Implementing Branch, Cut, and Price

Ted Ralphs
Industrial and Systems Engineering, Lehigh University

Joint work with:
Laci Ladanyi
IBM T.J. Watson Research Center

Workshop on Novel Approaches to Discrete Optimization Problems
University of Waterloo, April 27, 2001
Outline of Talk

- Introduction to Branch, Cut, and Price (BCP)
- Frameworks for BCP
- Implementing BCP
- Description of SYMPHONY and COIN/BCP
- Lessons Learned
LP-based Branch and Bound

- Consider problem $P$:

\[
\begin{align*}
\min & \quad c^T x \\
\text{s.t.} & \quad Ax \leq b \\
& \quad x_i \in \mathbb{Z} \ \forall \ i \in I
\end{align*}
\]

where $(A, b) \in \mathbb{R}^{m \times n+1}, c \in \mathbb{R}^n$.

- Let $\mathcal{P} = \text{conv}\{x \in \mathbb{R}^n : Ax \leq b, x_i \in \mathbb{Z} \ \forall \ i \in I\}$.

- Basic Algorithmic Approach
  - Use LP relaxations to produce lower bounds.
  - Branch using hyperplanes.

- Basic Algorithmic Elements
  - A method for producing and tightening the LP relaxations.
  - A method for branching.
Branch, Cut, and Price

• Weyl-Minkowski
  - $\exists (\bar{A}, \bar{b}) \in \mathbb{R}^{\bar{m} \times n+1}$ s.t. $\mathcal{P} = \{x \in \mathbb{R}^n : \bar{A}x \leq \bar{b}\}$
  - We want the solution to $\min\{c^T x : \bar{A}x \leq \bar{b}\}$.
  - Solving this LP isn’t practical (or necessary).

• BCP Approach
  - Form LP relaxations using submatrices of $\bar{A}$.
  - The submatrices are defined by sets $\mathcal{V} \subseteq [1..n]$ and $\mathcal{C} \subseteq [1..\bar{m}]$.
  - *Forming/managing these relaxations efficiently is one of the primary challenge of BCP.*
The Challenge of BCP

• The efficiency of BCP depends heavily on the size (number of rows and columns) and tightness of the LP relaxations.

• Tradeoff
  - Small LP relaxations ⇒ faster LP solution.
  - Big LP relaxations ⇒ better bounds.

• The goal is to keep relaxations small while not sacrificing bound quality.

• We must be able to easily move constraints and variables in and out of the active set.

• This means dynamic generation and deletion.
An Object-oriented Approach

- The rows/columns of a static LP are called constraints and variables.
- What do these terms mean in a dynamic context?
- Conceptual Definitions
  - **Constraint**: A mapping
    \[ f^c_i(C) : 2^{[1..n]} \rightarrow \mathbb{R}^{|C|} \]
    generating coefficients for the submatrix \( C \).
  - **Variable**: A mapping
    \[ f^v_j(V) : 2^{[1..\bar{m}]} \rightarrow \mathbb{R}^{|V|} \]
    generating coefficients for the submatrix \( C \).
- To construct a relaxation, an initial core is needed.
- From the core, we can build up the relaxation.
Software Frameworks

• **Concept**: Provide a *framework* in which the user has only to define constraints, variables, and a core.
  - Branch and bound $\Rightarrow$ core only
  - Branch and cut $\Rightarrow$ core plus constraints
  - Branch and price $\Rightarrow$ core plus variables
  - Branch, cut, and price $\Rightarrow$ the whole caboodle

• **BCP frameworks**
  - SYMPHONY (parallel)
  - COIN/BCP (parallel)
  - ABACUS (sequential)

• **Other frameworks**
  - PICO, PUBB, BoB, PPBB-Lib (branch and bound)
  - MINTO (MIP), BARON (NLP)
Generating the Objects

- We will generically call the constraints and variables *objects*.
- We need to define methods for generating these objects.
- For *constraints*, such a method is a mapping
  \[ g^c(x) : \mathbb{R}^n \rightarrow 2^{[1..\bar{m}]} \]
  where \( x \) is a *primal solution* vector.
- For *variables*, we have
  \[ g^v(y) : \mathbb{R}^m \rightarrow 2^{[1..n]} \]
  where \( y \) is a *dual solution* vector.
- We can also use *object pools* to help with generation.
Object Representation

- In practice, we may not know the cardinality of the object set.
- We may not easily be able to assign indices to the objects.
- Instead, we must define **abstract representations** of these objects.
- In C, this means defining a function which takes the abstract data structure as an argument and returns a row/column.
- In C++, this means deriving a new class.
- **Example**: Subtour elimination constraints.
Example: Traveling Salesman Problem

Feasible solutions are those incidence vectors satisfying:

\[ \sum_{j=1}^{n} x_{ij} = 2 \quad \forall i \in V \]
\[ \sum_{\substack{i \in S \atop j \notin S}} x_{ij} \geq 2 \quad \forall S \subset V, \ |S| > 1. \]

- The variables correspond to the edges of a graph (easy to index).
- The number of facets (constraints) is astronomical.
- The core
  - The $k$ shortest edges adjacent to each node.
  - The degree constraints.
- Generate subtour elimination constraints and other variables dynamically.
Object Pools

• One or more object pools maintain a list of the most “effective” objects found so far.

• Each pool services a subtree – pools are dynamically allocated.

• The use of multiple pools allows locally valid cuts to be generated.

• With multiple pools, pools are smaller and contain cuts that were generated “closer” in the tree ⇒ more likely to be violated.

• The size of each pool is controlled through the purging of “ineffective” objects.
BCP Modules

There are six module types:

**Master** Maintains problem instance data, spawns other processes, performs I/O, fault tolerance.

**Tree Manager** Controls overall execution by tracking growth of the tree and dispatching subproblems to the LP solvers.

**LP Solvers** Perform processing and branching operations.

**Object Generators** Generate objects.

**Object Pools** Act as auxiliary object generators by maintaining a list of the “most effective” objects found so far.

**GUI** Allows graphical display of fractional and integer solutions.
Managing the LP Relaxation

Constraints

- Cuts are generated by the cut generators and using cut pools.
- Violated cuts are received and processed by the LP modules.
- Each LP module maintains a small local cut pool.
- A limited number of cuts are added to the LP relaxations each iteration to prevent “saturation.”
- Ineffective (non-core) cuts are aggressively removed.
- Cuts are only sent to a global pool if they prove effective locally.
Managing the LP Relaxation
Variables

- **Reduced cost/logical fixing** are used to remove (non-core) variables.

- **Variable generation** is needed only for very large problems.
  - Unlike cuts, adding variables “loosens” the formulation.
  - Variable generation may be inefficient in cases where it is not needed.
  - **Exact generation must take place before fathoming!**

- **Two-phase algorithm**
  - BCP is run to completion on the core variables before generating new ones.
  - Using the upper bound and cuts from the first phase, all variables are priced out in the root node and are then propagated down into the leaves as required.
  - **The tree is trimmed** by aggregating children back into their parent.
  - Afterwards, each leaf is processed again.
Managing the Search Tree

Fathoming

• Fathoming occurs when
  - the lower bound for the subproblem is provably greater than or equal to the known upper bound, or
  - The subproblem is proven infeasible.

• It is necessary to have every variable either
  - Present in the subproblem, or
  - Fixed by reduced cost.

• This means exact column generation.
Managing the Search Tree
Branching

• If we fail to fathom or locate any more objects that should be included in the current relaxation, we must branch.

• Branching can be done on any object or set of objects.

• All that is needed is to specify object bounds in each branch.

• No matter what objects are used, strong branching is a critical tool.
  - Select several branching candidates.
  - “Presolve” each candidate.
  - Choose the “best” for branching.

• Fractional branching is also an option.
Managing the Search Tree
Storage and Search Strategy

• Data Storage
  - Efficient data storage is essential.
  - The state of the entire tree is stored, including warm-start info—*important*!
  - The description of each node can be stored explicitly or with respect to its parent, whichever is smaller.

• Tree Management
  - The search algorithm should be an adaptable hybrid of depth-first and best-first.
  - Best-first theoretically minimizes the size of the tree.
  - Depth-first avoids node set-up costs.
Conclusions

Key Points:

- Keep relaxations small!
  - Be conservative when adding cuts and liberal when deleting them.
  - Use object pools wisely.

- Choose the core properly.

- Maintain warm-start information.

- Be careful with variable generation.

- Use strong branching.

- Use an adaptable search strategy.
SYMPHONY and COIN/BCP

- Designed to run in a parallel environment.
  - Serial, fully distributed (using PVM), or shared-memory (using threads) modes.
  - No knowledge of parallelism required.
  - Runs in heterogeneous Unix environments.

- User supplies:
  - data structures for objects,
  - object generation subroutines,
  - description of the core,
  - feasibility checker, and
  - other optional subroutines.

- The framework takes care of everything else.

- www.BranchAndCut.org