch!

So only implement it if you have to.

In many cases, cut-and-branch or plain branch-and-bound will suffice (or indeed heuristics).

Separation

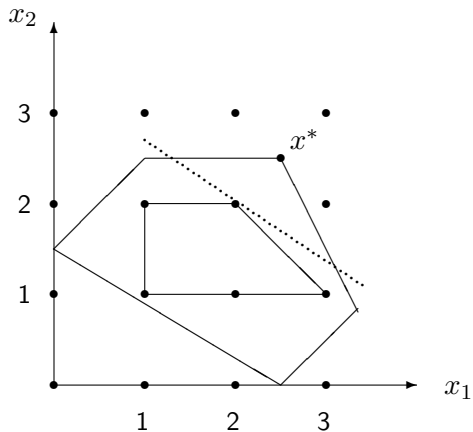
There is one important issue that we have not discussed: how exactly do we generate cutting planes?

More formally:

Given a polyhedron P , whose integral hull is P_I , and a vector $x^ \in P \setminus P_I$, how do we find an inequality that is valid for P_I but violated by x^* ?*

This is called the *separation problem* (Grötschel, Lovász & Schrijver, 1988).

Separation (cont.)



We seek a hyperplane that separates x^* from P_I .

Separation (cont.)

Some bad news:

- For most COPs, we do not know all the facets.
- Even the strong inequalities that we do know about can be exponential in number.
- If the ILP is \mathcal{NP} -hard, then the separation problem is also \mathcal{NP} -hard (Grötschel *et al.*, 1988)!

The good news:

- For some *particular* COPs and some *particular classes* of strong inequalities, we can efficiently search for violated inequalities in that class.

Separation (cont.)

This leads to the following concepts:

Definition

An *exact separation algorithm* for a given class of inequalities is a procedure which takes a vector x^* as input, and either outputs one or more inequalities in the class which are violated by x^* , or proves that none exists.

Definition

A *heuristic separation algorithm* is similar, except that it outputs either one or more inequalities in the class which are violated by x^* , or a failure message.

Example I: the TSP

Recall that the following *subtour elimination constraints* induce facets of the traveling salesman polyhedron:

$$\sum_{e \in \delta(S)} x_e \geq 2 \quad (\forall S \subset V).$$

Recall also that they are exponential in number.

To solve the separation problem, we need to check if there exists a vertex set S such that $\sum_{e \in \delta(S)} x_e^* < 2$.

This amounts to finding a minimum-weight cut in a graph.

Gomory & Hu (1961) showed how to do this by solving $n - 1$ max-flow/min-cut problems.

Example I: the TSP (cont.)

Exact separation algorithms are known for various other TSP inequalities:

- 2-matching inequalities (Padberg & Rao, 1982; Letchford *et al.*, 2004).
- Simple comb inequalities (Letchford & Lodi, 2002)
- Comb inequalities on planar graphs (Letchford, 2000)
- Clique-tree inequalities with fixed number of handles (Carr, 1995)

There are also many effective separation heuristics (see book by Applegate *et al.*, 2007).

Example II: matching

Recall that the following *odd cut* inequalities induce facets of the perfect matching polyhedron:

$$\sum_{e \in \delta(S)} x_e \geq 1 \quad (\forall S \subset V : |S| \text{ odd}).$$

To solve the separation problem, we need to check if there exists a vertex set S , with $|S|$ odd, such that $\sum_{e \in \delta(S)} x_e^* < 1$.

This amounts to finding a minimum-weight *odd cut* in a graph. Padberg & Rao (1982) showed that this too can be reduced to $n - 1$ max-flow/min-cut problems.

Example II: matching (cont.)

You might be thinking:

Who cares about the separation problem for odd cut inequalities? We can solve the perfect matching problem directly with Edmonds' blossom algorithm!

In fact, many important \mathcal{NP} -hard COPs have a 'matching aspect'.

So odd cut inequalities (or inequalities like them) are very useful cutting planes for many COPs.

Example III: knapsack

Recall that the following *cover* inequalities are valid for the 0-1 knapsack polyhedron:

$$\sum_{i \in C} x_i \leq |C| - 1 \quad (\forall C : \sum_{i \in C} w_i > c).$$

To solve the separation problem, we need to check if there exists a set of items C such that $\sum_{i \in C} w_i > c$ and $\sum_{i \in C} x_i^* > |C| - 1$.

Crowder *et al.* (1983) transformed this problem itself into a knapsack problem.

They then solved this auxiliary knapsack problem using a greedy heuristic.

Example III: knapsack (cont.)

You might be thinking:

Who cares about the separation problem for cover inequalities? We can solve the 0-1 knapsack problem directly with dynamic programming!

This too would be a mistake. Many important \mathcal{NP} -hard COPs have a 'knapsack aspect'.

So cover inequalities (and strengthened versions) are very useful cutting planes for many COPs.

Enhancements

Crowder *et al.* (1983) found an effective way to speed up cut-and-branch:

- Run a heuristic to get an upper bound U .
- Run the cutting plane algorithm to get a lower bound L .
- Let rc_i be the reduced cost of variable x_i .
- For all i such that $L + rc_i > U$, fix x_i to zero.
- This reduces the size of the ILP.

This is now called *reduced-cost fixing*.

Enhancements (cont.)

Padberg & Rinaldi (1987, 1991) proposed some enhancements to the basic branch-and-cut scheme:

- Every time a violated inequality is found, store it in a 'cut pool'.
- At each new node, scan the cut pool for violated inequalities before running the separation algorithms.
- Periodically remove non-binding inequalities from the LP, to keep the LPs small.
- Call reduced-cost fixing at each node.

Brief survey of applications

Linear Ordering: Grötschel, Jünger & Reinelt (1984).

TSP: Grötschel & Holland (1991), Padberg & Rinaldi (1987, 1991), Naddef & Thienel (2002), Applegate *et al.* (1998, 2003)...

General 0-1 LPs: Crowder *et al.* (1983), Hoffman & Padberg (1985, 1991), Gu *et al.* (1998), Kaparis & Letchford (2008).

General Mixed 0-1 LPs: Balas *et al.* (1996).

General MILPs: Courdier *et al.* (1999).

Brief survey of applications (cont.)

Airline Crew Scheduling: Hoffman & Padberg (1993).

Facility Location: Aardal (1998).

Graph Drawing: Jünger & Mutzel (1996).

Graph Partitioning: Brunetta *et al.* (1997).

Lot-Sizing: Belvaux & Wolsey (2000).

Vehicle Routing: Araque *et al.* (1994), Augerat *et al.* (1995), Letchford *et al.* (2002, 2004).

... and many more.

Final Remark

Although cut-and-branch and branch-and-cut work well for many COPs, they tend not to work well for highly-constrained COPs (such as certain machine scheduling and vehicle routing problems).

In such cases one can try other exact methods such as:

- dynamic programming
- Lagrangian relaxation
- constraint programming
- branch-and-price.

Or one can use heuristics!