Branch, Constrain, and Price Algorithms for Large-scale Discrete Optimization

Ted Ralphs  
Industrial and Systems Engineering  
Lehigh University  
http://www.lehigh.edu/~tkr2

Laszlo Ladanyi  
IBM T.J. Watson Research Center

Matt Saltzman  
Clemson University

Rutgers Center for Operations Research 20 September 2001
Outline of Talk

- Introduction to Branch, Cut, and Price (BCP)
- Frameworks for BCP
- The Abstract Library for Parallel Search
- Implementation Issues for BCP
  - Parallel Scalability
  - Data Handling
- Advanced Algorithms
- What’s Available
LP-based Branch and Bound

• Consider problem $P$:

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

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

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

• 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**
  
  - \( \exists (\bar{A}, \bar{b}) \in \mathbb{R}^{m \times n+1} \) 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 challenges 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 $f_i(C) : 2^{[1..n]} \rightarrow \mathbb{R}^{|C|}$ generating coefficients for the submatrix $C$.
  - **Variable**: A mapping $g_j(V) : 2^{[1..\bar{m}]} \rightarrow \mathbb{R}^{|V|}$ generating coefficients for the submatrix $C$.
  - **Subproblem**: A subset $(C, V)$ of $[1..n] \times [1..\bar{m}]$.
- To construct a subproblem, an initial *core relaxation* is needed.
- From the core, we can build up other relaxations using the mappings.
Frameworks for BCP

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

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

• **Other frameworks**
  
  – PICO, PUBB, BoB, PPBB-Lib (branch and bound)
  – MINTO (branch and cut)
The ALPS Project

• In partnership with IBM and the COIN-OR project.

• Multi-layered C++ class library for implementing scalable, parallel tree search algorithms.

• Design is fully generic and portable.
  – Design “abstracts” notions from BCP.
  – Support for implementing general tree search algorithms.
  – Support for any bounding scheme.
  – No assumptions on problem/algorithm type.
  – No dependence on architecture/operating system.
  – No dependence on third-party software (communications, solvers).

• Increased parallel scalability.

• Support for large-scale, data-intensive applications (such as BCP).

• Support for advanced methods not available in commercial codes.
ALPS Design

Modular library design with minimal assumptions in each layer.

**ALPS** Abstract Library for Parallel Search

- manages the search tree.
- prioritizes based on **quality**.

**BiCePS** Branch, Constrain, and Price Software

- manages the data.
- adds notion of **primal and dual objects**.

**BLIS** BiCePS Linear Integer Solver

- assumes **linear constraints** and a **linear objective**.
- utilizes LP relaxations.
ALPS: Abstract Library for Parallel Search

Properties of a search tree node:

- **status**: candidate, processed, branched, fathomed.
- **quality**: a numerical priority (below threshold ⇒ fathomed).

Operations on the search tree nodes:

- create children (branch).
- remove a node (recursively: remove a subtree).
ALS Design

ALPS: Abstract Notions

Procedural abstractions:

• Process
  – status candidate → processed/fathomed.

• Branch
  – status processed → branched.
  – create children (candidate nodes) and add to queue.

Data management abstraction:

• Differencing scheme
  – node description can be stored with respect to parent.
  – explicit description can be extracted.
  – relative description can be created.
BiCePS: Branch, Constrain, and Price Software

Adds the notion of *objects* (think Lagrangean duality):

- **primal objects** or **variables** have associated
  - *value*: must lie between the primal object’s *bounds*.
  - *reduced cost*: the partial derivative of the objective function.

- **dual objects** or **constraints** are functions of the primal objects and have associated
  - *value*: the value of the Lagrange multiplier.
  - *slack*: must lie between the dual object’s *bounds*.

- **objective**: a function of the primal objects (dual object without bounds).
- **primal** and **dual solutions**: the objects with their associated values.
- *Note that primal and dual objects can be handled symmetrically.*
**BiCePS: Processing a Subproblem**

A subproblem is a set of objects with an objective.

Processing a subproblem

- solve a relaxation.
- generate new objects.
- tighten bounds.
- remove objects with value 0.

If all else fails or when desired, **branch**.
BiCePS: Branching

**Traditional branching:** Choose $\hat{x}_j$ fractional.

Children: $x_j \leq \lfloor \hat{x}_j \rfloor$ and $x_j \geq \lceil \hat{x}_j \rceil$.

**General branching:**

- **Add** new objects.
- **Change** object bounds.
- **Children** must cover the feasible region.

**Example:** $y_i$ binary variable and $y_i = 0 \Rightarrow a^T x \leq \beta$.

Children: $y_i = 1$ and $\{y_i = 0$ and $a^T x \leq \beta\}$.

*(this avoids using the big $M$ method)*

**Strong branching:**

- **“Pre-solve”** with multiple candidates to estimate bound.
- **Pick** the best.
BLIS: BiCePS Linear Integer Solver

A concretization of BiCePS specifying the bounding relaxation to be used.

Defines:

- the relaxation to be used: LP relaxation.
- the dual objects: linear functions.
- the realization of the objects: columns/rows of a matrix.

Leaves the notion of the relaxation solver abstract by using Open Solver Interface.
BLIS: Generating the Objects

- We need to define methods for generating the primal and dual objects.
- For constraints, such a method takes the primal solution vector and a list of the active variables and generates a row of the current matrix.
- For variables, we take the dual solution vector and a list of the active constraints and generate a column of the current matrix.
- We can also use object pools to help with generation.
- Note that we are working with various projections of the full polyhedron.
Scalability

- **Parallel System**: Parallel algorithm + parallel architecture [Kumar and Gupta ’94].

- **Scalability**: How well a parallel system takes advantage of increased computing resources.

- **Fixed problem size**: Efficiency decreases with more processors (Amdahl’s Law) [Amdahl ’67].

- **Fixed number of processors**: Efficiency increases with problem size.

- **Isoefficiency**: The rate problem size must be increased to maintain a fixed efficiency [Kumar and Rao ’87].
Scalability Issues for Parallel Search

- Grain size
- Decentralization
- Synchronous vs. asynchronous messaging
- Ramp-up/ramp-down time
Scalability: Increased Granularity

Work unit is a subtree.

Advantages:

• less communication.
• more compact storage via differencing.

Disadvantage:

• load balancing is more difficult.
Scalability: Master - Hubs - Workers Paradigm

Master

- has global information (node quality and distribution).
- balances load between hubs.
- balances quantity and quality.

Hubs

- manage collections of subtrees (may not have full descriptions)
- balances load between workers

Workers

- processes one subtree.
- hub can interrupt.
- sends branch and quality information to hub.
Scalability: Master - Hubs - Workers Paradigm
Scalability: Asynchronous Messaging

Possible communication bottlenecks:

- **Too many messages.**
  - avoided by the increased task granularity.
  - master-hub-worker paradigm also contributes.

- **Too much synchronization (handshaking)**
  - almost no handshaking.
  - must take place when a worker finishes exploring a subtree.
Scalability: Ramp-up/Ramp-down

- **Ramp-up time**: Time until all processors have useful work to do.
- **Ramp-down time**: Time during which there is not enough work for all processors.
- **Controlling Ramp-up/ramp-down**
  - use different branching rules.
  - hub instructs workers when to change rules.
Data Handling Issues

• Focused on data-intensive applications.
• Need to deal with huge numbers of objects.
• Need compact storage.
• Need to avoid duplication (generation and storage).
Data Handling: Object Representation

Each object has three representations:

• the user’s representation.
  – information to generate the realization.
  – core, indexed, or algorithmic.

• the realization in the solver.
  – (projected) matrix row
  – (projected) matrix column

• the encoded representation.
  – for identification and transfer between processors.
Data Handling: Encoding / Decoding

**Encodable objects:** Any object that is sent between processes.

**Question:** How to encode/decode objects the framework does not know about?

**Encoding:** Easy ⇒ use virtual methods.

**Decoding:** Catch-22

- can’t invoke constructor for unknown type.
- can’t invoke decode method without an object.

**Solution:** “Register” encodable objects.
Data Handling: Tracking Lists of Objects

Goal is memory conservation: no unnecessary object storage.

Implementation:

1. Object arrives in encoded form with hash value.
2. Object is looked up in hash map.
3. If it does not exist, then it is inserted.
4. A pointer to the unique copy in the hash map is added to the list.

Primary uses:

- storing variables and constraints—object can be active in multiple search nodes, but only one copy will be stored.
- object pools.
Data Handling: Object Pools

- Share objects across nodes in the tree.
- **Object pools** allow generated objects to be shared.
  - Cut pool is one example.
- **Quality measures** (slack or reduced cost, tree level, touches) ensure only the most effective objects are utilized.
- **Object encoding** is used to ensure that objects are stored only once.
Enhancements

- Domain decomposition
- Multi-phase methods
- Fault tolerance
- On-the-fly reconfiguration
Domain Decomposition

Useful when primal and dual objects can be partitioned such that

- primal objects of a group do not interact with dual objects of other groups.
- the objective object is $f$-separable in terms of the primal variables in the groups. $f$ can be additive (traditional MILP, stochastic programming), multiplicative (MIQP), etc.

After decomposing solve the children recursively, propagating intermediate results (thus creating new upper bound or fathoming the whole subtree).

$$[l, u] = [f(l_1, l_2, l_3), f(u_1, u_2, u_3)]$$
Multi-phase Methods

- **Solve** the problem on a *subset of the variables*, resulting in
  - a good upper bound,
  - a collection of good cuts,
  - an explored search tree.

- **Price** the remaining variables and propagate survivors down the tree, repeating pricing periodically.

- Unfathomed leaves are entered into the candidate list for the next phase.

- **Difficulty**: reproducing the search tree.
Fault Tolerance

Recovering from process failure (hardware or software)

- **Worker** ⇒ easy.
  - only the processed subtree is lost.
  - the hub can reassign the work.

- **Object pool** ⇒ easy.
  - has no effect on correctness.
  - can be restarted.

- **Hub** ⇒ hard.
  - all managed subtrees are lost.
  - other hubs must discard subtrees whose parents were on the dead hub.
  - workers must be reassigned to another hub.

- **Master**
  - can attempt to restart from saved data.
On-the-fly Reconfiguration

- On the fly reconfiguration (planned “fault”) — restricted use of processors.

- Messages from external source instructing master to
  - offload work from a process and kill it—work is redistributed.
  - to start new processes and redistribute work.
What’s Available

- **SYMPHONY**: C library for implementing BCP
  - User fills in stub functions.
  - Supports shared or distributed memory.

- **COIN/BCP**: C++ library for implementing BCP
  - User derives classes from library.
  - Documentation and source code available www.coin-or.org.

- **ALPS/BiCePS/BLIS**
  - In early development.
  - Sequential version by year end.

- The **COIN-OR** repository www.coin-or.org
Applications

• **SYMPHONY** has been used for various combinatorial problems:
  – Traveling Salesman Problem
  – Vehicle Routing Problem
  – Capacitated Network Routing
  – Airline Crew Scheduling

• **SYMPHONY** is also being adapted for constraint programming.

• **COIN/BCP** has also been used for combinatorial problems
  – Minimum Weight Steiner Tree
  – Max Cut
  – Multi-knapsack with Color Constraints

• We have a number of new applications under development.
The COIN-OR Project

- Supports the development of interoperable, open source software for operations research.
- Maintains a CVS repository for open source projects.
- Supports peer review of open source software as a supplement to the open literature.
- Software and documentation is freely downloadable from www.coin-or.org