Software Tools for Implementing Branch, Cut, and Price Algorithms

László Ladányi
IBM T. J. Watson Research Center

Ted Ralphs
Lehigh University

Matthew Saltzman
Clemson University

INFORMS Annual Meeting, Wednesday, November 20, 2002

Agenda

- Overview of COIN-OR
- Overview of branch, cut, and price
- Overview of COIN-OR branch, cut, and price toolbox
  - BCP
  - OSI
  - CGL
  - CLP
  - VOL
- Using the toolbox
  - Getting started
  - Developing an application
- Examples

Agenda

- ⇒ **Overview of COIN-OR** ⇐
- Overview of branch, cut, and price
- Overview of COIN-OR branch, cut, and price toolbox
  - BCP
  - OSI
  - CGL
  - CLP
  - VOL
- Using the toolbox
  - Getting started
  - Developing an application
- Examples

What is COIN-OR?

- The COIN-OR Project
  - A consortium of researchers in both industry and academia dedicated to improving the state of computational research in OR.
  - An initiative promoting the development and use of interoperable, open-source software for operations research.
- The COIN-OR Repository
  - A library of interoperable software tools for building optimization codes, as well as a few stand alone packages.
  - A venue for peer review of OR software tools.
  - A development platform for open source projects, including a CVS repository.

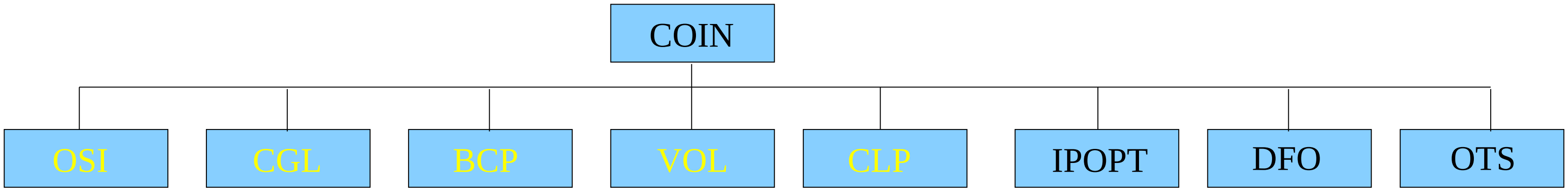
Our Agenda

- Accelerate the pace of research in computational OR.
  - Reuse instead of reinvent.
  - Reduce development time and increase robustness.
  - Increase interoperability (standards and interfaces).
- Provide for software what the open literature provides for theory.
  - Peer review of software.
  - Free distribution of ideas.
  - Adherence to the principles of good scientific research.
- Define standards and interfaces that allow software components to interoperate.
- Increase synergy between various development projects.
- Provide robust, open-source tools for practitioners.

Open Source Development

- *Open source development* is a coding paradigm in which development is done in a cooperative and distributed fashion.
- Strictly speaking, an open source license must satisfy the requirements of the *Open Source Definition*.
- A license cannot call itself "open source" until it is approved by the Open Source Initiative.
- Basic properties of an open source license
  - Access to source code.
  - The right to redistribute.
  - The right to modify.
- The license may require that modifications also be kept open.

Components of the COIN-OR Library



- Branch, cut, price toolbox
  - OSI: Open Solver Interface
  - CGL: Cut Generator Library
  - BCP: Branch, Cut, and Price Library
  - VOL: Volume Algorithm
  - CLP: COIN-OR LP Solver
- Stand-alone components
  - IPOPT: Interior Point Optimization (Nonlinear)
  - DFO: Derivative Free Optimization
  - OTS: Open Tabu Search

Agenda

- Overview of COIN-OR
- ⇒ **Overview of branch, cut, and price** ⇐
- Overview of COIN-OR branch, cut, and price toolbox
  - BCP
  - OSI
  - CGL
  - CLP
  - VOL
- Using the toolbox
  - Getting started
  - Developing an application
- Examples

Introduction to Branch, Cut, and Price

- Consider problem P:

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

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

- 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.

Branch, Cut, and Price

- Weyl-Minkowski
  - $\exists \bar{A} \in \mathbb{R}^{\bar{m} \times n}, \bar{b} \in \mathbb{R}^{\bar{m}}$ 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 $\mathcal{V} \subseteq \{1, \ldots, n\}$ and $\mathcal{C} \subseteq \{1, \ldots, \bar{m}\}$.
  - 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 $\Rightarrow$ faster LP solution.
  - Big LP relaxations $\Rightarrow$ 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 that generates coefficients for the *realization* of an inequality for the current set of active variables.
  - Variable: A mapping that generates coefficients corresponding to a variable for the current set of active constraints.
  - Subproblem: Defined by a subset of the global set of variables and constraints.
- To construct a subproblem, an initial *core relaxation* is needed.
- From the core, we can build up other relaxations using the mappings.

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

$$g^c(x) : \mathbb{R}^n \to 2^{\{1,\ldots,\bar{m}\}}$$

where x is a primal solution vector.

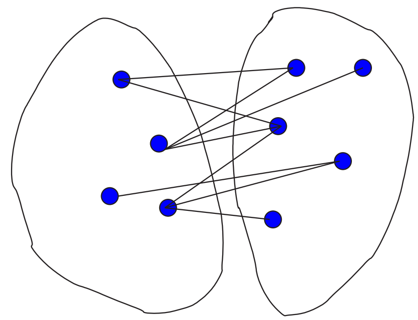
- For variables, we have

$$g^v(y) : \mathbb{R}^m \to 2^{\{1,\ldots,n\}}$$

where y is a dual solution vector.

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.
- Example: Subtour elimination constraints.



Example: Traveling Salesman Problem

Feasible solutions are those incidence vectors satisfying:

$$\begin{array}{rcl} \sum_{j=1}^{n} x_{ij} & = & 2 \quad \forall i \in V \\ \sum_{\substack{i \in S \\ j \notin S}} x_{ij} & \geq & 2 \quad \forall S \subset V,\ |S| > 1. \end{array}$$

- 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.

Frameworks for BCP

- 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
- Existing BCP frameworks
  - SYMPHONY (parallel, C)
  - COIN/BCP (parallel, C++)
  - ABACUS (sequential, C++)
- Tools Needed
  - Framework
  - LP Solver
  - Cut Generator

Agenda

- Overview of COIN-OR
- Overview of branch, cut, and price
- $\Rightarrow$ **Overview of COIN-OR branch, cut, and price toolbox** $\Leftarrow$
  - BCP
  - OSI
  - CGL
  - CLP
  - VOL
- Using the toolbox
  - Getting started
  - Developing an application
- Examples

The COIN-OR Branch, Cut, and Price Toolbox

Branch, cut, price toolbox

- BCP: Tool for implementing BCP algorithms
- OSI: Tool for interfacing to third-party solvers (particularly LP solvers)
- CGL: Tool for generating valid inequalities (within BCP)
- VOL: Fast approximate LP solver (with OSI interface)
- CLP: COIN-OR LP Solver (with OSI interface)

COIN/BCP Overview

- The COIN/BCP library is divided into modules, of which there are four basic 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.

  Node Processor Perform processing and branching operations.

  Object Generator Generate objects.

- The division into separate modules makes the code highly configurable and parallelizable.

COIN/BCP Overview: Node Processor

Handling of Constraints

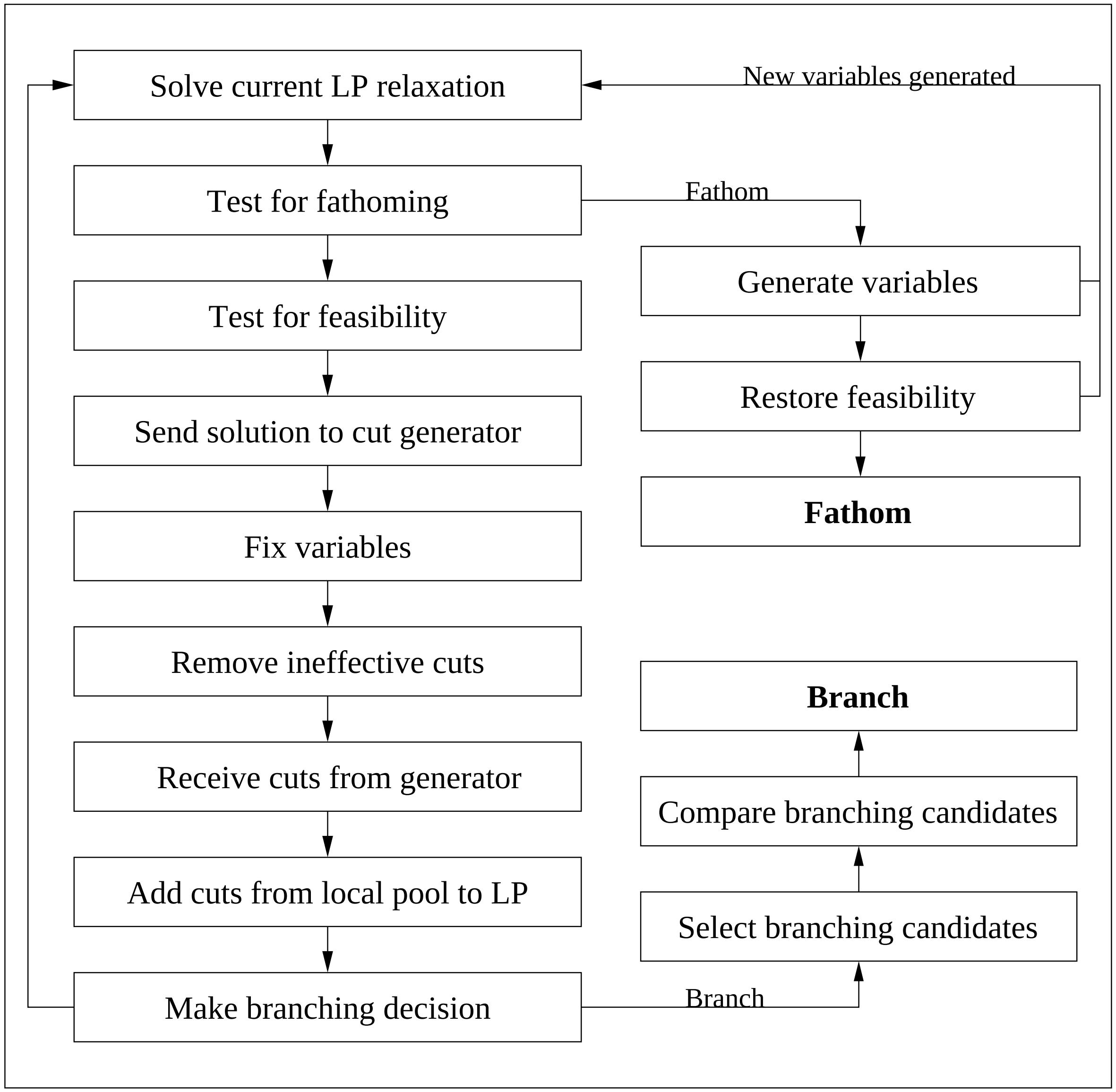
- Cuts are generated by the cut generators and/or by the node processor itself.
- Violated cuts are received and processed by the LP modules.
- Each LP module maintains a small local cut pool.
- A limited number of cuts are added to the LP relaxations each iteration to prevent “saturation.”
- Ineffective (non-core) cuts are aggressively removed.

COIN/BCP Overview: Node Processor

Handling of Variables

- Reduced cost/logical fixing are used to remove (non-core) variables.
- Variable generation may be needed for very large problems.
- 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.

COIN/BCP Overview: Node Processor Main Loop



OSI Overview

Uniform interface to LP solvers, including:

- CLP (COIN-OR LP Solver)
- CPLEX (ILOG)
- DyLP (BonsaiG LP Solver)
- GLPK (GNU LP Kit)
- OSL (IBM)
- SoPlex (Konrad-Zuse-Zentrum für Informationstechnik Berlin)
- Volume (COIN-OR)
- XPRESS (Dash Optimization)

CLP Overview

- Open source (Common Public License).
- Distributed with COIN-OR branch-cut-price toolbox.
- Reasonably robust and fast.

VOL Overview

- Generalized subgradient optimization algorithm.
- Compatible with branch, cut, and price:
  - provides approximate primal and dual solutions,
  - provides a valid lower bound (feasible dual solution), and
  - provides a method for warm starting.

CGL Overview

- Collection of cut generating methods integrated with OSI.
- Intended to provide robust implementations, including computational tricks not usually published.
- Currently includes:
  - Simple rounding cut
  - Gomory cut
  - Knapsack cover cut
  - Rudimentary lift-and-project cut (covering and packing)
  - Odd hole cut
  - Probing cut

Agenda

- Overview of COIN-OR
- Overview of branch, cut, and price
- Overview of COIN-OR branch, cut, and price toolbox
  - BCP
  - OSI
  - CGL
  - CLP
  - VOL
- ⇒ **Using the toolbox** ⇐
  - Getting started
  - Developing an application
- Examples

Getting Started with COIN-OR

- The source can be obtained from `www.coin-or.org` as “tarball” or using CVS.
- Platforms/Requirements
  - Linux, gcc 2.95.3/2.96RH/3.2
  - Windows, Visual C++, CygWin make (untested)
  - Sun Solaris, gcc 2.95.3/3.2 or SunWorkshop C++
- Editing the Makefiles
  - `Makefile.location`
  - `Makefile.<operating system>`
- Make the necessary libraries. They’ll be installed in `${CoinDir}/lib`.
  - Change to appropriate directory and type `make`.

Developing an Application

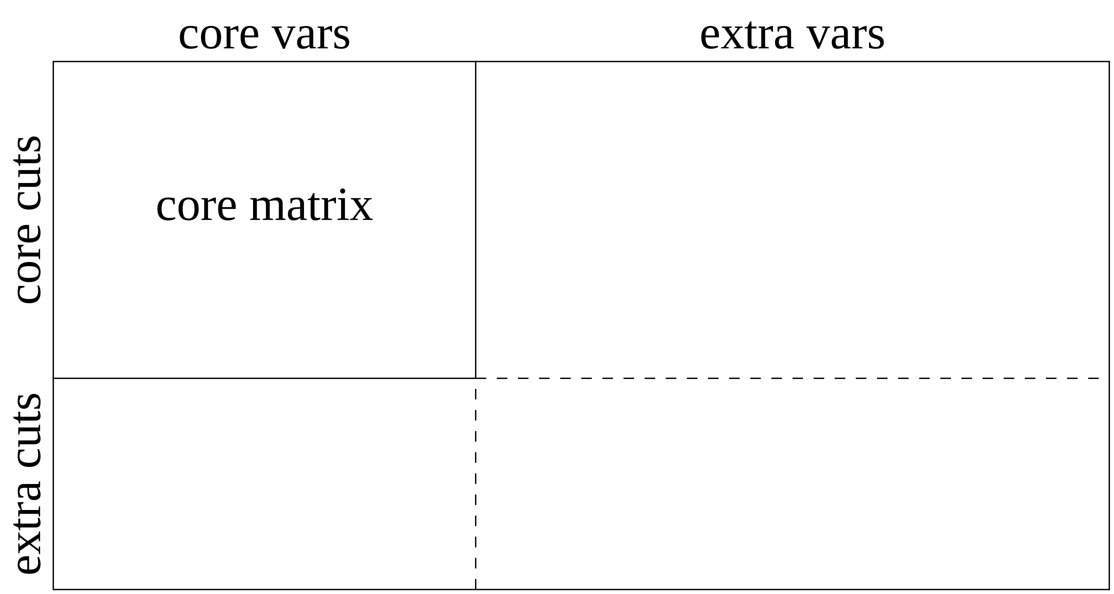
- The user API is implemented via a C++ class hierarchy.
- To develop an application, the user must derive the appropriate classes override the appropriate methods.
- Classes for customizing the behavior of the modules
  - BCP_tm_user
  - BCP_lp_user
  - BCP_cg_user
  - BCP_vg_user
- Classes for defining user objects
  - BCP_cut
  - BCP_var
  - BCP_solution
- Allowing COIN/BCP to create instances of the user classes.
  - The user must derive the class USER_initialize.
  - The function BCP_user_init() returns an instance of the derived initializer class.

Objects in COIN/BCP

- Most application-specific methods are related to handling of objects.
- Since representation is independent of the current LP, the user must define methods to add objects to a given subproblem.
- For parallel execution, the objects need to be packed into (and unpacked from) a buffer.
- Object Types
  - Core objects are objects that are active in *every* subproblem (BCP_xxx_core).
  - Indexed objects are extra objects that can be uniquely identified by an index (such as the edges of a graph) (BCP_xxx_indexed).
  - Algorithmic objects are extra objects that have an abstract representation (BCP_xxx_algo).

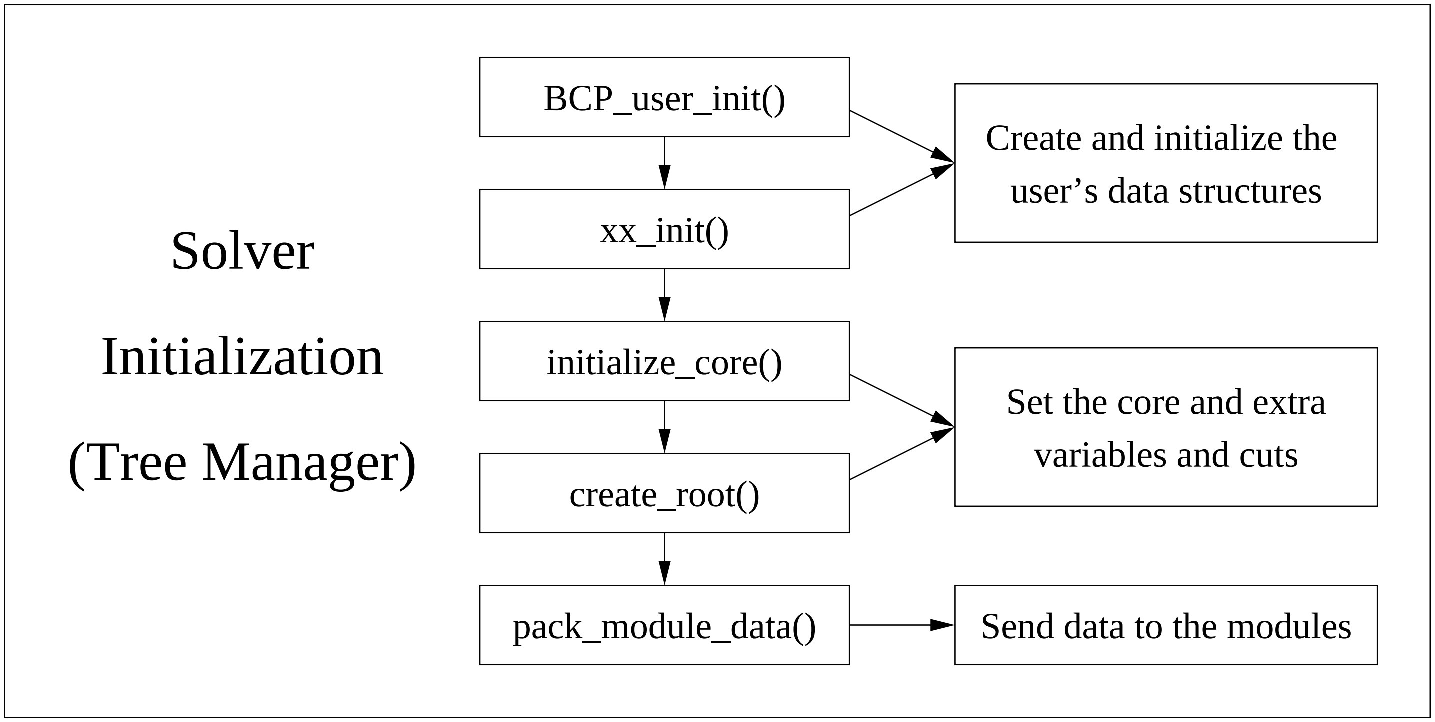
Forming the LP Relaxations in COIN/BCP

The current LP relaxation looks like this:

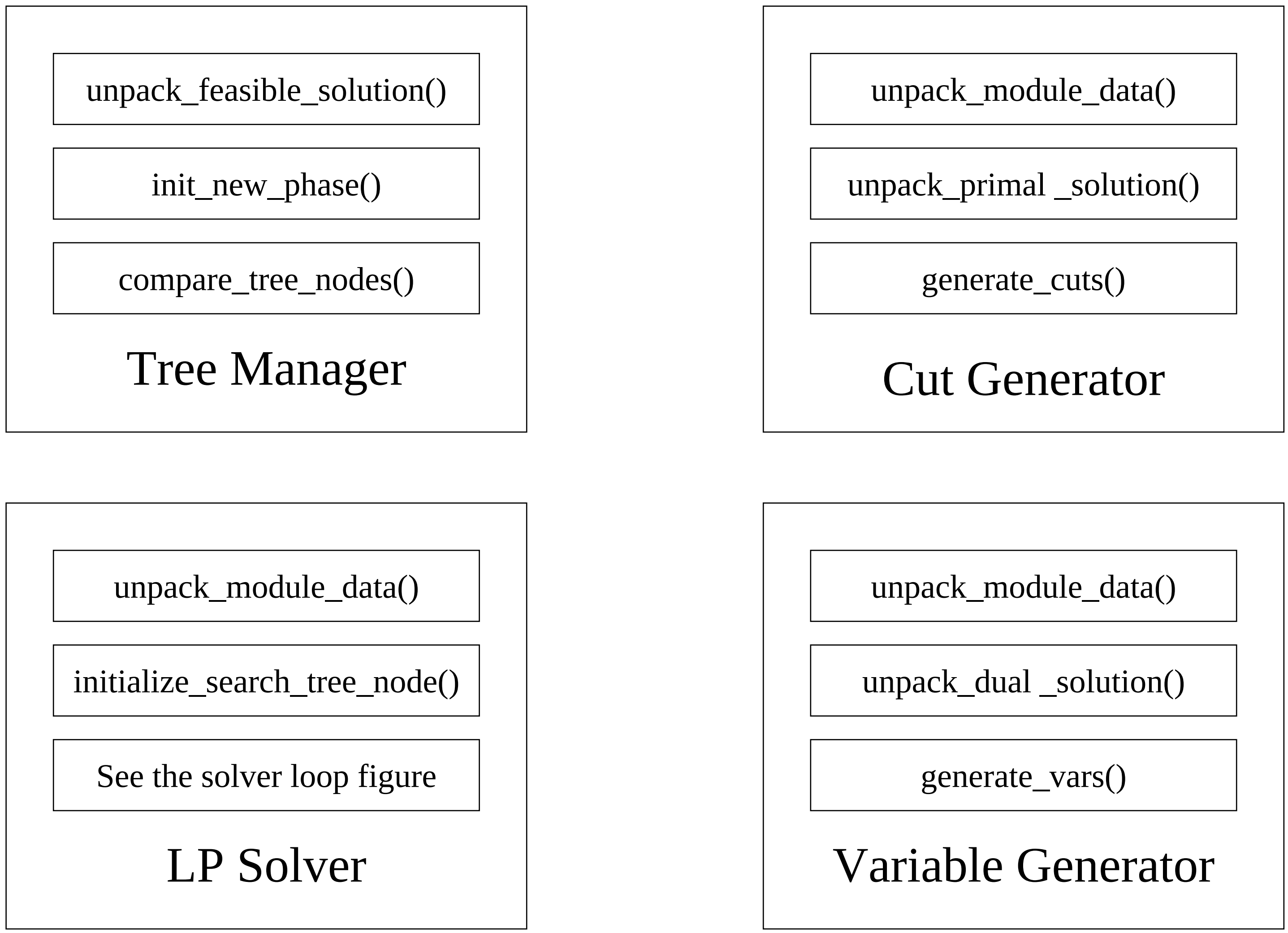


Reason for this split: efficiency.

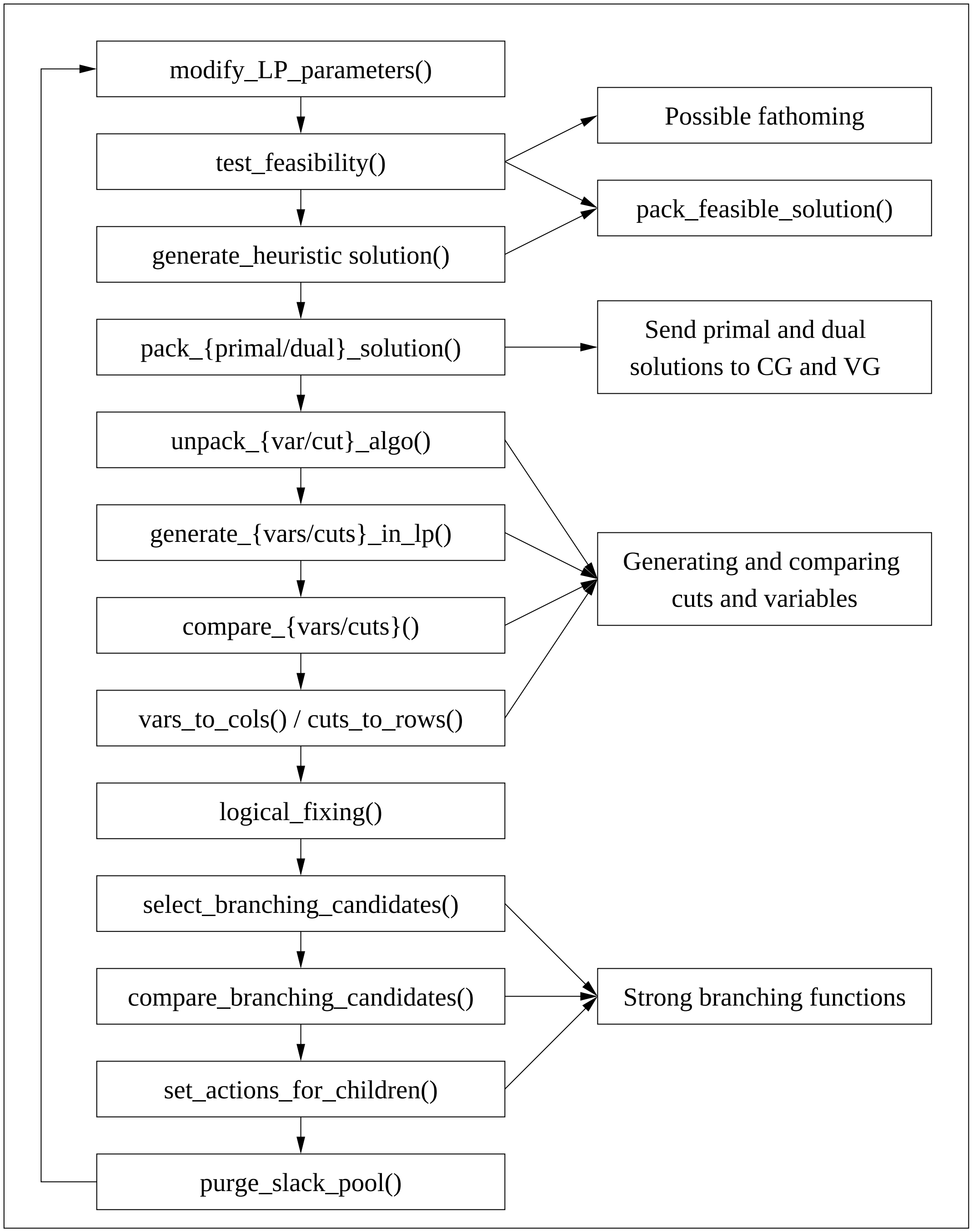
COIN/BCP Methods: Initialization



COIN/BCP Methods: Steady State



COIN/BCP Methods: Node Processing Loop



COIN/BCP Methods: Node Processing Loop

Var generation	B&B	Cut generation
	initialize_new_search_tree_node	
	modify_lp_parameters	
	solve the LP relaxation	
	display_lp_solution	
	test_feasibility	
compute_lower_bound		
pack_dual_solution		pack_primal_solution
	reduced cost fixing	
	logical_fixing	
generate_vars_in_lp		generate_cuts_in_lp
	generate_heuristic_solution	
	select_branching_candidates	
	If branching is decided on	
	compare_branching_candidates	
	set_actions_for_children	
vars are added	*Otherwise*	*cuts are added*

Misc methods in the TM (no particular order)

- unpack_feasible_solution. Default: Unpack a generic solution. Write if own solution type used.
- display_feasible_solution. Default: display a generic solution. Write if own solution type used.
- (un)pack_warmstart. Default: handles all warmstarts defined in Osi. Unlikely to write.
- (un)pack_var_algo. Write if algorithmic vars are generated.
- (un)pack_cut_algo. Write if algorithmic cuts are generated.
- compare_tree_nodes. Defaults: Breadth/Depth/Best First Search. Unlikely to write (only if none of the defaults are satisfactory).
- init_new_phase. Unlikely to write (only if multiphase is used).

Misc methods in the LP (no particular order)

- pack_feasible_solution. Invoked whenever a feasible soln is found. Default: Pack a generic solution.
  Write if own solution type used.
- (un)pack_warmstart. Default: handles all warmstarts defined in Osi.
  Unlikely to write.
- (un)pack_var_algo.
  Write if algorithmic vars are generated.
- (un)pack_cut_algo.
  Write if algorithmic cuts are generated.
- cuts_to_rows and vars_to_cols.
  Write if any sort of cut/var generation is done.
- compare_cuts and compare_vars.
  Write if any sort of cut/var generation is done.

Parameters and using the finished code

- Create a parameter file
- Run your code with the parameter file name as an argument (command line switches will be added).
- `BCP_` for BCP's parameters
- Defined and documented in BCP_tm_par, BCP_lp_par, etc.
- Helper class for creating your parameters.
- Output controlled by verbosity parameters.

Agenda

- Overview of COIN-OR
- Overview of branch, cut, and price
- Overview of COIN-OR branch, cut, and price toolbox
  - BCP
  - OSI
  - CGL
  - CLP
  - VOL
- Using the toolbox
  - Getting started
  - Developing an application
- ⇒ **Examples** ⇐

Example: Uncapacitated Facility Location

User classes and methods

- UFL_init
  - `tm_init()`
  - `lp_init()`
- UFL_lp
  - `unpack_module_data()`
  - `pack_cut_algo()`
  - `unpack_cut_algo()`
  - `generate_cuts_in_lp()`
  - `cuts_to_rows()`
- UFL_tm
  - `read_data()`
  - `initialize_core()`
  - `pack_module_data()`
- UFL_cut

Example: Generic MIP solver

- Implement branch and cut to solve an IP by
  - reading in from an MPS file,
  - designating all vars as core vars,
  - selecting some of the constraints as core constraints
  - making the rest extra (indexed) constraints, and
  - interfacing to CGL to generate cuts.
- Classes and methods are similar to the previous example.

Final advice

Use the source, Luke...

...and feel free to ask questions either by email or on the discussion list.