sbb

COIN-OR Simple Branch-and-Cut

John Forrest

Lou Hafer

lou@cs.sfu.ca

Outline

- Basic Branch-and-Bound
- Branching
- Node Selection
- Cuts and Heuristics
- Using sbb

Basic Branch-and-Bound

```
solve initial LP relaxation ;
prep relaxation for branching ;
add node to search tree ;
while (search tree is not empty)
{ pop a node and rebuild relaxation ;
  execute branch actions to restrict relaxation ;
  reoptimise restricted relaxation ;
  if (solution)
  { remember solution ; }
  else
  { prep relaxation for branching ;
    add node to search tree ; } }
extract solution ;
```

Branching Concepts

Branching in sbb is based on the notion of a *branching object*. In the abstract, a branching object

- Has a set of *feasible regions*.
- Has a method to *evaluate the infeasibility* of a solution to a relaxation.
- Has a method to *execute branching actions* to restrict a relaxation.
- Has a method to *rank the desirability* of a given branch alternative *vs.* other branch alternatives.

Sbb provides several common branching methods.

Custom Branching Strategies

Custom branching strategies can be created:

- A custom branch object class is derived from `SbbObject`. It provides methods to evaluate the infeasibility of a relaxed solution, enforce a feasible solution, and construct a branching object.
- A class to execute branching actions is derived from `SbbBranchingObject`. It provides a method to execute actions that modify a relaxation to produce a restricted relaxation.
- A class to rank branch alternatives is derived from `SbbBranchDecision`. Given two branch alternatives, it must indicate which is the preferred alternative.

Node Selection Concepts

A ranking function is used to order the search tree as a heap of relaxations requiring further evaluation.

- The ranking function is an attribute of the tree.
- An opportunity is provided to modify the ranking function every 1000 nodes and whenever a new solution is discovered.

Sbb provides several common ranking functions.

Custom Node Selection

Custom node selection functions can be created. A custom ranking function is derived from `SbbCompareBase`. It should provide

- A method for ranking two nodes.
- A method which is called each time a new solution is discovered.
- A method which is called every 1000 nodes.

Cut and Hueristic Concepts

Sbb provides facilities for the use of cut generation and heuristic methods.

- Sbb maintains a vector of cut generators and a vector of heuristics. Each generator and heuristic carries information about when and how often it should be invoked.
- Cut generators and heuristics can be called as part of a loop which evaluates a relaxation. Each iteration generates cuts, applies heuristics, and reoptimises.
- In addition, cut generators can be called when a new solution is found. Heuristics can be called immediately before a relaxation is added to the search tree.

Custom Cuts and Heuristics

Custom cut generators and heuristics should conform to a standard interface.

- A cut generator is called with a pointer to an `OsiSolverInterface`, which contains the solution to the current relaxation. It is expected to return a collection of cuts in an `OsiCuts` object.
- A heuristic has access to the entire `SbbModel`, including the solver and search tree. It is expected to return a solution, or an estimate of the best possible objective.

Notionally, cut generators are external, independent of sbb, while heuristics can be more tightly coupled.

The COIN-OR Cut Generation Library

The COIN-OR Cgl is a library of cut generators. It includes

- Cut generators for simple rounding, clique, odd hole, Gomory, knapsack, and lift-and-project cuts.
- A probing method which investigates the effect of forcing unsatisfied integer variables to integer values. It can tighten bounds on integer and continuous variables.
- All cut generators provide a standard interface.

sbb — the program

COIN-OR provides a branch-and-cut application, also called sbb.

- Created by '`make unitTest`' in the Sbb directory.
- By default, includes probing, Gomory, knapsack, and odd hole cut methods from the Cgl, and a rounding heuristic.
- Supports solvers through the OSI. Can be built to allow choice of underlying solver at runtime.

Branch-and-Cut in Your Application

Here's a minimal branch-and-bound solver:

```
{ OsiSolverInterface *osi =
      new OsiDylpSolverInterface ;
  SbbModel *model = new SbbModel(*osi) ;
  model->solver()->readMps("p0033") ;
  model->initialSolve() ;
  model->branchAndBound() ;
  std::cout << "Best solution "
            << model->solver()->getObjValue()
            << std::endl ;
  return ; }
```