Introduction to BCP – MCF Example

Laszlo Ladanyi\textsuperscript{1}
François Margot\textsuperscript{2}

July 18, 2006

1: IBM T.J. Watson Research Center
2: Tepper School of Business, Carnegie Mellon University
BCP: Branch-Cut-Price

- Software for branch-and-cut-and-price
- Parallel code
- LP solver: Clp, Cplex, Xpress, ...
- Most flexible in COIN-OR
- Research code (no stand-alone executable)
**BCP: Branch-Cut-Price**

- Software for branch-and-cut-and-price
- Parallel code
- LP solver: Clp, Cplex, Xpress, ...
- Most flexible in COIN-OR
- Research code (no stand-alone executable)

BCP code split into four directories: (see coin-Bcp/Bcp/src)

- **include**: all header files
- **Tree Manager (TM)**: Maintain the LP associated with each node, manage cuts and variables
- **Node level operations (LP)**: cutting, branching, heuristics, fixing, column generation
- **Utilities (Member)**: code for interface between TM and LP, initialization
## Solver Initialization

<table>
<thead>
<tr>
<th>Tree Manager</th>
<th>Solver</th>
</tr>
</thead>
<tbody>
<tr>
<td>• read data</td>
<td></td>
</tr>
</tbody>
</table>
## Solver Initialization

<table>
<thead>
<tr>
<th>Tree Manager</th>
<th>Solver</th>
</tr>
</thead>
<tbody>
<tr>
<td>• read data</td>
<td></td>
</tr>
<tr>
<td>• pack module data</td>
<td></td>
</tr>
</tbody>
</table>
## Solver Initialization

<table>
<thead>
<tr>
<th>Tree Manager</th>
<th>Solver</th>
</tr>
</thead>
<tbody>
<tr>
<td>• read data</td>
<td></td>
</tr>
<tr>
<td>• pack module data</td>
<td>→ unpack module data</td>
</tr>
</tbody>
</table>

### Solver Initialization

<table>
<thead>
<tr>
<th>Tree Manager</th>
<th>Solver</th>
</tr>
</thead>
<tbody>
<tr>
<td>• read data</td>
<td>• unpack module data</td>
</tr>
<tr>
<td>• pack module data</td>
<td>• setup the LP solver</td>
</tr>
</tbody>
</table>
Processing a node

- select node

<table>
<thead>
<tr>
<th>Tree Manager</th>
<th>Solver</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• pack node LP data</td>
</tr>
<tr>
<td></td>
<td>• unpack node LP data</td>
</tr>
<tr>
<td></td>
<td>• solve</td>
</tr>
<tr>
<td></td>
<td>• generate cuts/vars</td>
</tr>
<tr>
<td></td>
<td>• branch</td>
</tr>
<tr>
<td></td>
<td>• create LP data for sons</td>
</tr>
<tr>
<td></td>
<td>• unpack node LP data for sons</td>
</tr>
<tr>
<td></td>
<td>← • pack node LP data for sons</td>
</tr>
<tr>
<td></td>
<td>• add sons to tree</td>
</tr>
</tbody>
</table>
## Processing a node

<table>
<thead>
<tr>
<th>Tree Manager</th>
<th>Solver</th>
</tr>
</thead>
<tbody>
<tr>
<td>select node</td>
<td></td>
</tr>
<tr>
<td>pack node LP data</td>
<td></td>
</tr>
</tbody>
</table>
### Processing a node

<table>
<thead>
<tr>
<th>Tree Manager</th>
<th>Solver</th>
</tr>
</thead>
<tbody>
<tr>
<td>• select node</td>
<td>• unpack node LP data</td>
</tr>
<tr>
<td>• pack node LP data</td>
<td>→</td>
</tr>
</tbody>
</table>

• unpack node LP data for sons
### Processing a node

<table>
<thead>
<tr>
<th>Tree Manager</th>
<th>Solver</th>
</tr>
</thead>
<tbody>
<tr>
<td>• select node</td>
<td>• unpack node LP data</td>
</tr>
<tr>
<td>• pack node LP data</td>
<td>• solve</td>
</tr>
<tr>
<td></td>
<td>• generate cuts/vars</td>
</tr>
<tr>
<td></td>
<td>• branch</td>
</tr>
<tr>
<td></td>
<td>• create LP data for sons</td>
</tr>
</tbody>
</table>
### Processing a node

<table>
<thead>
<tr>
<th>Tree Manager</th>
<th>Solver</th>
</tr>
</thead>
<tbody>
<tr>
<td>• select node</td>
<td>• unpack node LP data</td>
</tr>
<tr>
<td>• pack node LP data</td>
<td>• solve</td>
</tr>
<tr>
<td></td>
<td>• generate cuts/vars</td>
</tr>
<tr>
<td></td>
<td>• branch</td>
</tr>
<tr>
<td></td>
<td>• create LP data for sons</td>
</tr>
<tr>
<td></td>
<td>• pack node LP data for sons</td>
</tr>
<tr>
<td>Tree Manager</td>
<td>Solver</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>• select node</td>
<td>• unpack node LP data</td>
</tr>
<tr>
<td>• pack node LP data</td>
<td>• solve</td>
</tr>
<tr>
<td></td>
<td>• generate cuts/vars</td>
</tr>
<tr>
<td></td>
<td>• branch</td>
</tr>
<tr>
<td>• unpack node LP data for sons</td>
<td>• create LP data for sons</td>
</tr>
<tr>
<td></td>
<td>• pack node LP data for sons</td>
</tr>
</tbody>
</table>
## Processing a node

<table>
<thead>
<tr>
<th>Tree Manager</th>
<th>Solver</th>
</tr>
</thead>
<tbody>
<tr>
<td>• select node</td>
<td>• unpack node LP data</td>
</tr>
<tr>
<td>• pack node LP data</td>
<td>• solve</td>
</tr>
<tr>
<td></td>
<td>• generate cuts/vars</td>
</tr>
<tr>
<td></td>
<td>• branch</td>
</tr>
<tr>
<td>• unpack node LP data for sons</td>
<td>• create LP data for sons</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Types of Constraints/Variables:
- Core: present at all nodes
- Algorithmic: separation/generation algorithm
- Indexed: e.g. stored in a vector
BCP Constraints/Variables

Types of Constraints/Variables:

- Core: present at all nodes
- Algorithmic: separation/generation algorithm
- Indexed: e.g. stored in a vector

Algorithmic constraints and variables are local
BCP Constraints/Variables

Types of Constraints/Variables:
- Core: present at all nodes
- Algorithmic: separation/generation algorithm
- Indexed: e.g. stored in a vector

Algorithmic constraints and variables are local

Representation: Constraints are stored as ranged constraints:

\[ lb \leq ax \leq ub \]

with \( lb = -\text{DBL\_MAX} \) or \( ub = \text{DBL\_MAX} \) possible
Implementing a Column Generation Application

Member:
- Read input
- Implement variables

TM:
- Set up the LP at the root node
- Display of a solution

LP:
- Test feasibility of a solution
- Column generation method
- Computation of a lower bound
- Branching decision
Col. Gen. Example: Multicommodity Flow (MCF-1)

- Directed graph $G = (V, E)$
- $N$ commodities
- $(s^i, t^i)$: source-sink pair, $i = 0, \ldots, N - 1$
- $d^i$: supply/demand vector for $s^i t^i$ flow, $i = 0, \ldots, N - 1$
Col. Gen. Example: Multicommodity Flow (MCF-1)

- Directed graph \( G = (V, E) \)
- \( N \) commodities
- \((s^i, t^i)\): source-sink pair, \( i = 0, \ldots, N - 1 \)
- \( d^i \): supply/demand vector for \( s^i t^i \) flow, \( i = 0, \ldots, N - 1 \)

For each arc \( e \in E \):
- \( 0 \): lower bound for total flow on arc
- \( u_e \): finite upper bound for total flow on arc \( (0 \leq u_e) \)
- \( w_e \): unit cost \( (0 \leq w_e) \)
Solution:

- \( f^i \): \( s^i t^i \)-flow with supply/demand vector \( d^i \)
- \( \sum_i f_e^i \leq u_e \) for all \( e \in E \)
MCF: ILP Formulation

Solution:

• $f^i$: $s^i t^i$-flow with supply/demand vector $d^i$

• $\sum_i f^i_e \leq u_e$ for all $e \in E$

ILP Formulation:

$$\min \sum_i w^T f^i$$

$$\sum_i f^i \leq u$$ \hspace{1cm} (1)

$$\sum_{e=(v,w) \in E} f^i_e - \sum_{e=(w,v) \in E} f^i_e = d^i_v \hspace{0.5cm} \forall v \in V, \forall i$$ \hspace{1cm} (2)

$$0 \leq f^i \leq u \hspace{1cm} \forall i$$ \hspace{1cm} (3)

$f^i$ integral \hspace{1cm} $\forall i$ \hspace{1cm} (4)
Class MCF\_data (see Member/MCF\_data.hpp):

- **arcs**: vector of struct (tail, head, lb, ub, weight)
- **commodities**: vector of struct (source, sink, demand)
- **numarcs**
- **numnodes**
- **numcommodities**
- **Setup by MCF\_data::readDimacsFormat()**
**MCF: Input data**

Class MCF_data (see Member/MCF_data.hpp):

- **arcs**: vector of struct (tail, head, lb, ub, weight)
- **commodities**: vector of struct (source, sink, demand)
- **numarcs**
- **numnodes**
- **numcommodities**
- Setup by MCF_data::readDimacsFormat()

Parameter MCF_AddDummySourceSinkArcs: Add numcommodities dummy arcs with large weight to ensure feasibility
MCF: Master Problem

Master Problem:

- Column: $s^i t^i$-flow satisfying $d^i$ for some $i$
- $F^i$: matrix of all generated $s^i t^i$-flows (+ dummy flow)
- $\lambda^i$: multiplier for generated $s^i t^i$-flows
MCF: Master Problem

Master Problem:

- Column: $s^i t^i$-flow satisfying $d^i$ for some $i$
- $F^i$: matrix of all generated $s^i t^i$-flows (+ dummy flow)
- $\lambda^i$: multiplier for generated $s^i t^i$-flows

Example: all arcs upper capacity 2, source = 0, sink = 3, $d = 2$. 

\begin{array}{c|c}
01 & 0 \\
02 & 0 \\
12 & 0 \\
13 & 0 \\
23 & 0 \\
03 & 2 \\
\end{array}
Master Problem:

- Column: $s^i t^i$-flow satisfying $d^i$ for some $i$
- $F^i$: matrix of all generated $s^i t^i$-flows (+ dummy flow)
- $\lambda^i$: multiplier for generated $s^i t^i$-flows

Example: all arcs upper capacity 2, source = 0, sink = 3, $d = 2$.

01 | 0 1  
02 | 0 1  
12 | 0 1  
13 | 0 0  
23 | 0 1  
03 | 2 0  

Diagram of network flow with capacities and values.
Master Problem:

- Column : $s^i t^i$-flow satisfying $d^i$ for some $i$
- $F^i$ : matrix of all generated $s^i t^i$-flows (+ dummy flow)
- $\lambda^i$ : multiplier for generated $s^i t^i$-flows

Example: all arcs upper capacity 2, source = 0, sink = 3, $d = 2$.

\[
\begin{array}{ccc}
01 & 0 & 1 & 2 \\
02 & 0 & 1 & 0 \\
12 & 0 & 1 & 2 \\
13 & 0 & 0 & 0 \\
23 & 0 & 1 & 2 \\
03 & 2 & 0 & 0 \\
\end{array}
\]
MCF: Master Problem

\[
\begin{align*}
\min & \sum_i \mathbf{w}^T \mathbf{F}_i \lambda^i \\
\sum_i F^i \lambda^i & \leq u \\
e^T \lambda^i & = 1 \quad \forall i \\
\lambda^i & \geq 0 \quad \forall i \\
F^i \lambda^i & \text{ integer} \quad \forall i
\end{align*}
\]
MCF: Master Problem

\[
\min \sum_i w^T F^i \lambda^i
\]

\[
\sum_i F^i \lambda^i \leq u
\]  \hspace{1cm} (5)

\[
e^T \lambda^i = 1 \hspace{1cm} \forall i
\]  \hspace{1cm} (6)

\[
\lambda^i \geq 0 \hspace{1cm} \forall i
\]  \hspace{1cm} (7)

\[
F^i \lambda^i \text{ integer} \hspace{1cm} \forall i
\]  \hspace{1cm} (8)

\[
\begin{bmatrix} F^0 & F^1 & F^2 \end{bmatrix} \begin{bmatrix} \lambda^0 \\ \lambda^1 \\ \lambda^2 \end{bmatrix} \leq u
\]

\[
\begin{bmatrix} 1^T & \cdot & \cdot \\ \cdot & 1^T & \cdot \\ \cdot & \cdot & 1^T \end{bmatrix} \begin{bmatrix} \lambda^0 \\ \lambda^1 \\ \lambda^2 \end{bmatrix} = 1
\]
MCF: Master Problem

\[
\begin{align*}
\text{min} & \; \sum_i w^T F^i \lambda^i \\
\sum_i F^i \lambda^i & \leq u \quad (\pi) \tag{5} \\
e^T \lambda^i & = 1 \; \forall i \quad (\nu^i) \tag{6} \\
\lambda^i & \geq 0 \; \forall i \tag{7} \\
F^i \lambda^i & \text{ integer} \; \forall i \tag{8}
\end{align*}
\]

Pricing of feasible \( s^i t^i \)-flow \( f \):

- weight of flow: \( w^T f \)
- dual activity: \( \pi^T f + \nu^i \)

Reduced cost of flow \( f = w^T f - \pi^T f - \nu^i = (w^T - \pi^T)f - \nu^i \)
Class MCF-vars

MCF_var:

- int commodity: index of commodity
- CoinPackedVector flow: positive flow on arcs
- weight: objective coefficient

See include/MCF_var.hpp, Member/MCF_var.cpp
Class MCF_vars

MCF_var:
  • int commodity: index of commodity
  • CoinPackedVector flow: positive flow on arcs
  • weight: objective coefficient

See include/MCF_var.hpp, Member/MCF_var.cpp

MCF_lp::vars_to_cols(): generate columns of the master problem for vars
MCF: Setting the Master at the Root

Variables:
- Dummy flow variables are algorithmic variables ($\lambda_i^0 \forall i$)
- All generated variables are algorithmic

See in TM/MCF_tm.cpp:
MCF_tm::initialize_core
MCF_tm::create_root
MCF: Setting the Master at the Root

Variables:

- Dummy flow variables are algorithmic variables ($\lambda_i^0 \forall i$)
- All generated variables are algorithmic

Constraints:

- All constraints are core constraints
- Upper bound constraints: $0 \leq u_e \forall e \in E$
- Dummy upper bound constraints: $dem(i)\lambda_0^i \leq dem(i) \forall i$
- Convexity constraints: $\lambda_0^i = 1 \forall i$

See in TM/MCF_tm.cpp:
MCF_tm::initialize_core
MCF_tm::create_root
Class MCF_tm: Derived from BCP_tm_user

Data:
  - MCF_data data

Methods:
  - pack_module_data(): pack data needed at the node level. Called once for each processor used as a solver.
  - initialize_core(): Transmit core constraints/variables to BCP.
  - create_root: set up the problem at the root node
  - pack_var_algo(): pack algorithmic vars
  - unpack_var_algo(): unpack algorithmic vars
  - display_feasible_solution(): display solution
Node operations

1. Initialize new node
2. Solve node LP
3. Test feasibility of node LP solution
4. Compute lower bound for node LP
5. Fathom node (if possible)
6. Perform fixing on vars
7. Update row effectiveness records
8. Generate cuts, Generate vars
9. Generate heuristic solution
10. Fathom node (if possible)
11. Decide to branch, fathom, or repeat loop
12. Add to node LP the cuts/vars generated, if loop is repeated
13. Purge cut pool, var pool
Class MCF_LP: Derived from BCP_lp_user

Data:
- OsiSolverInterface* cg_lp: pointer on Osi LP solver used for column generation
- MCF_data data: problem data
- vector<MCF_branch_decision>* branch_history: branch_history[i]: vector of branching decision involving commodity i (arc, lb, ub)
- map<int, double>* flows: flows[i]: map between index of arc and positive flow for commodity i in LP solution
- BCP_vec<BCP_var*> gen_vars: vector holding generated vars
- bool generated_vars: indicator for success in column generation

See LP/MCF_lp.cpp, include/MCF_lp.hpp
Class MCF_LP: Derived from BCP_lp_user (cont)

Methods:

- `unpack_module_data()`
- `pack_var_algo(), unpack_var_algo()`
- `initialize_new_search_tree_node()`: Natural place for initializing user defined variables of MCF_lp.
- `test_feasibility()`: Test feasibility of current LP solution.
- `compute_lower_bound()`: Lower bound on optimal value of subproblem
- `generate_vars_in_lp()`: Pass new variables to BCP
- `vars_to_cols()`: Function generating a column from the var representation
- `select_branching_candidates()`: Generate rules for creating potential sons
Initially, lower bound of a node is set to the lower bound of its father

- Try to generate a variable with negative reduced cost
- If successful, lower bound is currently known lower bound
- If unsuccessful, lower bound is the current LP value

See `MCF_lp::compute_lower_bound()` in LP/MCF_lp.cpp
\textbf{MCF: Column Generation}

- $\pi, \nu^i$: optimal dual solution of the Master

(see \texttt{MCF\_lp::compute\_lower\_bound} in \texttt{LP/MCF\_lp.cpp})
MCF: Column Generation

• $\pi, \nu^i$: optimal dual solution of the Master

Column generation:

$$\min (w^T - \pi^T) f^i - \nu^i$$

$$\sum_{e=(w,v) \in E} f_e^i - \sum_{e=(w,v) \in E} f_e^i = d_v^i \quad \forall v \in V$$

(9)

$$\ell^i \leq f^i \leq u^i$$

(10)

$f^i$ integral

(11)

If solution is negative, then $f^i$ is the new column

(see MCF::compute_lower_bound in LP/MCF_lp.cpp)
MCF: Column Generation

- \( \pi, \nu^i \): optimal dual solution of the Master

Column generation:

\[
\begin{align*}
\min (w^T - \pi^T) f^i \\
\sum_{e=(w,v) \in E} f^i_e - \sum_{e=(w,v) \in E} f^i_e = d^i_v \quad \forall v \in V \\
\ell^i \leq f^i \leq u^i
\end{align*}
\] 

(9)

If solution is \( < \nu^i \), then \( f^i \) is the new column

(see MCF_lp::compute_lower_bound in LP/MCF_lp.cpp)
MCF: Column Generation

- $\pi, \nu^i$ : optimal dual solution of the Master

Column generation:

$$\min (w^T - \pi^T) f^i$$

$$\sum_{e=(w,v) \in E} f_e^i - \sum_{e=(w,v) \in E} f_e^i = d_v^i \quad \forall v \in V$$  \hspace{1cm} (9)

$$\ell^i \leq f^i \leq u^i$$ \hspace{1cm} (10)

$f^i$ integral \hspace{1cm} (11)

If solution is $< \nu^i$, then $f^i$ is the new column

Minimum cost flow problem $\Rightarrow$ Solve as an LP
(see MCF.lp::compute_lower_bound in LP/MCF.lp.cpp)
Branching

MCF_lp::select_branching_candidates(): Called at the end of each iteration. Possible return values are:

- **BCP_DoNotBranch_Fathomed**: fathomed without branching
- **BCP_DoNotBranch**: continue to work on this node
- **BCP_DoBranch**: Branching must be done. Must create the candidates
MCF: Branching

- Solution of the Master is fractional
- No new column is generated

Branching rule:
- Select an arc $e$ (not dummy) and $i$ with $F_i \lambda_i = z$ fractional
- First child: Use only columns where flow of $i$ on $e$ is $> z$
- Second child: Use only columns where flow of $i$ on $e$ is $< z$

Need to know $\ell_i e$ and $u_i e$ for all $i$ and $e$ for col. gen.
**MCF: Branching**

- Solution of the Master is fractional
- No new column is generated

⇒ Must branch

**Branching rule:**
- Select an arc $e$ (not dummy) and $i$ with $F^i \lambda^i = z$ fractional
- First child: Use only columns where flow of $i$ on $e$ is $> z$
- Second child: Use only columns where flow of $i$ on $e$ is $< z$
**MCF: Branching**

- Solution of the Master is fractional
- No new column is generated

⇒ Must branch

Branching rule:
- Select an arc \( e \) (not dummy) and \( i \) with \( F^i \lambda^i = z \) fractional
- First child: Use only columns where flow of \( i \) on \( e \) is > \( z \)
- Second child: Use only columns where flow of \( i \) on \( e \) is < \( z \)
- Need to know \( \ell^i_e \) and \( u^i_e \) for all \( i \) and \( e \) for col. gen.
MCF: Branching

- Solution of the Master is fractional
- No new column is generated

⇒ Must branch

Branching rule:
- Select an arc $e$ (not dummy) and $i$ with $F_i^i \lambda^i = z$ fractional
- First child: Use only columns where flow of $i$ on $e$ is $> z$
- Second child: Use only columns where flow of $i$ on $e$ is $< z$
- Need to know $\ell^i_e$ and $u^i_e$ for all $i$ and $e$ for col. gen.
  ⇒ use branch\_history[i]
Class MCF_branching_var

MCF_branching_var:
- artificial variable used to keep branching history around
- weight 0
- coefficients 0
- upper: 1, lower 0: identify child

See include/MCF_var.hpp, Member/MCF_var.cpp
Class MCF_branching_var

MCF_branching_var:
- artificial variable used to keep branching history around
- weight 0
- coefficients 0
- upper: 1, lower 0: identify child

Data:
- commodity: commodity $i$ used in branching
- arc_index: arc $e$ used in branching
- $lb_{\text{child}0}$, $ub_{\text{child}0}$, $lb_{\text{child}1}$, $ub_{\text{child}1}$: bounds for commodity $i$ on $e$ in the children

See include/MCF_var.hpp, Member/MCF_var.cpp
Branching object

Create the candidates using:

BCP_lp_branching_object::BCP_lp_branching_object()

Its relevant parameters are:

- `int children`: # children
- `BCP_vec<BCP_var*>* new_vars`: vector for new vars
- `BCP_vec<BCP_cut*>* new_cuts`: vector for new cuts
- `BCP_vec<int>* fvp`: vector for indices of variables whose bounds are changed. Negative indices: vars from `new_vars`, index $-i-1$ corresponding to entry $i$
- `BCP_vec<int>* fcp`: vector for indices of cuts whose bounds are changed. Negative indices: cuts from `new_cuts`, index $-i-1$ corresponding to entry $i`
Create the candidates using:
`BCP_lp_branching_object::BCP_lp_branching_object()`

Its relevant parameters are:

- `int children`: # children
- `BCP_vec<BCP_var*> *new_vars`: vector for new vars
- `BCP_vec<BCP_cut*> *new_cuts`: vector for new cuts
- `BCP_vec<int> *fvp`: vector for indices of variables whose bounds are changed
  Negative indices: vars from `new_vars`, index \(-i - 1\) corresponding to entry \(i\)
- `BCP_vec<int> *fcp`: vector for indices of cuts whose bounds are changed.
  Negative indices: cuts from `new_cuts`, index \(-i - 1\) corresponding to entry \(i\)
Branching object MCF

Create the candidates using:

```cpp
BCP_lp_branching_object::BCP_lp_branching_object()
```

Its relevant parameters are:

- `int children`: # children 2
- `BCP_vec<BCP_var*> *new_vars`: vector for new vars
- `BCP_vec<BCP_cut*> *new_cuts`: vector for new cuts
- `BCP_vec<int> *fvp`: vector for indices of variables whose bounds are changed
  - Negative indices: vars from `new_vars`, index $-i-1$ corresponding to entry $i$
- `BCP_vec<int> *fcp`: vector for indices of cuts whose bounds are changed.
  - Negative indices: cuts from `new_cuts`, index $-i-1$ corresponding to entry $i`
Branching object **MCF**

Create the candidates using:

```cpp
BCP_lp_branching_object::BCP_lp_branching_object()
```

Its relevant parameters are:

- `int children`: # children 2
- `BCP_vec<BCP_var*>* new_vars`: vector for new vars
  - one new MCF_branching_var
- `BCP_vec<BCP_cut*>* new_cuts`: vector for new cuts
- `BCP_vec<int>* fvp`: vector for indices of variables whose bounds are changed
  - Negative indices: vars from `new_vars`, index $-i - 1$ corresponding to entry $i$
- `BCP_vec<int>* fcp`: vector for indices of cuts whose bounds are changed.
  - Negative indices: cuts from `new_cuts`, index $-i - 1$ corresponding to entry $i`
**Branching object MCF**

Create the candidates using:

```cpp
BCP_lp_branching_object::BCP_lp_branching_object()
```

Its relevant parameters are:

- `int children`: # children 2
- `BCP_vec<BCP_var*>* new_vars`: vector for new vars
  one new MCF_branching_var
- `BCP_vec<BCP_cut*>* new_cuts`: vector for new cuts NULL
- `BCP_vec<int>* fvp`: vector for indices of variables whose bounds are changed
  Negative indices: vars from `new_vars`, index $-i - 1$ corresponding to entry $i$
- `BCP_vec<int>* fcp`: vector for indices of cuts whose bounds are changed.
  Negative indices: cuts from `new_cuts`, index $-i - 1$ corresponding to entry $i`
Branching object MCF

Create the candidates using:

BCP lp branching object::BCP lp branching object()

Its relevant parameters are:

- int children: # children 2
- BCP_vec<BCP_var*> *new_vars: vector for new vars
  one new MCF_branching_var
- BCP_vec<BCP_cut*> *new_cuts: vector for new cuts NULL
- BCP_vec<int> *fvp: vector for indices of variables whose
  bounds are changed
  Negative indices: vars from new_vars,
  index \(-i - 1\) corresponding to entry \(i\) [-1, 4, 7]
- BCP_vec<int> *fcp: vector for indices of cuts whose
  bounds are changed.
  Negative indices: cuts from new_cuts,
  index \(-i - 1\) corresponding to entry \(i\)
Branching object \textit{MCF}

Create the candidates using:
\texttt{BCP\_lp\_branching\_object::BCP\_lp\_branching\_object()}

Its relevant parameters are:

- \texttt{int children} : \# children 2
- \texttt{BCP\_vec<BCP\_var*> \*new\_vars} : vector for new vars
  one new MCF\_branching\_var
- \texttt{BCP\_vec<BCP\_cut*> \*new\_cuts} : vector for new cuts \texttt{NULL}
- \texttt{BCP\_vec<int> \*fvp} : vector for indices of variables whose
  bounds are changed
  Negative indices : vars from \texttt{new\_vars},
  index \(-i-1\) corresponding to entry \(i\) \([-1, 4, 7]\)
- \texttt{BCP\_vec<int> \*fcp} : vector for indices of cuts whose
  bounds are changed.
  Negative indices : cuts from \texttt{new\_cuts},
  index \(-i-1\) corresponding to entry \(i\) \texttt{NULL}
Branching object (cont)

- `BCP_vec<double> *fvb`: vector for lower/upper bounds for each var in `fvp`, for each child

- `BCP_vec<double> *fcb`: vector for lower/upper bounds for each constraint in `fcp`, for each child

- 4 additional parameters (implied parts)

Pass NULL for irrelevant parameters
Branching object (cont) \textbf{MCF}

- \texttt{BCP\_vec<double> \*fvb} : vector for lower/upper bounds for each vars in \texttt{fvp}, for each child
  \[0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1\]
- \texttt{BCP\_vec<double> \*fcb} : vector for lower/upper bounds for each constraint in \texttt{fcp}, for each child
- 4 additional parameters (implied parts)

Pass NULL for irrelevant parameters
Branching object (cont) MCF

- BCP_vec<double> *fvb : vector for lower/upper bounds for each vars in fvp, for each child [0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1]
- BCP_vec<double> *fcb : vector for lower/upper bounds for each constraint in fcp, for each child NULL
- 4 additional parameters (implied parts)

Pass NULL for irrelevant parameters
Branching object (cont) MCF

- BCP_vec<double> *fvb : vector for lower/upper bounds for each vars in fvp, for each child [0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1]
- BCP_vec<double> *fcb : vector for lower/upper bounds for each constraint in fcp, for each child NULL
- 4 additional parameters (implied parts) NULL

Pass NULL for irrelevant parameters
Branching object: Forced vs. Implied

Forced changes:
- Used during strong branching
- Sent to the tree manager if branching object is selected
- Used in the children if branching object is selected

Implied changes:
- Used during strong branching
- NOT Sent to the tree manager if branching object is selected
- NOT Used in the children if branching object is selected

Many implied changes \Rightarrow storing them is costly.

If implied changes are used, implement them also in MCF lp::initialize new search tree node()
**Branching object: Forced vs. Implied**

Forced changes:
- Used during strong branching
- Sent to the tree manager if branching object is selected
- Used in the children if branching object is selected

Implied changes:
- Used during strong branching
- **NOT** Sent to the tree manager if branching object is selected
- **NOT** Used in the children if branching object is selected
Branching object: Forced vs. Implied

Forced changes:
- Used during strong branching
- Sent to the tree manager if branching object is selected
- Used in the children if branching object is selected

Implied changes:
- Used during strong branching
- **NOT** Sent to the tree manager if branching object is selected
- **NOT** Used in the children if branching object is selected

Many implied changes $\Rightarrow$ storing them is costly.
If implied changes are used, implement them also in 
`MCF::initialize_new_search_tree_node()`
Predefined parameters:
Class MCF_par (see include/MCF_par.hpp)
Class BCP_lp_par Class BCP_tm_par
Predefined parameters:
Class MCF_par (see include/MCF_par.hpp)
Class BCP_lp_par Class BCP_tm_par

Some parameters with their default values:
- MCF_AddDummySourceSinkArcs: 1
- MCF_InputFilename: small
- BCP_VerbosityShutUp: 0
- BCP_MaxRunTime: 3600
- BCP_Granularity: 1e-8
- BCP_IntegerTolerance: 1e-5
- BCP_TreeSearchStrategy: 1
  // 0: Best Bound 1: BFS 2: DFS