Branching, Presolving, and Constraint Handlers

CO@Work Berlin

Timo Berthold and Kati Wolter

09/26/2009

Berlin Mathematical School

DFG Research Center MATHEON
Mathematics for key technologies

Outline

Outline

How do we solve discrete optimization problems?

Mixed Integer Programming

- LP relaxation
- cutting planes
- branch-and-bound

SATisfiability problems

- conflict analysis
- periodic restarts
- branch-and-bound

Constraint Programming

- domain propagation
- symmetry handling
- branch-and-bound

How do we solve discrete optimization problems?

Mixed Integer Programming

- LP relaxation
- cutting planes
- branch-and-bound

SATisfiability problems

- conflict analysis
- periodic restarts
- branch-and-bound

Constraint Programming

- domain propagation
- symmetry handling
- branch-and-bound

Branch-and-bound

- Land, Doig [1960], Dakin [1965]
- "smart" enumeration
- use dual bounds

LP-based Branch-and-Bound (Algorithm)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching

$\mathcal{L} \leftarrow \{P\}, \quad U \leftarrow \infty, \quad x^{\mathrm{IP}} \leftarrow \mathrm{NULL}$

if $\mathcal{L} = \emptyset$ **then return** x^{IP} *and* U

Select $P_i \in \mathcal{L}, \quad \mathcal{L} \leftarrow \mathcal{L} \backslash \{P_i\}$

Solve LP relaxation of $P_i, \quad L_{\mathrm{loc}} \leftarrow c(x^{\mathrm{LP}})$ or ∞

if $L_{loc} \geq U$ **then goto** line 2

if $x^{LP} \in P$ **then** $x^{\mathrm{IP}} \leftarrow x^{\mathrm{LP}}, \quad U \leftarrow L_{\mathrm{loc}}, \quad$ **goto** line 2

Select $j \in I$: $x_j^{\mathrm{LP}} \notin \mathbb{Z}$. Split P_i into $P_{2i+1} := P_i \cup \{x_j \leq \lfloor x_j^{\mathrm{LP}} \rfloor\}$, and $P_{2i+2} := P_i \cup \{x_j \geq \lceil x_j^{\mathrm{LP}} \rceil\}, \quad \mathcal{L} \leftarrow \mathcal{L} \cup \{P_{2i+1}, P_{2i+2}\}, \quad$ **goto** line 2

LP-based Branch-and-Bound (Algorithm)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching

$\mathcal{L} \leftarrow \{P\}, \quad U \leftarrow \infty, \quad x^{\mathrm{IP}} \leftarrow \mathrm{NULL}$

if $\mathcal{L} = \emptyset$ **then return** x^{IP} *and* U

Select $P_i \in \mathcal{L}, \quad \mathcal{L} \leftarrow \mathcal{L} \setminus \{P_i\}$

Solve LP relaxation of $P_i, \quad L_{\mathrm{loc}} \leftarrow c(x^{\mathrm{LP}})$ or ∞

if $L_{loc} \geq U$ **then goto** line 2

if $x^{LP} \in P$ **then** $x^{\mathrm{IP}} \leftarrow x^{\mathrm{LP}}, \quad U \leftarrow L_{\mathrm{loc}}, \quad$ **goto** line 2

Select $j \in I$: $x_j^{\mathrm{LP}} \notin \mathbb{Z}$. Split P_i into $P_{2i+1} := P_i \cup \{x_j \leq \lfloor x_j^{\mathrm{LP}} \rfloor\}$, and $P_{2i+2} := P_i \cup \{x_j \geq \lceil x_j^{\mathrm{LP}} \rceil\}, \quad \mathcal{L} \leftarrow \mathcal{L} \cup \{P_{2i+1}, P_{2i+2}\}, \quad$ **goto** line 2

LP-based Branch-and-Bound (Algorithm)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching

$\mathcal{L} \leftarrow \{P\}, \quad U \leftarrow \infty, \quad x^{\mathrm{IP}} \leftarrow \mathrm{NULL}$

if $\mathcal{L} = \emptyset$ **then return** x^{IP} *and* U

Select $P_i \in \mathcal{L}, \quad \mathcal{L} \leftarrow \mathcal{L} \setminus \{P_i\}$

Solve LP relaxation of P_i, $\quad L_{\mathrm{loc}} \leftarrow c(x^{\mathrm{LP}})$ or ∞

if $L_{loc} \geq U$ **then goto** line 2

if $x^{LP} \in P$ **then** $x^{\mathrm{IP}} \leftarrow x^{\mathrm{LP}}, \quad U \leftarrow L_{\mathrm{loc}}$, **goto** line 2

Select $j \in I$: $x_j^{\mathrm{LP}} \notin \mathbb{Z}$. Split P_i into $P_{2i+1} := P_i \cup \{x_j \leq \lfloor x_j^{\mathrm{LP}} \rfloor\}$, and $P_{2i+2} := P_i \cup \{x_j \geq \lceil x_j^{\mathrm{LP}} \rceil\}, \quad \mathcal{L} \leftarrow \mathcal{L} \cup \{P_{2i+1}, P_{2i+2}\}$, **goto** line 2

LP-based Branch-and-Bound (Algorithm)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching

$\mathcal{L} \leftarrow \{P\}, \quad U \leftarrow \infty, \quad x^{\mathrm{IP}} \leftarrow \text{NULL}$

if $\mathcal{L} = \emptyset$ **then return** x^{IP} *and* U

Select $P_i \in \mathcal{L}, \quad \mathcal{L} \leftarrow \mathcal{L} \backslash \{P_i\}$

Solve LP relaxation of $P_i, \quad L_{\mathrm{loc}} \leftarrow c(x^{\mathrm{LP}})$ or ∞

if $L_{loc} \geq U$ **then goto** line 2

if $x^{LP} \in P$ **then** $x^{\mathrm{IP}} \leftarrow x^{\mathrm{LP}}, \quad U \leftarrow L_{\mathrm{loc}}$, **goto** line 2

Select $j \in I$: $x_j^{\mathrm{LP}} \notin \mathbb{Z}$. Split P_i into $P_{2i+1} := P_i \cup \{x_j \leq \lfloor x_j^{\mathrm{LP}} \rfloor\}$, and $P_{2i+2} := P_i \cup \{x_j \geq \lceil x_j^{\mathrm{LP}} \rceil\}, \quad \mathcal{L} \leftarrow \mathcal{L} \cup \{P_{2i+1}, P_{2i+2}\}$, **goto** line 2

LP-based Branch-and-Bound (Algorithm)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching

$\mathcal{L} \leftarrow \{P\}, \quad U \leftarrow \infty, \quad x^{\text{IP}} \leftarrow \text{NULL}$

if $\mathcal{L} = \emptyset$ **then return** x^{IP} *and* U

Select $P_i \in \mathcal{L}, \quad \mathcal{L} \leftarrow \mathcal{L} \setminus \{P_i\}$

Solve LP relaxation of $P_i, \quad L_{\text{loc}} \leftarrow c(x^{\text{LP}})$ or ∞

if $L_{loc} \geq U$ **then goto** line 2

if $x^{LP} \in P$ **then** $x^{\text{IP}} \leftarrow x^{\text{LP}}, \quad U \leftarrow L_{\text{loc}},$ **goto** line 2

Select $j \in I$: $x_j^{\text{LP}} \notin \mathbb{Z}$. Split P_i into $P_{2i+1} := P_i \cup \{x_j \leq \lfloor x_j^{\text{LP}} \rfloor\}$, and $P_{2i+2} := P_i \cup \{x_j \geq \lceil x_j^{\text{LP}} \rceil\}, \quad \mathcal{L} \leftarrow \mathcal{L} \cup \{P_{2i+1}, P_{2i+2}\},$ **goto** line 2

LP-based Branch-and-Bound (Algorithm)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching

$\mathcal{L} \leftarrow \{P\}, \quad U \leftarrow \infty, \quad x^{\text{IP}} \leftarrow \text{NULL}$

if $\mathcal{L} = \emptyset$ **then return** x^{IP} *and* U

Select $P_i \in \mathcal{L}, \quad \mathcal{L} \leftarrow \mathcal{L} \setminus \{P_i\}$

Solve LP relaxation of $P_i, \quad L_{\text{loc}} \leftarrow c(x^{\text{LP}})$ or ∞

if $L_{loc} \geq U$ **then goto** line 2

if $x^{LP} \in P$ **then** $x^{\text{IP}} \leftarrow x^{\text{LP}}, \quad U \leftarrow L_{\text{loc}}, \quad$ **goto** line 2

Select $j \in I$: $x_j^{\text{LP}} \notin \mathbb{Z}$. Split P_i into $P_{2i+1} := P_i \cup \{x_j \leq \lfloor x_j^{\text{LP}} \rfloor\}$, and $P_{2i+2} := P_i \cup \{x_j \geq \lceil x_j^{\text{LP}} \rceil\}, \quad \mathcal{L} \leftarrow \mathcal{L} \cup \{P_{2i+1}, P_{2i+2}\}, \quad$ **goto** line 2

LP-based Branch-and-Bound (Algorithm)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching

$\mathcal{L} \leftarrow \{P\}, \quad U \leftarrow \infty, \quad x^{\mathrm{IP}} \leftarrow \mathrm{NULL}$

if $\mathcal{L} = \emptyset$ **then return** x^{IP} *and* U

Select $P_i \in \mathcal{L}, \quad \mathcal{L} \leftarrow \mathcal{L} \backslash \{P_i\}$

Solve LP relaxation of $P_i, \quad L_{\mathrm{loc}} \leftarrow c(x^{\mathrm{LP}})$ or ∞

if $L_{loc} \geq U$ **then goto** line 2

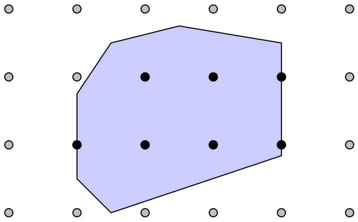
if $x^{LP} \in P$ **then** $x^{\mathrm{IP}} \leftarrow x^{\mathrm{LP}}, \quad U \leftarrow L_{\mathrm{loc}}, \quad$ **goto** line 2

Select $j \in I$: $x_j^{\mathrm{LP}} \notin \mathbb{Z}$. Split P_i into $P_{2i+1} := P_i \cup \{x_j \leq \lfloor x_j^{\mathrm{LP}} \rfloor\}$, and $P_{2i+2} := P_i \cup \{x_j \geq \lceil x_j^{\mathrm{LP}} \rceil\}, \quad \mathcal{L} \leftarrow \mathcal{L} \cup \{P_{2i+1}, P_{2i+2}\}, \quad$ **goto** line 2

LP-based Branch-and-Bound (Colorful Picture)

Steps

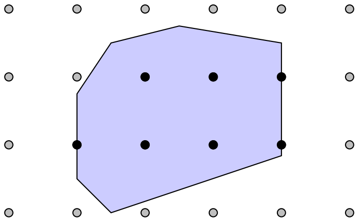
1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching



LP-based Branch-and-Bound (Colorful Picture)

Steps

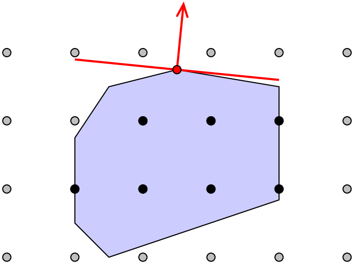
1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching



LP-based Branch-and-Bound (Colorful Picture)

Steps

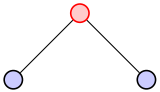
1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching

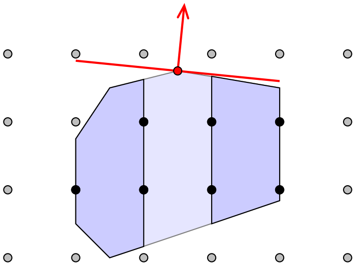


LP-based Branch-and-Bound (Colorful Picture)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching

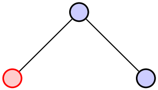


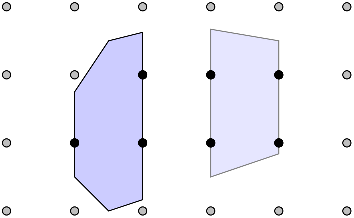


LP-based Branch-and-Bound (Colorful Picture)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching

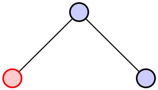


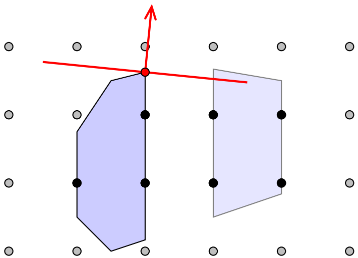


LP-based Branch-and-Bound (Colorful Picture)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching

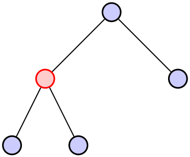


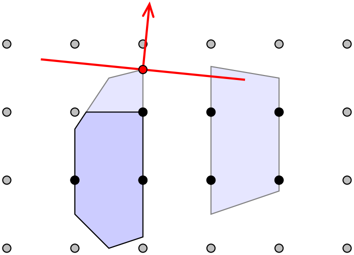


LP-based Branch-and-Bound (Colorful Picture)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching

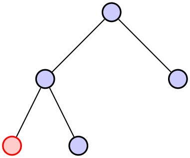


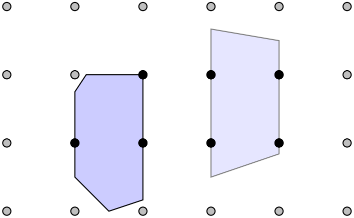


LP-based Branch-and-Bound (Colorful Picture)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching

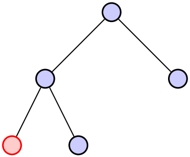


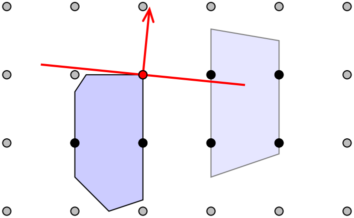


LP-based Branch-and-Bound (Colorful Picture)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching

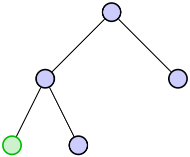


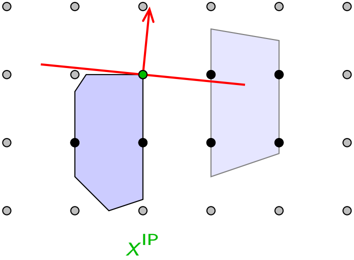


LP-based Branch-and-Bound (Colorful Picture)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching

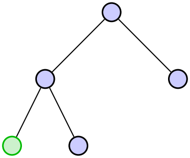


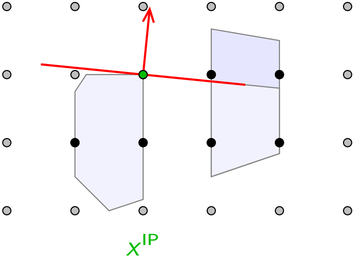


LP-based Branch-and-Bound (Colorful Picture)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching

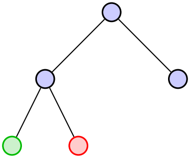


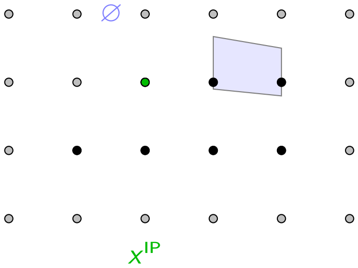


LP-based Branch-and-Bound (Colorful Picture)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching

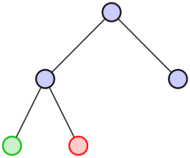


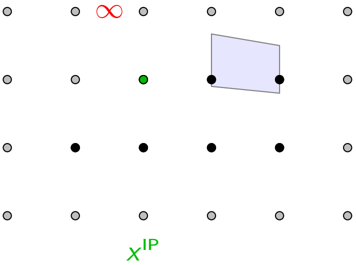


LP-based Branch-and-Bound (Colorful Picture)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching

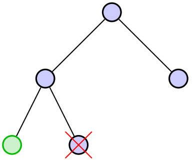


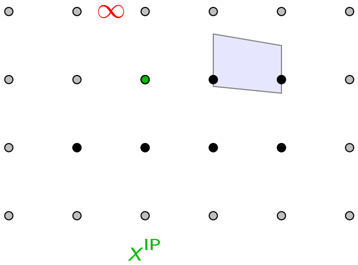


LP-based Branch-and-Bound (Colorful Picture)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching

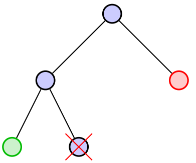


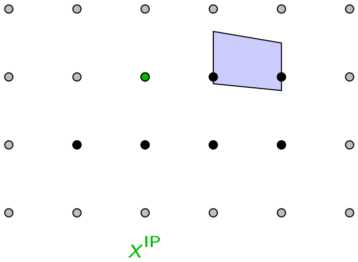


LP-based Branch-and-Bound (Colorful Picture)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching

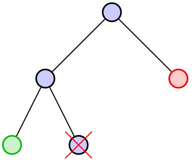


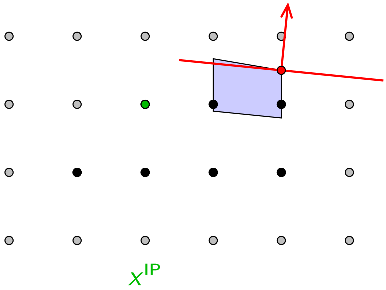


LP-based Branch-and-Bound (Colorful Picture)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching

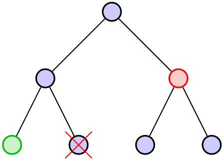


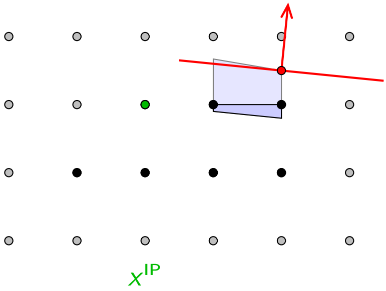


LP-based Branch-and-Bound (Colorful Picture)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching

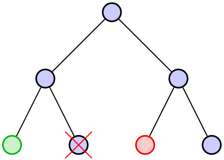


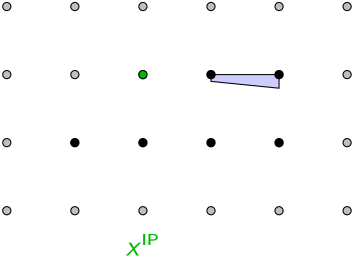


LP-based Branch-and-Bound (Colorful Picture)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching

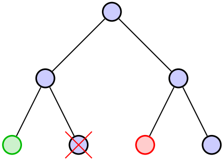


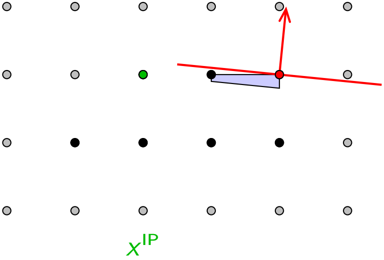


LP-based Branch-and-Bound (Colorful Picture)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching

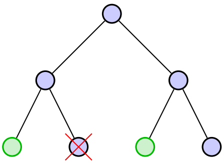


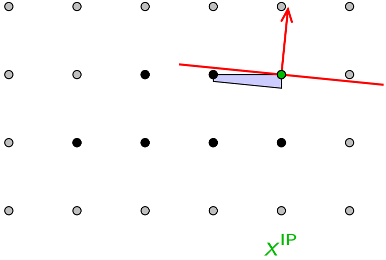


LP-based Branch-and-Bound (Colorful Picture)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching

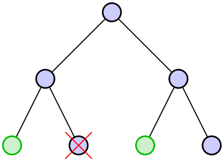


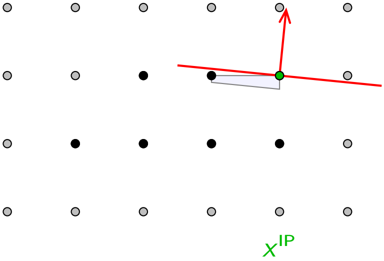


LP-based Branch-and-Bound (Colorful Picture)

Steps

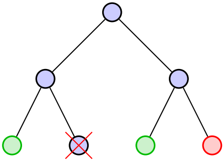
1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching

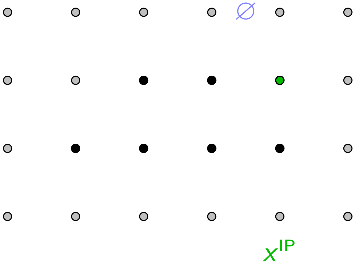




Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching

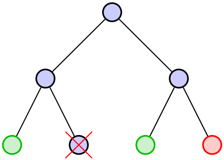


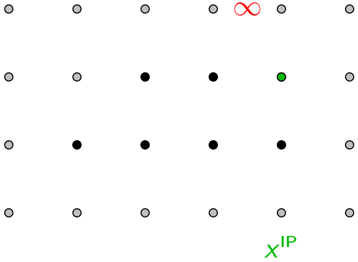


LP-based Branch-and-Bound (Colorful Picture)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching

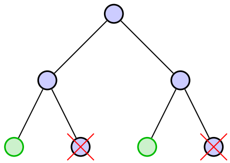


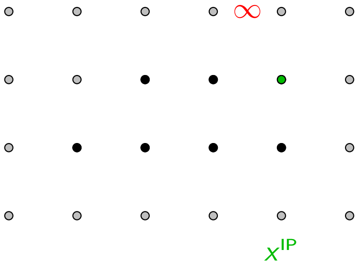


LP-based Branch-and-Bound (Colorful Picture)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching

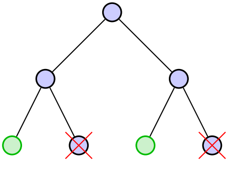


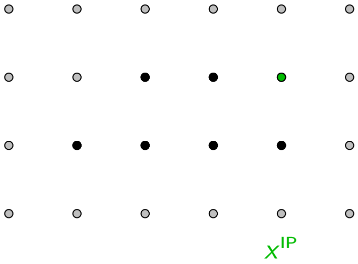


LP-based Branch-and-Bound (Colorful Picture)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching





```
SCIP> read check/IP/miplib2003/air04.mps.gz
SCIP> optimize
...
presolved problem has 7370 variables (7370 bin, 0 int, 0 impl, 0 cont)
    601 constraints of type <setppc>
  node   | left   |LP iter| mem |  dualbound     | primalbound    |  gap
      1 |      0 |  2756 |  40M| 5.553544e+04 |      --        |   Inf
...
      1 |      2 |  8342 |  41M| 5.564871e+04 |      --        |   Inf
     50 |     49 | 32939 |  41M| 5.576070e+04 |      --        |   Inf
*    73 |      8 | 47394 |  41M| 5.576165e+04 | 5.641900e+04 |  1.18%
  node   | left   |LP iter| mem |  dualbound     | primalbound    |  gap
*    76 |      9 | 47533 |  37M| 5.576165e+04 | 5.640000e+04 |  1.14%
*    79 |      8 | 47636 |  37M| 5.576165e+04 | 5.638500e+04 |  1.12%
*    82 |      7 | 48134 |  37M| 5.580334e+04 | 5.636300e+04 |  1.00%
*    98 |      4 | 50423 |  36M| 5.584525e+04 | 5.613800e+04 |  0.52%
    100 |      4 | 50529 |  35M| 5.584525e+04 | 5.613800e+04 |  0.52%
*   128 |      2 | 52075 |  35M| 5.584525e+04 | 5.613700e+04 |  0.52%
    150 |      2 | 54032 |  35M| 5.593390e+04 | 5.613700e+04 |  0.36%

SCIP Status        : problem is solved [optimal solution found]
Solving Time (sec) : 59.78
Solving Nodes      : 162
```

```
Welcome to CPLEX Interactive Optimizer 12.1.0
CPLEX is a registered trademark of IBM Corp.
CPLEX> read check/IP/Bugs/Kaibel/ggt3.lp
CPLEX> optimize
Presolve time =    0.00 sec.
MIP search method: dynamic search.
Parallel mode: none, using 1 thread.
Root relaxation solution time =    0.00 sec.

   Node     Left    Objective  IInf  Best Integer  Best Node      Gap

        0      0       1.0000     1                   1.0000
*       0+     0                          3.0000      1.0000   66.67%
        0      2       1.0000     1       3.0000      1.0000   66.67%
...
```

```
Welcome to CPLEX Interactive Optimizer 12.1.0
CPLEX is a registered trademark of IBM Corp.
CPLEX> read check/IP/Bugs/Kaibel/ggt3.lp
CPLEX> optimize
Presolve time =    0.00 sec.
MIP search method: dynamic search.
Parallel mode: none, using 1 thread.
Root relaxation solution time =    0.00 sec.

   Node     Left    Objective  IInf  Best Integer  Best Node       Gap

        0      0       1.0000     1                   1.0000
*       0+     0                          3.0000      1.0000    66.67%
        0      2       1.0000     1       3.0000      1.0000    66.67%
...
224000000      2       1.0000     1       3.0000      1.0000    66.67%

Solution pool: 1 solution saved.

MIP - Time limit exceeded, integer feasible:  Objective =  3.0000000000e+00
Current MIP best bound =  1.0000000000e+00 (gap = 2, 66.67%)
Solution time = 3600.01 sec.  Iterations = 224141556  Nodes = 224141558 (2)
```

```
bzfberth@opt29 ~/projects/scip> less check/IP/Bugs/Kaibel/ggt3.lp


Minimize
obj: x3
Subject To
c1:  12 x1 + 9  x2 - x3 = 0
Bounds
-inf <= x1
-inf <= x2
   1 <= x3
General
x1
x2
x3
End
```

```
bzfberth@opt29 ~/projects/scip> less check/IP/Bugs/Kaibel/ggt3.lp


Minimize
obj: x3
Subject To
c1:  12 x1 + 9  x2 - x3 = 0
Bounds
-inf <= x1
-inf <= x2
   1 <= x3
General
x1
x2
x3
End
```

Not a CPLEX issue!

Other codes do not perform better!

Branch-And-Bound (Algorithm) continued

$\mathcal{L} \leftarrow \{P\}, \quad U \leftarrow \infty, \quad x^{\mathrm{IP}} \leftarrow \mathrm{NULL}$

if $\mathcal{L} = \emptyset$ **then return** x^{IP} *and* U

Select $P_i \in \mathcal{L}, \quad \mathcal{L} \leftarrow \mathcal{L} \setminus \{P_i\}$

Solve LP relaxation of $P_i, \quad L_{\mathrm{loc}} \leftarrow c(x^{\mathrm{LP}})$ or ∞

if $L_{loc} \geq U$ **then goto** line 2

if $x^{LP} \in P$ **then** $x^{\mathrm{IP}} \leftarrow x^{\mathrm{LP}}, \quad U \leftarrow L_{\mathrm{loc}}$, **goto** line 2

Select $j \in I : x_j^{\mathrm{LP}} \notin \mathbb{Z}$. Split P_i into $P_{2i+1} := P_i \cup \{x_j \leq \lfloor x_j^{\mathrm{LP}} \rfloor\}$, and $P_{2i+2} := P_i \cup \{x_j \geq \lceil x_j^{\mathrm{LP}} \rceil\}, \quad \mathcal{L} \leftarrow \mathcal{L} \cup \{P_{2i+1}, P_{2i+2}\}$, **goto** line 2

Two degrees of freedom

1. Node selection
2. Variable selection (branching rule)

Branch-And-Bound (Algorithm) continued

$\mathcal{L} \leftarrow \{P\}, \quad U \leftarrow \infty, \quad x^{\mathrm{IP}} \leftarrow \mathrm{NULL}$

if $\mathcal{L} = \emptyset$ **then return** x^{IP} *and* U

Select $P_i \in \mathcal{L}, \quad \mathcal{L} \leftarrow \mathcal{L} \setminus \{P_i\}$

Solve LP relaxation of $P_i, \quad L_{\mathrm{loc}} \leftarrow c(x^{\mathrm{LP}})$ or ∞

if $L_{loc} \geq U$ **then goto** line 2

if $x^{LP} \in P$ **then** $x^{\mathrm{IP}} \leftarrow x^{\mathrm{LP}}, \quad U \leftarrow L_{\mathrm{loc}}, \quad$ **goto** line 2

Select $j \in I$: $x_j^{\mathrm{LP}} \notin \mathbb{Z}$. Split P_i into $P_{2i+1} := P_i \cup \{x_j \leq \lfloor x_j^{\mathrm{LP}} \rfloor\}$, and $P_{2i+2} := P_i \cup \{x_j \geq \lceil x_j^{\mathrm{LP}} \rceil\}, \quad \mathcal{L} \leftarrow \mathcal{L} \cup \{P_{2i+1}, P_{2i+2}\}, \quad$ **goto** line 2

Two degrees of freedom

1. Node selection
2. Variable selection (branching rule)

Same algorithm, different scope

Common goal: Keep branch-and-bound tree small!

MIP

- improve LP value
- reduce LP complexity

SAT

- detect infeasibilities
- generate short conflicts

CP

- enable propagations
- reduce domains

Same algorithm, different scope

Common goal: Keep branch-and-bound tree small!

MIP

- improve LP value
- reduce LP complexity

SAT

- detect infeasibilities
- generate short conflicts

CP

- enable propagations
- reduce domains

But

Also in MIP, sometimes...

- there's no objective
- instances are infeasible
- combinatorial structure

Same algorithm, different scope

Common goal: Keep branch-and-bound tree small!

MIP

- improve LP value
- reduce LP complexity

SAT

- detect infeasibilities
- generate short conflicts

CP

- enable propagations
- reduce domains

But

Also in MIP, sometimes...

- there's no objective
- instances are infeasible
- combinatorial structure

Standard MIP branching inferior in these cases

Branching rules in MIP

Most infeasible branching

- ▷ choose variable with fractional value closest to 0.5
- ▷ often referred to as a simple, standard rule
- ▷ computationally as bad as random branching!

Strong branching [ABCC '95]

- ▷ solve LP relaxations for some candidates, choose best
- ▷ effective w.r.t. number of nodes, expensive w.r.t. time

Pseudocost branching [Bénichou et al. '71]

- ▷ try to estimate LP values, based on history information
- ▷ effective, cheap, but weak in the beginning

Branching rules in MIP

Most infeasible branching

- ▷ choose variable with fractional value closest to 0.5
- ▷ often referred to as a simple, standard rule
- ▷ computationally as bad as random branching!

Strong branching [ABCC '95]

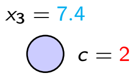
- ▷ solve LP relaxations for some candidates, choose best
- ▷ effective w.r.t. number of nodes, expensive w.r.t. time

Pseudocost branching [Bénichou et al. '71]

- ▷ try to estimate LP values, based on history information
- ▷ effective, cheap, but weak in the beginning

What are pseudocosts?

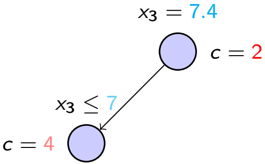
Estimating the objective



What are pseudocosts?

Estimating the objective

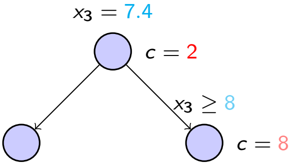
- objective gain per unit:
  - $\zeta^-(x_3) = \frac{4-2}{7.4-7} = \frac{2}{0.4} = 5$



What are pseudocosts?

Estimating the objective

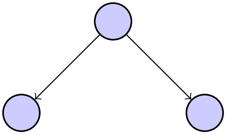
- objective gain per unit:
  - $\zeta^{+}(x_3) = \frac{8-2}{8-7.4} = \frac{6}{0.6} = 10$



What are pseudocosts?

Estimating the objective

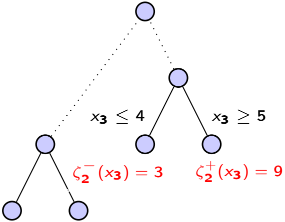
- objective gain per unit:
  - $\zeta_1^-(x_3) = 5$, $\zeta_1^+(x_3) = 10$



What are pseudocosts?

Estimating the objective

- objective gain per unit:
  - $\zeta_1^-(x_3) = 5$, $\zeta_1^+(x_3) = 10$
  - other values at other nodes



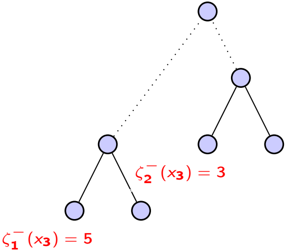
What are pseudocosts?

Estimating the objective

- objective gain per unit:
  - $\zeta_1^-(x_3) = 5$, $\zeta_1^+(x_3) = 10$
  - other values at other nodes
- pseudocosts:

  average objective gain

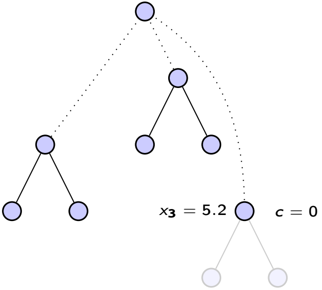
  $\psi^-(x_3) = \frac{\zeta_1^-(x_3)+\ldots+\zeta_n^-(x_3)}{n} = \frac{5+3}{2} = 4$



What are pseudocosts?

Estimating the objective

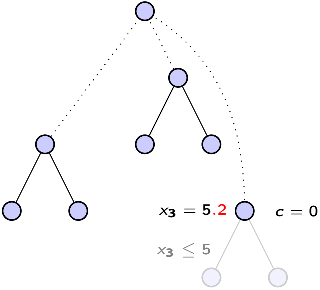
- objective gain per unit:
  - $\zeta_1^-(x_3) = 5$, $\zeta_1^+(x_3) = 10$
  - other values at other nodes
- pseudocosts:
  average objective gain
  $\psi^-(x_3) = 4$, $\psi^+(x_3) = 9.5$
- estimate increase of objective
  by pseudocosts and fractionality:



What are pseudocosts?

Estimating the objective

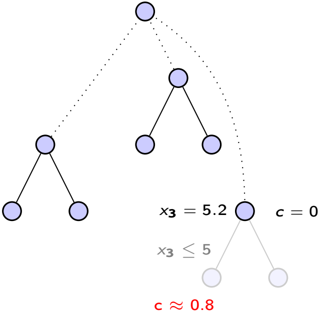
- objective gain per unit:
  - $\zeta_1^-(x_3) = 5$, $\zeta_1^+(x_3) = 10$
  - other values at other nodes
- pseudocosts:
  average objective gain
  $\psi^-(x_3) = 4$, $\psi^+(x_3) = 9.5$
- estimate increase of objective
  by pseudocosts and fractionality:
  $\psi^-(x_3) \cdot \mathrm{frac}(x_3)$



What are pseudocosts?

Estimating the objective

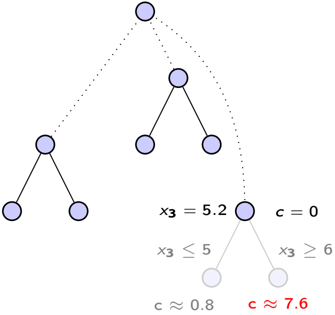
- objective gain per unit:
  - $\zeta_1^-(x_3) = 5$, $\zeta_1^+(x_3) = 10$
  - other values at other nodes
- pseudocosts:
  average objective gain
  $\psi^-(x_3) = 4$, $\psi^+(x_3) = 9.5$
- estimate increase of objective
  by pseudocosts and fractionality:
  $\psi^-(x_3) \cdot \mathrm{frac}(x_3) = 4 \cdot 0.2 = 0.8$,



What are pseudocosts?

Estimating the objective

- ▷ objective gain per unit:
  - $\zeta_1^-(x_3) = 5$, $\zeta_1^+(x_3) = 10$
  - other values at other nodes
- ▷ pseudocosts:
  average objective gain
  $\psi^-(x_3) = 4$, $\psi^+(x_3) = 9.5$
- ▷ estimate increase of objective
  by pseudocosts and fractionality:
  $\psi^-(x_3) \cdot \mathrm{frac}(x_3) = 4 \cdot 0.2 = 0.8$,
  and $\psi^+(x_3)(1 - \mathrm{frac}(x_3)) = 7.6$



Reliability Branching

Early branchings are the most important ones!
Problem: In the beginning, pseudocosts are all zero

Pseudocost with strong branching initialization [Linderoth and Savelsbergh '99]

- ▷ use strong branching, if pseudocosts have not been initialized yet

Reliability branching [Achterberg et. al. '05]

- ▷ use strong branching, if pseudocosts are unreliable
- ▷ unreliable: pseudocosts have been updated less than k times
- ▷ computational results: $k = 8$

A general branching rule for CP

Inference branching: [Li and Anbulagan '97]

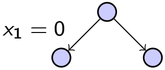
- average number of applied domain deductions
- history based
- captures combinatorial structure
- estimates tightening of subproblems

A general branching rule for CP

Inference branching: [Li and Anbulagan '97]

- average number of applied domain deductions
- history based
- captures combinatorial structure
- estimates tightening of subproblems

$$
\begin{aligned}
x_1 \quad\quad + x_2 &= 1 \\
x_1 + x_3 + x_4 &\leq 1 \\
-x_1 \quad\quad + z &\geq 3 \\
z &\in \mathbb{Z}_+ \\
x_i &\in \{0, 1\}
\end{aligned}
$$



$$
\begin{aligned}
x_1 = 0 \Rightarrow x_2 &= 1 \\
\Rightarrow z &\geq 3
\end{aligned}
$$

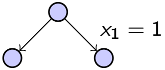
$$s_j^{\text{infer}}(-) = 2$$

A general branching rule for CP

Inference branching: [Li and Anbulagan '97]

- average number of applied domain deductions
- history based
- captures combinatorial structure
- estimates tightening of subproblems

$$\begin{aligned} x_1 \quad\quad\;\; + x_2 &= 1 \\ x_1 + x_3 + x_4 &\leq 1 \\ -x_1 \quad\quad\;\; + z &\geq 3 \\ z &\in \mathbb{Z}_+ \\ x_i &\in \{0,1\} \end{aligned}$$



$$\begin{aligned} x_1 = 1 &\Rightarrow x_2 = 0 \\ &\Rightarrow x_3 = 0 \\ &\Rightarrow x_4 = 0 \end{aligned}$$

$$s_j^{\text{infer}}(+) = 3$$

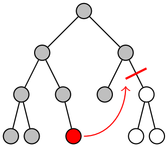
A general branching rule for CP

Inference branching: [Li and Anbulagan '97]

- average number of applied domain deductions
- history based
- captures combinatorial structure
- estimates tightening of subproblems

- analogy to pseudocost values in MIP
- one value for upwards branch, one for downwards
- initialization: probing ($\approx$ strong branching)

A general branching rule for SAT



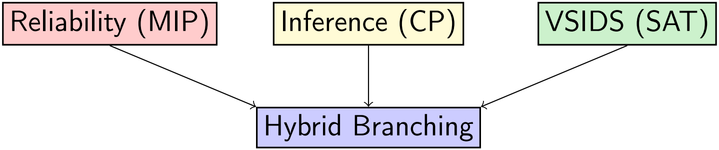
- important feature: conflict analysis / no-goods
- learning of small clauses which trigger infeasibility
- can be generalized to MIP, CP

VSIDS branching: [Moskewicz et. al. '01]

- largest number of (conflict) clauses, a variable appears in
- prefer "recent" conflicts
- "recent": exponentially decreasing importance
- works in particular well for infeasible problems
- state-of-the-art in SAT solving

Put it all together

Hybrid Branching: [Achterberg and B. '09]
Combine strategies to a new hybrid strategy for MIP



- use reliable pseudocosts, inference values, VSIDS
- additionally incorporate:
  - number of pruned subproblems
  - average length of conflict clauses

Hybrid branching

How the combination works:

- scaling: divide each value by average over all variables
- normalize each of the (scaled) values by $f : \mathbb{Q}_{\geq 0} \rightarrow [0, 1),\ x \mapsto \frac{x}{x+1}$
- use a weighted sum of all criteria

Hybrid branching

How the combination works:

- scaling: divide each value by average over all variables
- normalize each of the (scaled) values by $f: \mathbb{Q}_{\geq 0} \to [0,1),\ x \mapsto \frac{x}{x+1}$
- use a weighted sum of all criteria

In formulae:

$$s_j = \omega^{\text{pscost}} f\left(\frac{s_j^{\text{pscost}}}{s_\varnothing^{\text{pscost}}}\right) + \omega^{\text{infer}} f\left(\frac{s_j^{\text{infer}}}{s_\varnothing^{\text{infer}}}\right) + \omega^{\text{vsids}} f\left(\frac{s_j^{\text{vsids}}}{s_\varnothing^{\text{vsids}}}\right) + \omega^{\text{prune}} f\left(\frac{s_j^{\text{prune}}}{s_\varnothing^{\text{prune}}}\right) + \omega^{\text{conf}} f\left(\frac{s_j^{\text{conf}}}{s_\varnothing^{\text{conf}}}\right)$$

Hybrid branching

How the combination works:

- scaling: divide each value by average over all variables
- normalize each of the (scaled) values by $f : \mathbb{Q}_{\geq 0} \to [0, 1),\ x \mapsto \frac{x}{x+1}$
- use a weighted sum of all criteria

In formulae:

$$s_j = \omega^{\text{pscost}} f\left(\frac{s_j^{\text{pscost}}}{s_\varnothing^{\text{pscost}}}\right) + \omega^{\text{infer}} f\left(\frac{s_j^{\text{infer}}}{s_\varnothing^{\text{infer}}}\right) + \omega^{\text{vsids}} f\left(\frac{s_j^{\text{vsids}}}{s_\varnothing^{\text{vsids}}}\right) + \omega^{\text{prune}} f\left(\frac{s_j^{\text{prune}}}{s_\varnothing^{\text{prune}}}\right) + \omega^{\text{conf}} f\left(\frac{s_j^{\text{conf}}}{s_\varnothing^{\text{conf}}}\right)$$

Choice for the weights:

- high weight for pseudocosts: 1
- medium weight for VSIDS and conflict length: 10^{-2} and 10^{-3}, resp.
- low weight for inference and cutoff values: 10^{-4}

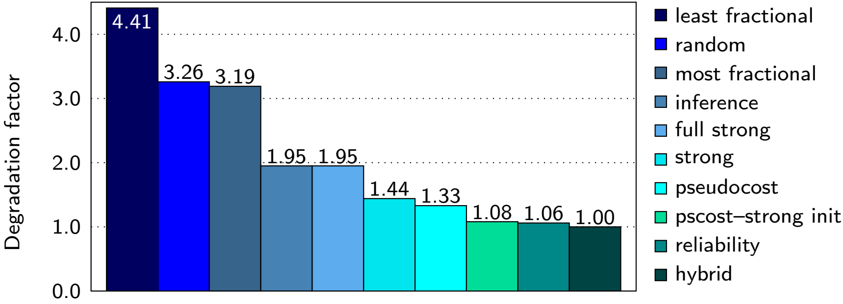
Hybrid branching

Branching score function

- yields two values: One for downwards, one for upwards branching
- need to combine them to a single value
- usually: convex sum
- includes minimum and maximum as extreme cases

- we use: multiplication
  $\text{score}(x_j) = \max\{s_j^-, \epsilon\} \cdot \max\{s_j^+, \epsilon\}$ $(\epsilon = 10^{-6})$
- computational results: 10% faster

Computational results



- ratio to hybrid branching (SCIP default)
- large test set: 575 instances
- shifted geometric time
- time limit of 1 hour

Computational results II

Test set:

- MIPLIB2003: selection of 60 quite different, difficult instances
- Cor@l: Huge collection of 350 instances
- Cor@l-BP: The 118 pure 0/1-programs of the Cor@l test set
- Infeasible: 30 infeasible graph coloring instances

Comparison:

- geometric means of overall running time and number of branch-and-bound nodes
- ratio between reliability branching and hybrid branching

Computational results III

test set	MIPLIB2003		Cor@l	
	Time	Nodes	Time	Nodes
reliability	450.4	5091	803.6	4110
hybrid	445.6	5051	735.0	3575
ratio reli/hyb	1.01	1.01	1.09	1.15

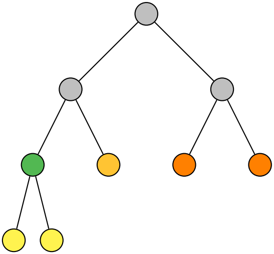
Result: No difference / slight improvement for general MIPs

test set	Cor@l-BP		Infeasible	
	Time	Nodes	Time	Nodes
reliability	672.4	2145	290.7	5612
hybrid	577.2	1681	166.0	1998
ratio reli/hyb	1.16	1.28	1.75	2.81

Result: Medium / large improvement for special MIPs

Outline

Node Selection



Task

- ▷ improve primal bound
- ▷ keep comp. effort small
- ▷ improve global dual bound

Techniques

- ▷ basic rules:
  - depth first search (DFS):
    $\longrightarrow$ early feasible solutions
  - best bound search (BBS):
    $\longrightarrow$ improve dual bound
  - best estimate search (BES):
    $\longrightarrow$ improve primal bound
- ▷ combinations:
  - BBS or BES with plunging
  - hybrid BES/BBS
  - interleaved BES/BBS

Remarks on Node Selection Rules

Computational Effort

Distinction of open leaf nodes:

- child node $\longrightarrow$ subproblem processing is less expensive
- sibling node $\longrightarrow$ subproblem processing is less expensive
- other open leaf node

Results

- best estimate faster than best bound
- interleaved BES/BBS faster than hybrid
- speed-up by plunging (interleaved DFS)

Outline

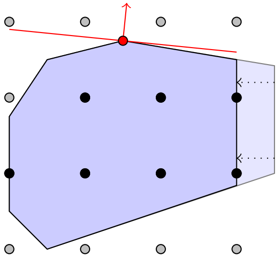
Presolving

It's a dirty job but somebody has to do it!

Presolving

Task

- ▷ reduce size of model by removing irrelevant information
- ▷ strengthen LP relaxation by exploiting integrality information
- ▷ make the LP relaxation numerically more stable
- ▷ extract useful information



Primal Reductions:
- ▷ based on feasibility reasoning
- ▷ no feasible solution is cut off

Dual Reductions:
- ▷ consider objective function
- ▷ at least one optimal solution remains

Trivial stuff

- remove empty rows, columns
  - e.g., $\mathbf{0}^T x \leq b_i,\ b_i < 0 \Rightarrow$ infeasible
- tighten fractional bounds of integer variables
- substitute fixed variables
- boundshifting of general integers
  - e.g., Replace $x_i \in \{N, N+1\},\ N \in \mathbb{Z}$ by $\tilde{x} \in \{0,1\},\ \tilde{x} = x - N$
- replace singleton rows
  - e.g., $a_{ij}x_j \leq b_i,\ a_{ij} < 0 \Rightarrow x_j \geq \frac{b_i}{a_{ij}} \Rightarrow$ new lower bound on x_j
- normalize constraints
  - e.g., if all coefficients are integral, divide by greatest common divisor
- upgrade constraints
- ...

Linear presolving

Minimal and maximal activities

Let a linear constraint $a^T x \leq b$, and bounds $\ell \leq x \leq u$ be given.

$$\alpha_{\min} \quad := \quad \begin{array}{cc} \min & a^T x \\ s.t. & \ell \leq x \leq u \end{array} \quad = \sum_{j, a_j > 0} a_j \ell_j + \sum_{j, a_j < 0} a_j u_j .$$

the minimal activity of the linear constraint.

Linear presolving

Minimal and maximal activities

Let a linear constraint $a^T x \leq b$, and bounds $\ell \leq x \leq u$ be given.

$$\alpha_{\max} := \begin{array}{ll} \max & a^T x \\ s.t. & \ell \leq x \leq u \end{array} = \sum_{j, a_j > 0} a_j u_j + \sum_{j, a_j < 0} a_j \ell_j.$$

the maximal activity of the linear constraint.

Linear presolving

Minimal and maximal activities

Let a linear constraint $a^T x \leq b$, and bounds $\ell \leq x \leq u$ be given.

$$\alpha_{\max} := \begin{array}{cc} \max & a^T x \\ s.t. & \ell \leq x \leq u \end{array} = \sum_{j, a_j > 0} a_j u_j + \sum_{j, a_j < 0} a_j \ell_j.$$

the maximal activity of the linear constraint.

First observation

- $\alpha_{\min} > b \Rightarrow$ problem infeasible
- $\alpha_{\max} \leq b \Rightarrow$ constraint redundant

Linear presolving

Minimal and maximal activities

Let a linear constraint $a^T x \leq b$, and bounds $\ell \leq x \leq u$ be given.

$$\alpha_{\min} \quad := \quad \begin{array}{cc} \min & a^T x \\ s.t. & \ell \leq x \leq u \end{array} \quad = \sum_{j, a_j > 0} a_j \ell_j + \sum_{j, a_j < 0} a_j u_j.$$

the minimal activity of the linear constraint.

Bound strengthening

Let $a_k > 0$. For all feasible solutions x, it holds that:

$$a^T x - a_k x_k + a_k x_k \leq b$$

Linear presolving

Minimal and maximal activities

Let a linear constraint $a^T x \le b$, and bounds $\ell \le x \le u$ be given.

$$\alpha_{\min} := \begin{array}{cc} \min & a^T x \\ s.t. & \ell \le x \le u \end{array} = \sum_{j, a_j > 0} a_j \ell_j + \sum_{j, a_j < 0} a_j u_j.$$

the minimal activity of the linear constraint.

Bound strengthening

Let $a_k > 0$. For all feasible solutions x, it holds that:

$$a^T x - a_k x_k + a_k x_k \le b \;\Leftrightarrow\; x_k \le \frac{b - (a^T x - a_k x_k)}{a_k}$$

Linear presolving

Minimal and maximal activities

Let a linear constraint $a^T x \leq b$, and bounds $\ell \leq x \leq u$ be given.

$$\alpha_{\min} \quad := \quad \begin{array}{cc} \min & a^T x \\ s.t. & \ell \leq x \leq u \end{array} \quad = \sum_{j, a_j > 0} a_j \ell_j + \sum_{j, a_j < 0} a_j u_j .$$

the minimal activity of the linear constraint.

Bound strengthening

Let $a_k > 0$. For all feasible solutions x, it holds that:

$$\begin{aligned} a^T x - a_k x_k + a_k x_k \leq b \;\Leftrightarrow\; x_k &\leq \frac{b - (a^T x - a_k x_k)}{a_k} \\ \Rightarrow\; x_k &\leq \frac{b - \alpha_{\min} + a_k \ell_k}{a_k}. \end{aligned}$$

Linear presolving

Minimal and maximal activities

Let a linear constraint $a^T x \leq b$, and bounds $\ell \leq x \leq u$ be given.

$$\alpha_{\max} \quad := \quad \begin{array}{ll} \max & a^T x \\ s.t. & \ell \leq x \leq u \end{array} \quad = \sum_{j, a_j > 0} a_j u_j + \sum_{j, a_j < 0} a_j \ell_j.$$

the maximal activity of the linear constraint.

Coefficient tightening

Let $a_k > 0$, $x_k \in \{0, 1\}$, $\alpha_{\max} - a_k < b$. Then

$$a_k x_k + \sum_{j \neq k} a_j x_j \leq b$$

can be reformulated as

$$(\alpha_{\max} - b) x_k + \sum_{j \neq k} a_j x_j \leq \alpha_{\max} - a_k.$$

Linear presolving

Minimal and maximal activities

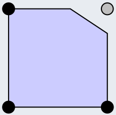
Let a linear constraint $a^T x \leq b$, and bounds $\ell \leq x \leq u$ be given.

$$\alpha_{\max} := \begin{array}{ll} \max & a^T x \\ s.t. & \ell \leq x \leq u \end{array} = \sum_{j, a_j > 0} a_j u_j + \sum_{j, a_j < 0} a_j \ell_j .$$

the maximal activity of the linear constraint.

Coefficient tightening

Consider $7x_1 + 8x_2 \leq 13$. $\alpha_{\max} = 15$



Linear presolving

Minimal and maximal activities

Let a linear constraint $a^T x \leq b$, and bounds $\ell \leq x \leq u$ be given.

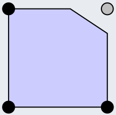
$$\alpha_{\max} := \begin{array}{cc} \max & a^T x \\ s.t. & \ell \leq x \leq u \end{array} = \sum_{j, a_j > 0} a_j u_j + \sum_{j, a_j < 0} a_j \ell_j.$$

the maximal activity of the linear constraint.

Coefficient tightening

Consider $7x_1 + 8x_2 \leq 13$. $\alpha_{\max} = 15$

$(15 - 13)x_1 + 8x_2 \leq (15 - 7)$



Linear presolving

Minimal and maximal activities

Let a linear constraint $a^T x \leq b$, and bounds $\ell \leq x \leq u$ be given.

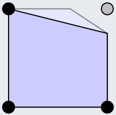
$$\alpha_{\max} := \begin{array}{cc} \max & a^T x \\ s.t. & \ell \leq x \leq u \end{array} = \sum_{j, a_j > 0} a_j u_j + \sum_{j, a_j < 0} a_j \ell_j .$$

the maximal activity of the linear constraint.

Coefficient tightening

Consider $7x_1 + 8x_2 \leq 13$. $\alpha_{\max} = 15$

$2x_1 + 8x_2 \leq 8 \quad \alpha_{\max} = 10$



Linear presolving

Minimal and maximal activities

Let a linear constraint $a^T x \leq b$, and bounds $\ell \leq x \leq u$ be given.

$$\alpha_{\max} \quad := \quad \begin{array}{cc} \max & a^T x \\ s.t. & \ell \leq x \leq u \end{array} \quad = \sum_{j, a_j > 0} a_j u_j + \sum_{j, a_j < 0} a_j \ell_j.$$

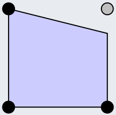
the maximal activity of the linear constraint.

Coefficient tightening

Consider $7x_1 + 8x_2 \leq 13$. $\alpha_{\max} = 15$

$$2x_1 + 8x_2 \leq 8 \quad \alpha_{\max} = 10$$

$$2x_1 + (10-8)x_2 \leq (10-8)$$



Linear presolving

Minimal and maximal activities

Let a linear constraint $a^T x \leq b$, and bounds $\ell \leq x \leq u$ be given.

$$\alpha_{\max} := \begin{array}{cc} \max & a^T x \\ s.t. & \ell \leq x \leq u \end{array} = \sum_{j, a_j > 0} a_j u_j + \sum_{j, a_j < 0} a_j \ell_j.$$

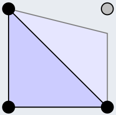
the maximal activity of the linear constraint.

Coefficient tightening

Consider $7x_1 + 8x_2 \leq 13$. $\alpha_{\max} = 15$

$2x_1 + 8x_2 \leq 8 \quad \alpha_{\max} = 10$

$2x_1 + 2x_2 \leq 2$



Linear presolving

Minimal and maximal activities

Let a linear constraint $a^T x \leq b$, and bounds $\ell \leq x \leq u$ be given.

$$\alpha_{\max} \;:=\; \begin{array}{ll} \max & a^T x \\ s.t. & \ell \leq x \leq u \end{array} \;=\; \sum_{j, a_j > 0} a_j u_j + \sum_{j, a_j < 0} a_j \ell_j .$$

the maximal activity of the linear constraint.

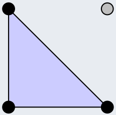
Coefficient tightening

Consider $7x_1 + 8x_2 \leq 13.$ $\alpha_{\max} = 15$

$$2x_1 + 8x_2 \leq 8 \quad \alpha_{\max} = 10$$

$$2x_1 + 2x_2 \leq 2$$

$$x_1 + x_2 \leq 1$$



Linear presolving

Minimal and maximal activities

Let a linear constraint $a^T x \leq b$, and bounds $\ell \leq x \leq u$ be given.

$$\alpha_{\max} := \begin{array}{cc} \max & a^T x \\ s.t. & \ell \leq x \leq u \end{array} = \sum_{j, a_j > 0} a_j u_j + \sum_{j, a_j < 0} a_j \ell_j.$$

the maximal activity of the linear constraint.

Coefficient tightening

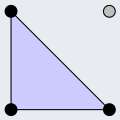
Consider $7x_1 + 8x_2 \leq 13$. $\alpha_{\max} = 15$

$2x_1 + 8x_2 \leq 8 \quad \alpha_{\max} = 10$

$2x_1 + 2x_2 \leq 2$

$x_1 + \ x_2 \leq 1$

$\text{setpack}(x_1, x_2)$



And so on...

- probing: tentatively fix binary variables and propagate
- dominance test: pairwise comparison of rows/columns
- aggregation of equations with only two variables
  $a_k x_k + a_j x_j = b \Rightarrow x_k = \frac{b}{ai} - \frac{a_j}{ak} x_j$
- dual fixing: If $a_{ik} \geq 0$ for all i and $c_k \geq 0$, then x_k can be fixed to its lower bound
- dual aggregation: If $c_k \geq 0$ and there is exactly one i for which $a_{ik} < 0$, we can aggregate $x_k = \frac{b_i}{a_j} - \frac{1}{a_k} \sum a_j x_j$.
- dual bound reduction: Strengthen bounds of variables to the tightest value for which all its constraints are redundant
- clique detection (e.g., for knapsack constraints)
- variable lifting
- ...

Outline

The Heart of the CIP Concept

Mixed Integer Program

- Linear objective function
- Linear constraints
- Real and integer variables

Constraint Program

- Arbitrary objective function
- Arbitrary constraints
- Arbitrary (discrete) variables

Constraint Integer Program

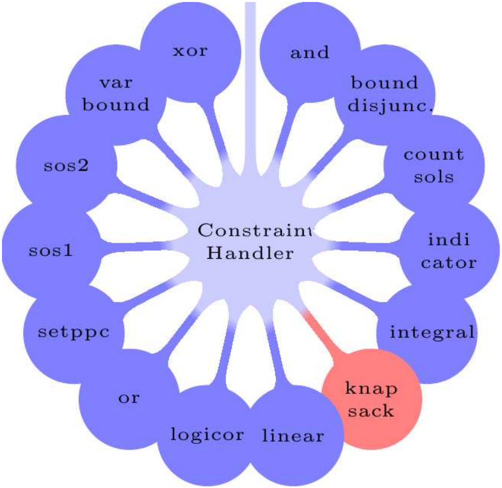
- Linear objective function
- Arbitrary constraints
- Real and integer variables
- After fixing all integer variables: CIP becomes an LP

Remark:

- Arbitrary objective or variables modeled by constraints

$\longrightarrow$ For each type of constraint, one constraint handler is responsible.

Default Plugins



Special Linear Constraints

$$\begin{aligned}
\min \quad & x_1 + x_2 + x_3 + x_4 + x_5 + y_1 + y_2 \\
\text{s.t.} \quad & y_1 \le 3 + 4\,x_1 \\
& y_2 \le 4 + 5\,x_2 \\
& 3\,x_1 + 2\,x_2 + 4\,x_3 \le 8 \\
& x_2 + x_3 + x_4 = 1 \\
& 3\,x_1 + 4.5\,x_2 + 3.2\,x_5 + 0.8\,y_1 + 1.2\,y_2 \ge 4 \\
& x_1,\, x_2,\, x_3,\, x_4 \in \{0,1\} \\
& x_5 \in \mathbb{Z}_+ \\
& y_1,\, y_2 \in \mathbb{R}_+
\end{aligned}$$

```
presolved problem has 7 variables (4 bin, 1 int, 0 impl, 2 cont) and 5
      2 constraints of type <varbound>
      1 constraints of type <knapsack>
      1 constraints of type <setppc>
      1 constraints of type <linear>
```

Special Linear Constraints

$$
\begin{aligned}
\min \quad & x_1 + x_2 + x_3 + x_4 + x_5 + y_1 + y_2 \\
\text{s.t.} \quad & y_1 \le 3 + 4\,x_1 \\
& y_2 \le 4 + 5\,x_2 \\
& 3\,x_1 + 2\,x_2 + 4\,x_3 \le 8 \\
& x_2 + x_3 + x_4 = 1 \\
& 3\,x_1 + 4.5\,x_2 + 3.2\,x_5 + 0.8\,y_1 + 1.2\,y_2 \ge 4 \\
& x_1,\ x_2,\ x_3,\ x_4 \in \{0,1\} \\
& x_5 \in \mathbb{Z}_+ \\
& y_1,\ y_2 \in \mathbb{R}_+
\end{aligned}
$$

```
presolved problem has 7 variables (4 bin, 1 int, 0 impl, 2 cont) and 5
      2 constraints of type <varbound>
      1 constraints of type <knapsack>
      1 constraints of type <setppc>
      1 constraints of type <linear>
```

Special Linear Constraints

$$
\begin{aligned}
\min \quad & x_1 + x_2 + x_3 + x_4 + x_5 + y_1 + y_2 \\
\text{s.t.} \quad & y_1 \le 3 + 4\,x_1 \\
& y_2 \le 4 + 5\,x_2 \\
& 3\,x_1 + 2\,x_2 + 4\,x_3 \le 8 \\
& x_2 + x_3 + x_4 = 1 \\
& 3\,x_1 + 4.5\,x_2 + 3.2\,x_5 + 0.8\,y_1 + 1.2\,y_2 \ge 4 \\
& x_1,\, x_2,\, x_3,\, x_4 \in \{0,1\} \\
& x_5 \in \mathbb{Z}_+ \\
& y_1,\, y_2 \in \mathbb{R}_+
\end{aligned}
$$

```
presolved problem has 7 variables (4 bin, 1 int, 0 impl, 2 cont) and 5
      2 constraints of type <varbound>
      1 constraints of type <knapsack>
      1 constraints of type <setppc>
      1 constraints of type <linear>
```

Special Linear Constraints

$$
\begin{aligned}
\min \quad & x_1 + x_2 + x_3 + x_4 + x_5 + y_1 + y_2 \\
\text{s.t.} \quad & y_1 \le 3 + 4\,x_1 \\
& y_2 \le 4 + 5\,x_2 \\
& 3\,x_1 + 2\,x_2 + 4\,x_3 \le 8 \\
& {\color{red} x_2 + x_3 + x_4 = 1} \\
& 3\,x_1 + 4.5\,x_2 + 3.2\,x_5 + 0.8\,y_1 + 1.2\,y_2 \ge 4 \\
& {\color{red} x_1,\, x_2,\, x_3,\, x_4 \in \{0,1\}} \\
& x_5 \in \mathbb{Z}_+ \\
& y_1,\, y_2 \in \mathbb{R}_+
\end{aligned}
$$

```
presolved problem has 7 variables (4 bin, 1 int, 0 impl, 2 cont) and 5
      2 constraints of type <varbound>
      1 constraints of type <knapsack>
      1 constraints of type <setppc>
      1 constraints of type <linear>
```

Special Linear Constraints

$$
\begin{aligned}
\min \quad & x_1 + x_2 + x_3 + x_4 + x_5 + y_1 + y_2 \\
\text{s.t.} \quad & y_1 \le 3 + 4\,x_1 \\
& y_2 \le 4 + 5\,x_2 \\
& 3\,x_1 + 2\,x_2 + 4\,x_3 \le 8 \\
& x_2 + x_3 + x_4 = 1 \\
& 3\,x_1 + 4.5\,x_2 + 3.2\,x_5 + 0.8\,y_1 + 1.2\,y_2 \ge 4 \\
& x_1, x_2, x_3, x_4 \in \{0,1\} \\
& x_5 \in \mathbb{Z}_+ \\
& y_1, y_2 \in \mathbb{R}_+
\end{aligned}
$$

```
presolved problem has 7 variables (4 bin, 1 int, 0 impl, 2 cont) and 5
       2 constraints of type <varbound>
       1 constraints of type <knapsack>
       1 constraints of type <setppc>
       1 constraints of type <linear>
```

Knapsack Constraints

Feasible region of 0-1 knapsack problem:

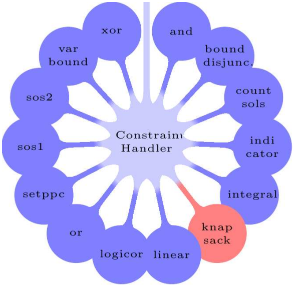
$$\{\ x \in \{0,1\}^{|N|}\ :\ \sum_{j \in N} a_j x_j \leq b\ \}$$

- weights of the variables: $a_j \in \mathbb{Z}_+$ for all $j \in N$
- capacity of the knapsack: $b \in \mathbb{Z}_+$

Ingredients of a Constraint Handler

Callback Methods

- ▷ **Fundamental:**
  - CONSLOCK
  - CONSCHECK
  - CONSENFOLP, CONSENFOPS
- ▷ **Additional:**
  - CONSINIT..., CONSEXIT...
  - CONSSEPALP, CONSSEPASOL
  - CONSPROP, CONSRESPROP, CONSPRESOL
  - CONSACTIVE, CONSDEACTIVE, CONSENABLE, CONSDISABLE, CONSTRANS, CONSDELETE, CONSFREE
  - CONSPRINT, CONSCOPY, CONSPARSE



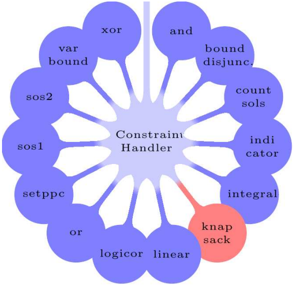
Further Ingredients

- ▷ Private data
- ▷ Interface methods
- ▷ Properties/Parameters

Ingredients of a Constraint Handler

Callback Methods

- Fundamental:
  - CONSLOCK
  - CONSCHECK
  - CONSENFOLP, CONSENFOPS
- Additional:
  - CONSINIT..., CONSEXIT...
  - CONSSEPALP, CONSSEPASOL
  - CONSPROP, CONSRESPROP, CONSPRESOL
  - CONSACTIVE, CONSDEACTIVE, CONSENABLE, CONSDISABLE, CONSTRANS, CONSDELETE, CONSFREE
  - CONSPRINT, CONSCOPY, CONSPARSE



Further Ingredients

- Private data
- Interface methods
- Properties/Parameters

Private Data

Constraint data: information needed to define single constraint

```
struct SCIP_ConsData
{
  SCIP_VAR**      vars;      // variables in knapsack
  int             nvars;     // number of variables
  SCIP_Longint*   weights;   // weights of variables
  SCIP_Longint    capacity;  // capacity of knapsack
};
```

Constraint handler data: information belonging to constraint handler itself

```
struct SCIP_ConshdlrData
{
  int   maxrounds;    // max nr. of sepa rounds per node
  int   maxsepacuts;  // max nr. of cuts per sepa round
};
```

Interface Methods

Creating a single knapsack constraint.

```
SCIP_RETCODE SCIPcreateConsKnapsack(
  SCIP*          scip,       // SCIP data structure
  SCIP_CONS**    cons,       // pointer to hold created cons
  const char*    name,       // name of constraint
  SCIP_VAR**     vars,       // array with variables
  int            nvars,      // number of variables in knapsack
  SCIP_Longint*  weights,    // array with weights
  SCIP_Longint   capacity,   // capacity of knapsack
  SCIP_Bool      separate,   // should constraint be separated?
  ...
)
{
  SCIP_CONSDATA* consdata;
  SCIP_CALL( consdataCreate(scip, &consdata, nvars, vars,
                            weights, capacity) );
  SCIP_CALL( SCIPcreateCons(scip, cons, name, conshdlr,
                            consdata, separate, ...) );
  ...
}
```

Interface Methods

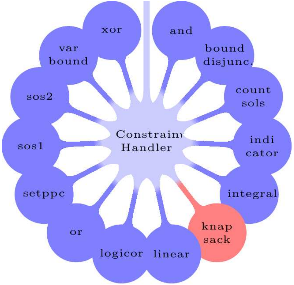
Including the knapsack constraint handler.

```
SCIP_RETCODE SCIPincludeConshdlrKnapsack(
  SCIP*          scip         // SCIP data structure
)
{
  SCIP_CONSHDLRDATA* conshdlrdata;

  SCIP_CALL( conshdlrdataCreate(scip, &conshdlrdata) );

  SCIP_CALL( SCIPincludeConshdlr(scip, CONSHDLR_CHECKPRIORITY,
             consCheckKnapsack, ..., conshdlrdata) );
  ...
}
```

Ingredients of a Constraint Handler

Callback Methods

- ▷ Fundamental:
  - CONSLOCK
  - CONSCHECK
  - CONSENFOLP, CONSENFOPS
- ▷ Additional:
  - CONSINIT..., CONSEXIT...
  - CONSSEPALP, CONSSEPASOL
  - CONSPROP, CONSRESPROP, CONSPRESOL
  - CONSACTIVE, CONSDEACTIVE, CONSENABLE, CONSDISABLE, CONSTRANS, CONSDELETE, CONSFREE
  - CONSPRINT, CONSCOPY, CONSPARSE



Further Ingredients

- ▷ Private data
- ▷ Interface methods
- ▷ Properties/Parameters

CONSLOCK

- Provides dual information for single constraints (useful for presolving, primal heuristics, ...)
- For each variable of a constraint, returns whether ...
  - increasing its value, and/or
  - decreasing its value

  may lead to a violation of the constraint

$$3\,x_1 - 5\,x_2 + 2\,x_3 \qquad \leq 7$$

increasing: x_1 and x_3

decreasing: x_2

CONSCHECK

Most important callback ...

- usually called by primal heuristics
- checks given solution for feasibility wrt all constraints of its type
- possible result values
  - SCIP_FEASIBLE
  - SCIP_INFEASIBLE

Given solution:

$$(x_1,\ x_2,\ x_3) = (1,\ 0,\ 1)$$

Knapsack constraints:

$$3x_1 + 6x_2 + 4x_3 \leq 8$$

$$2x_1 + \qquad + 2x_3 \leq 3$$

Result: SCIP_INFEASIBLE

CONSENFOLP and CONSENFOPS

CONSENFOLP: checks LP solution for feasibility
CONSENFOPS: checks Pseudo solution for feasibility

LP solution

- solution of LP relaxation

$$\begin{array}{ll} \min & x_1 - x_2 + x_3 \\ \text{s.t.} & 3\,x_1 + 8\,x_2 + 4\,x_3 \le 4 \\ & x_1,\, x_2,\, x_3 \in \{0,1\} \end{array}$$

LP solution: $(0, \frac{1}{2}, 0)$

Pseudo Solution

- solution of LP relaxation with only bound constraints
- used if LP solving disabled, or
- numerical difficulties occurred

$$\begin{array}{ll} \min & x_1 - x_2 + x_3 \\ \text{s.t.} & x_1,\, x_2,\, x_3 \in \{0,1\} \end{array}$$

Pseudo Solution: $(0, 1, 0)$

CONSENFOLP and CONSENFOPS

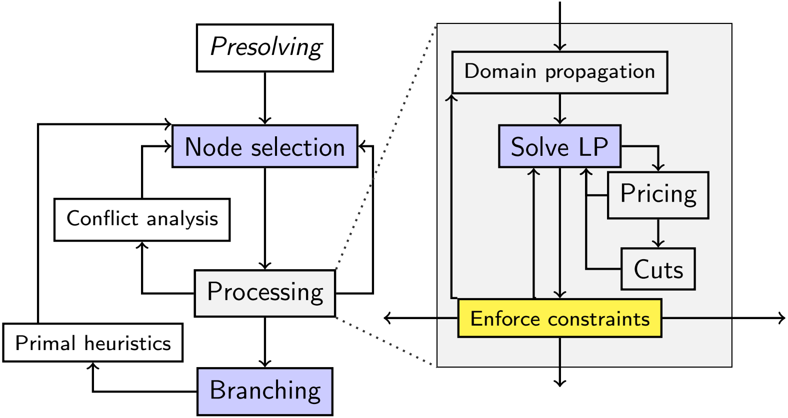
LP solution may violate a constraint not contained in the relaxation.

> Enforcement callbacks are necessary for a correct implementation!

In addition, they can resolve an infeasibility by ...

- reducing a variable's domain,
- separating a cutting plane (may use integrality),
- adding a (local) constraint,
- creating a branching,
- concluding that the subproblem is infeasible and can be cut off, or
- just saying "solution infeasible".

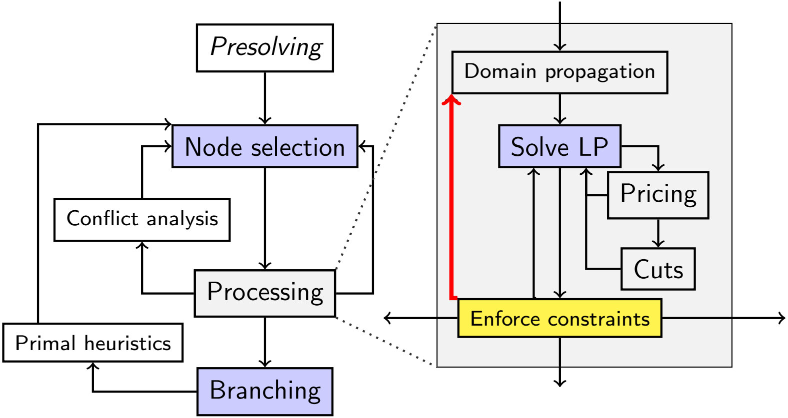
CONSENFOLP and CONSENFOPS



Enforcement result of constraint handler:

- reduced domain
- added constraint
- added cut
- branched
- cutoff
- infeasible
- feasible

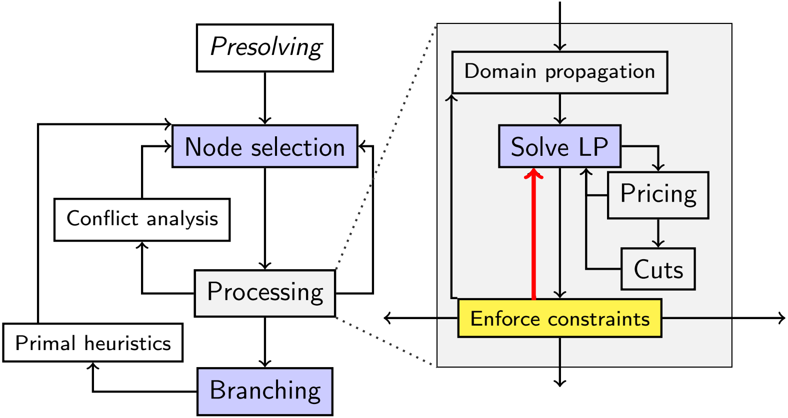
CONSENFOLP and CONSENFOPS



Enforcement result of constraint handler:

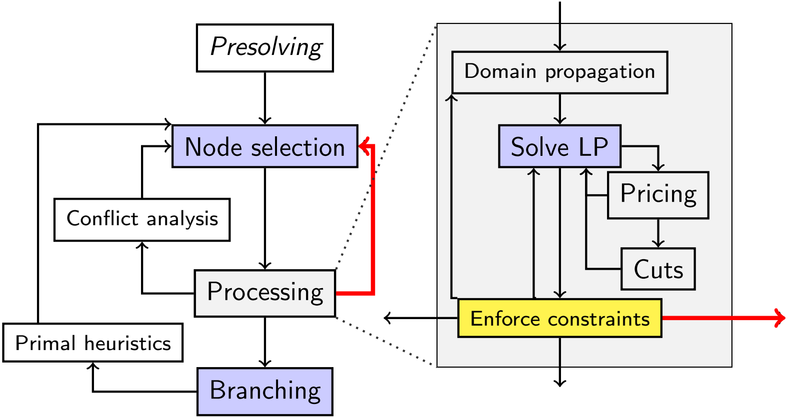
- ▷ reduced domain
- ▷ added constraint
- ▷ added cut
- ▷ branched
- ▷ cutoff
- ▷ infeasible
- ▷ feasible

CONSENFOLP and CONSENFOPS



Enforcement result of constraint handler:

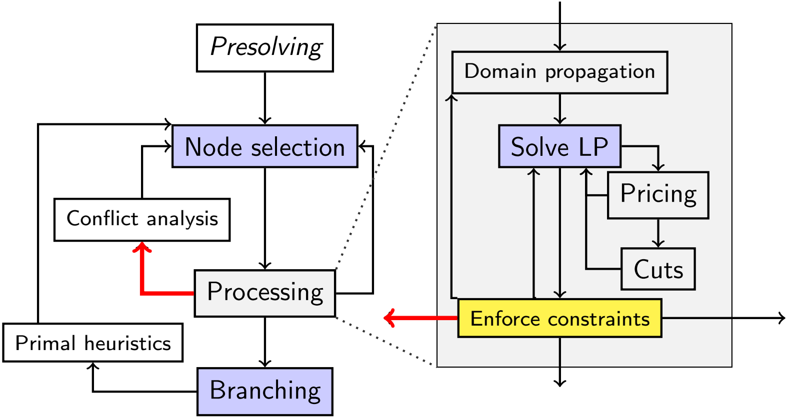
- reduced domain
- added constraint
- added cut
- branched
- cutoff
- infeasible
- feasible



Enforcement result of constraint handler:

- reduced domain
- added constraint
- added cut
- branched
- cutoff
- infeasible
- feasible

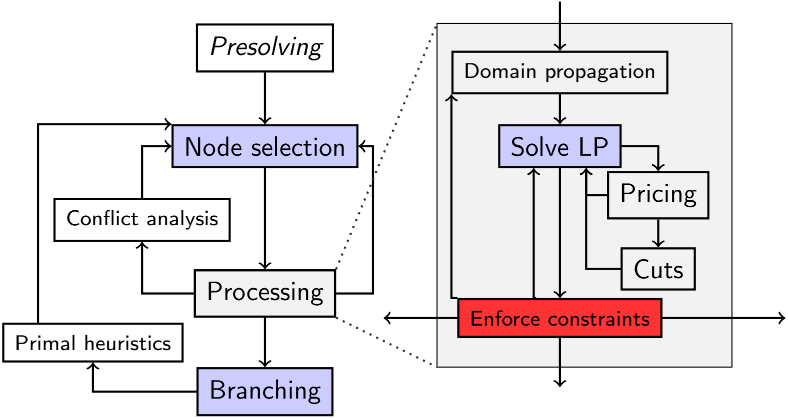
CONSENFOLP and CONSENFOPS



Enforcement result of constraint handler:

- reduced domain
- added constraint
- added cut
- branched
- cutoff
- infeasible
- feasible

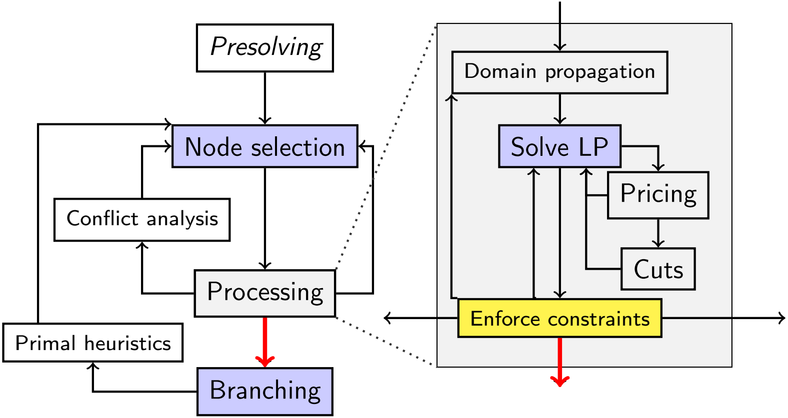
CONSENFOLP and CONSENFOPS



Enforcement result of constraint handler:

- reduced domain
- added constraint
- added cut
- branched
- cutoff
- infeasible
- feasible

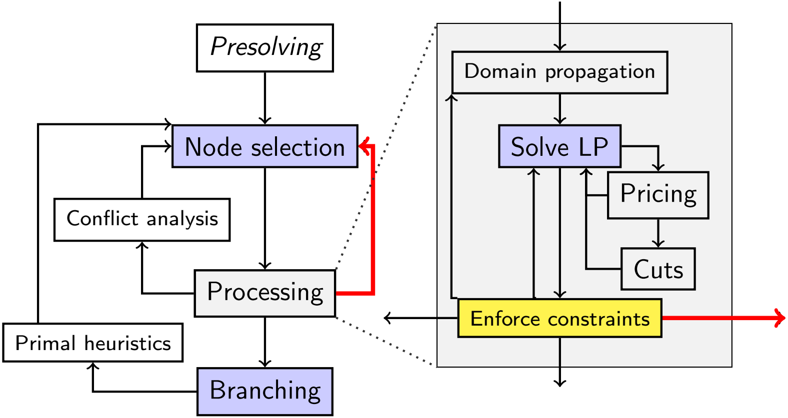
CONSENFOLP and CONSENFOPS



Enforcement result of constraint handler:

- reduced domain
- added constraint
- added cut
- branched
- cutoff
- infeasible
- feasible

CONSENFOLP and CONSENFOPS



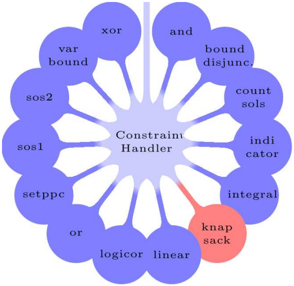
Enforcement result of constraint handler:

- reduced domain
- added constraint
- added cut
- branched
- cutoff
- infeasible
- feasible

Ingredients of a Constraint Handler

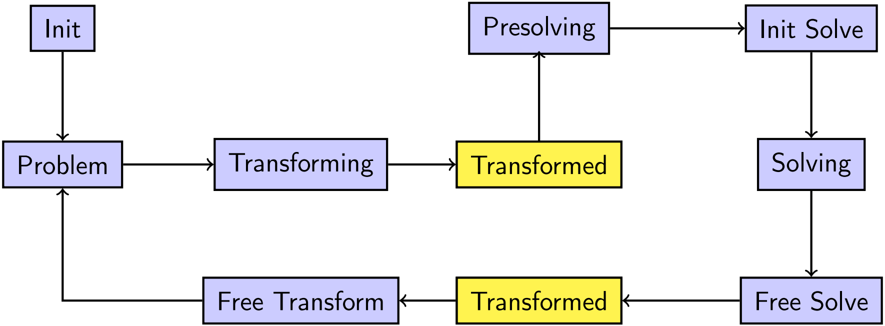
Callback Methods

- ▷ Fundamental:
  - CONSLOCK
  - CONSCHECK
  - CONSENFOLP, CONSENFOPS
- ▷ Additional:
  - CONSINIT..., CONSEXIT...
  - CONSSEPALP, CONSSEPASOL
  - CONSPROP, CONSRESPROP, CONSPRESOL
  - CONSACTIVE, CONSDEACTIVE, CONSENABLE, CONSDISABLE, CONSTRANS, CONSDELETE, CONSFREE
  - CONSPRINT, CONSCOPY, CONSPARSE



Further Ingredients

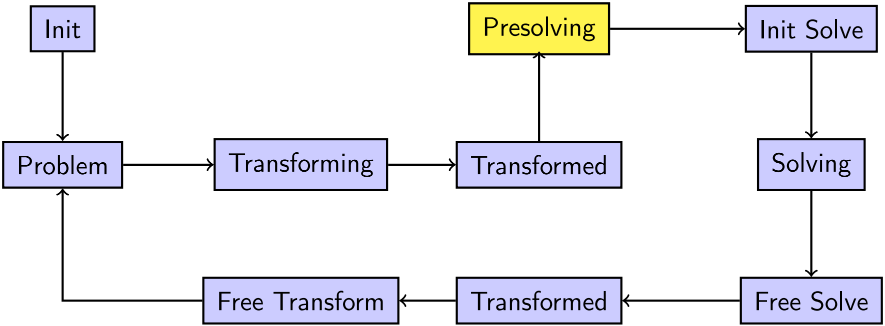
- ▷ Private data
- ▷ Interface methods
- ▷ Properties/Parameters



CONSINIT and CONSEXIT:

- called after problem was transformed /
  before transformed problem is freed
- initialize and free statistics in `SCIP_ConshdlrData`

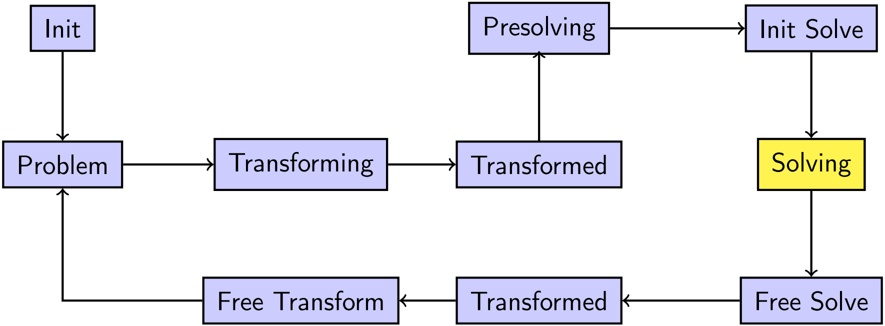
CONSINIT..., CONSEXIT...



CONSINITPRE and CONSEXITPRE:

- called before presolving starts / after presolving is finished
- initialize and free presolving data

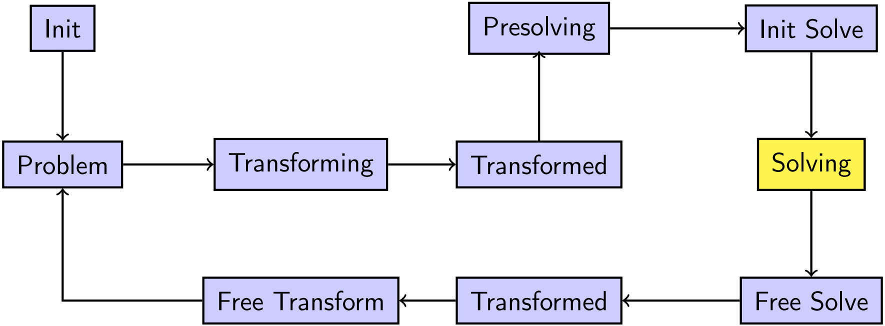
CONSINIT..., CONSEXIT...



CONSINITSOL and CONSEXITSOL:

- called before branch-and-bound process starts /
  before branch-and-bound process is freed
- initialize and release branch-and-bound specific data

CONSINIT..., CONSEXIT...



CONSINITLP:

- called before first LP relaxation is solved
- add linear relaxation of all "initial" constraints to the LP relaxation

Feasible region of 0-1 knapsack problem:

$$\{\ x \in \{0,1\}^{|N|}\ :\ \sum_{j \in N} a_j x_j \le b\ \}$$

Minimal Cover: $C \subseteq N$

- ▷ $\sum_{j \in C} a_j > b$
- ▷ $\sum_{j \in C \setminus \{i\}} a_j \le b \quad \forall\, i \in C$

Minimal Cover Inequality

$$\sum_{j \in C} x_j \le |C| - 1$$

$5x_1 + 6x_2 + 2x_3 + 2x_4 \le 8$

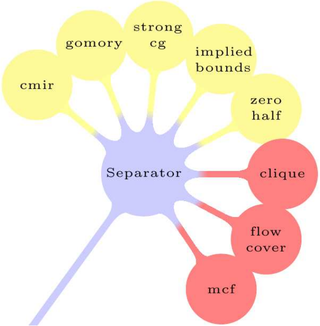
Minimal cover: $C = \{2,3,4\}$

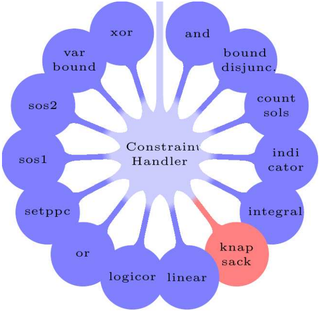
Minimal cover inequality:

$x_2 + x_3 + x_4 \le 2$

separated in
knapsack constraint handler

CONSSEPALP, CONSSEPASOL





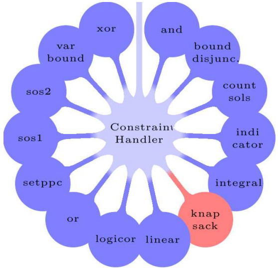
Separation is implemented in separators and constraint handlers.

1. Separators with SEPA_PRIORITY ≥ 0 (decreasing order)
2. Constraint handlers (decreasing order of CONSHDLR_SEPAPRIORITY)
3. Separators with SEPA_PRIORITY < 0 (decreasing order)

Ingredients of a Constraint Handler

Callback Methods

- ▷ Fundamental:
  - CONSLOCK
  - CONSCHECK
  - CONSENFOLP, CONSENFOPS
- ▷ Additional:
  - CONSINIT..., CONSEXIT...
  - CONSSEPALP, CONSSEPASOL
  - CONSPROP, CONSRESPROP, CONSPRESOL
  - CONSACTIVE, CONSDEACTIVE, CONSENABLE, CONSDISABLE
  - CONSTRANS, CONSDELETE, CONSFREE
  - CONSPRINT, CONSPARSE, CONSCOPY

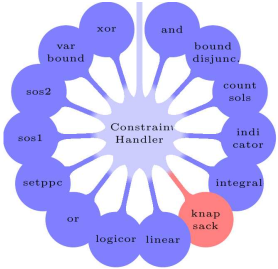


- ▷ Domain propagation (during subproblem processing)
- ▷ Reason for domain reductions (for conflict analysis)
- ▷ Presolving (before processing root node)

Ingredients of a Constraint Handler

Callback Methods

- **Fundamental:**
  - CONSLOCK
  - CONSCHECK
  - CONSENFOLP, CONSENFOPS
- **Additional:**
  - CONSINIT..., CONSEXIT...
  - CONSSEPALP, CONSSEPASOL
  - CONSPROP, CONSRESPROP, CONSPRESOL
  - CONSACTIVE, CONSDEACTIVE, CONSENABLE, CONSDISABLE
  - CONSTRANS, CONSDELETE, CONSFREE
  - CONSPRINT, CONSPARSE, CONSCOPY



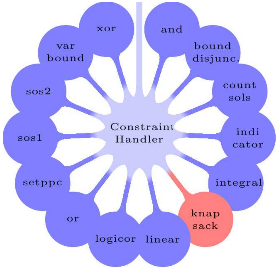
Called whenever ...

- SCIP enters/leaves subtree where local constraint exists
- constraint is enabled/disabled (no propagation, no separation)

Ingredients of a Constraint Handler

Callback Methods

- ▷ Fundamental:
  - CONSLOCK
  - CONSCHECK
  - CONSENFOLP, CONSENFOPS
- ▷ Additional:
  - CONSINIT..., CONSEXIT...
  - CONSSEPALP, CONSSEPASOL
  - CONSPROP, CONSRESPROP, CONSPRESOL
  - CONSACTIVE, CONSDEACTIVE, CONSENABLE, CONSDISABLE
  - CONSTRANS, CONSDELETE, CONSFREE
  - CONSPRINT, CONSPARSE, CONSCOPY



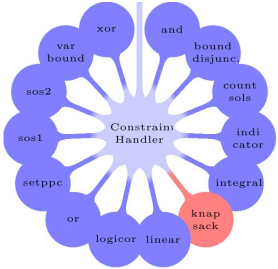
- ▷ Displaying, parsing problems
- ▷ Copying problems between different SCIP environments

Ingredients of a Constraint Handler

Callback Methods

- ▷ Fundamental:
  - CONSLOCK
  - CONSCHECK
  - CONSENFOLP, CONSENFOPS

- ▷ Additional:
  - CONSINIT..., CONSEXIT...
  - CONSSEPALP, CONSSEPASOL
  - CONSPROP, CONSRESPROP, CONSPRESOL
  - CONSACTIVE, CONSDEACTIVE, CONSENABLE, CONSDISABLE
  - CONSTRANS, CONSDELETE, CONSFREE
  - CONSPRINT, CONSPARSE, CONSCOPY



Further Ingredients

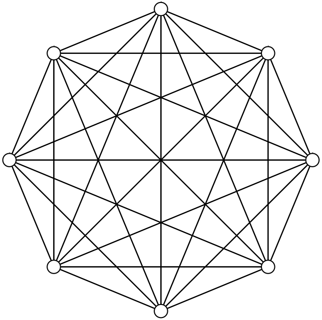
- ▷ Private data
- ▷ Interface methods
- ▷ Properties/Parameters

Exercise: Traveling Salesman Problem (TSP)

Definition

Given a complete graph $G = (V, E)$ with edge lengths c_e.

Find a Hamiltonian cycle (cycle containing all nodes, tour) of minimum length.



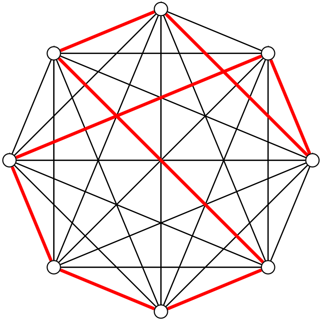
K_8

Exercise: Traveling Salesman Problem (TSP)

Definition

Given a complete graph $G = (V, E)$ with edge lengths c_e.

Find a Hamiltonian cycle (cycle containing all nodes, tour) of minimum length.



K_8

Exercise: TSP

MIP Formulation

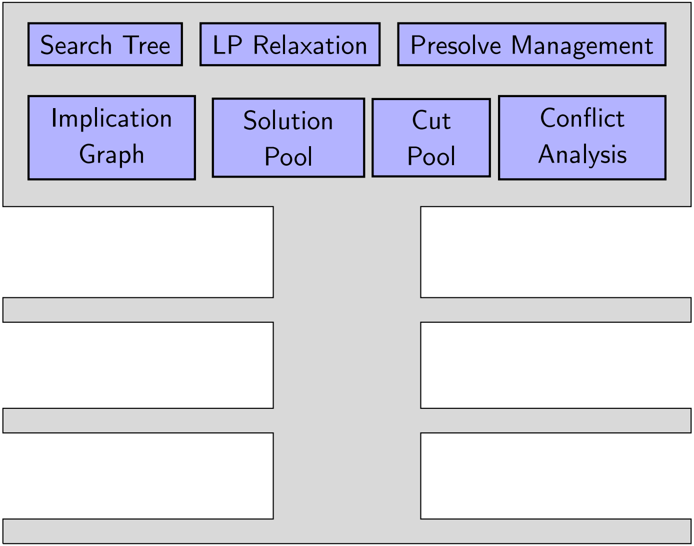
$$
\begin{aligned}
\min \quad & \sum_{e \in E} c_e \, x_e \\
s.t. \quad & \sum_{e \in \delta(v)} x_e = 2 \quad \forall\, v \in V \\
& \sum_{e \in \delta(S)} x_e \geq 2 \quad \forall\, S \subset V, S \neq \varnothing \\
& x_e \in \{0,1\} \quad \forall\, e \in E.
\end{aligned}
$$

CIP Formulation

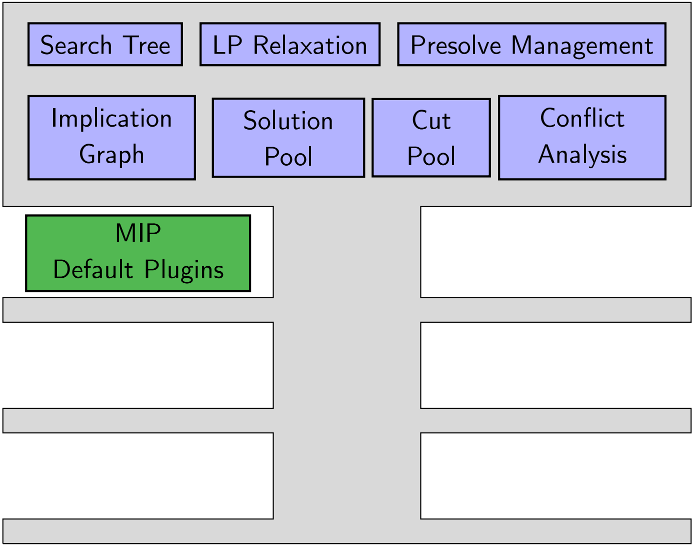
$$
\begin{aligned}
\min \quad & \sum_{e \in E} c_e \, x_e \\
s.t. \quad & \sum_{e \in \delta(v)} x_e = 2 \qquad \forall\, v \in V \\
& \mathrm{nosubtour}(G, x) \\
& x_e \in \{0,1\} \qquad \forall\, e \in E.
\end{aligned}
$$

$\mathrm{nosubtour}(G, x) \Leftrightarrow \nexists\, C \subseteq \{e \in E \mid x_e = 1\} : C$ is a cycle of length $|C| < |V|$

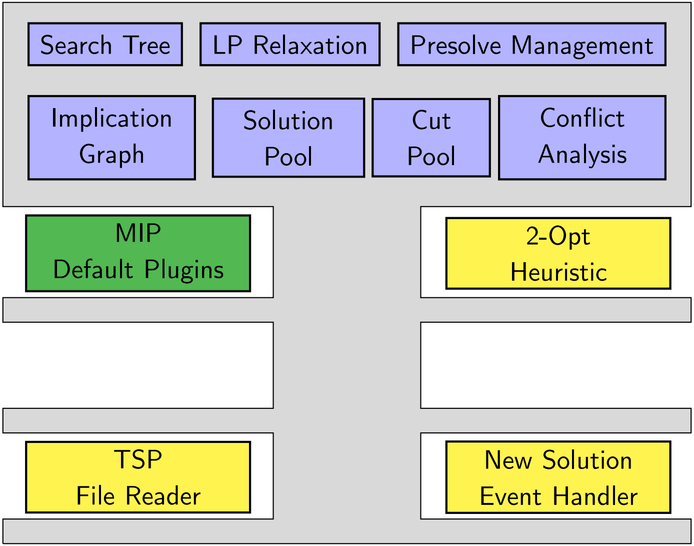
Exercise: TSP



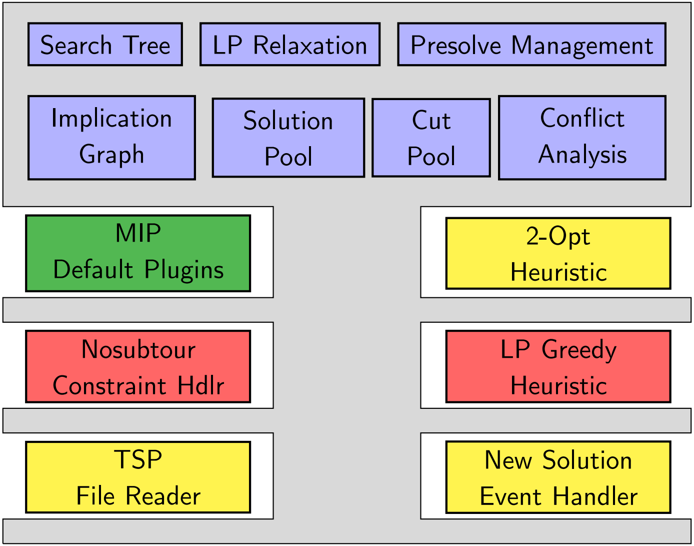
Exercise: TSP



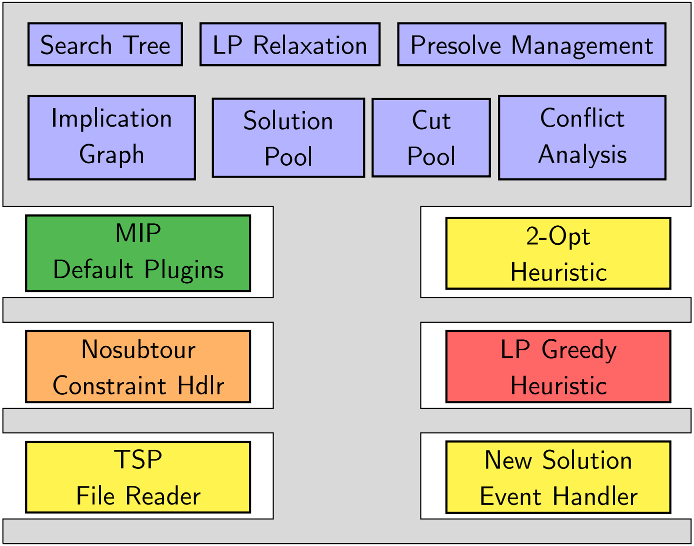
Exercise: TSP



Exercise: TSP



Exercise: TSP

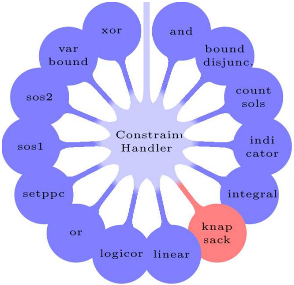


Exercise: TSP

Callback Methods

- **Fundamental:**
  - CONSLOCK
  - CONSCHECK
  - CONSENFOLP, CONSENFOPS

- **Additional:**
  - CONSINIT..., CONSEXIT...
  - CONSSEPALP, CONSSEPASOL
  - CONSPROP, CONSRESPROP, CONSPRESOL
  - CONSACTIVE, CONSDEACTIVE, CONSENABLE, CONSDISABLE
  - CONSTRANS, CONSDELETE, CONSFREE
  - CONSPRINT, CONSPARSE, CONSCOPY



Further Ingredients

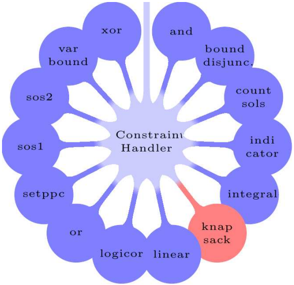
- Private data
- Interface methods
- Properties

Exercise: TSP

Callback Methods

- **Fundamental:**
  - CONSLOCK
  - CONSCHECK
  - CONSENFOLP, CONSENFOPS

- **Additional:**
  - CONSSEPALP, CONSSEPASOL
  - CONSTRANS, CONSDELETE



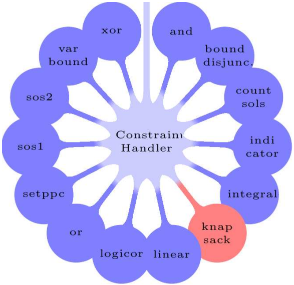
Further Ingredients

- Private data
- Interface methods
- Properties

Exercise: TSP

Callback Methods

- ▷ Fundamental:
  - CONSLOCK
  - CONSCHECK
  - CONSENFOLP, CONSENFOPS
- ▷ Additional:
  - CONSSEPALP, CONSSEPASOL
  - CONSTRANS, CONSDELETE



Further Ingredients

- ▷ Private data
- ▷ Interface methods
- ▷ Properties

CONSSEPALP, CONSSEPASOL

Separation Problem

Given complete graph $G = (V, E)$ and $x^* \in [0, 1]^{|E|}$.

- Decide whether x^* satisfies all subtour elimination constraints.
- If not, find violated subtour elimination constraint.

Separation Problem

Given complete graph $G = (V, E)$ and $x^* \in [0, 1]^{|E|}$.

- ▷ Decide whether x^* satisfies all subtour elimination constraints.
- ▷ If not, find violated subtour elimination constraint.

Trivial observation:

- ▷ Consider $G = (V, E)$ with edge capacities x^*_e.
- ▷ x^* violates at least one subtour elimination constraint
  $\Leftrightarrow \exists$ cut $\delta(S)$ with capacity $x^*(\delta(S)) < 2$.

Separation Problem

Given complete graph $G = (V, E)$ and $x^* \in [0,1]^{|E|}$.

- Decide whether x^* satisfies all subtour elimination constraints.
- If not, find violated subtour elimination constraint.

Idea of separation algorithm:

- $\forall\, s, t \in V$: Find (s,t)-cut $\delta(S)$ of minimum capacity
- If all cut capacities ≥ 2, all subtour elimination constraints satisfied

$\longrightarrow$ $\binom{|V|}{2}$ times MaxFlow-MinCut algo

Separation Problem

Given complete graph $G = (V, E)$ and $x^* \in [0,1]^{|E|}$.

- Decide whether x^* satisfies all subtour elimination constraints.
- If not, find violated subtour elimination constraint.

Idea of separation algorithm:

- $\forall\, s, t \in V$: Find (s,t)-cut $\delta(S)$ of minimum capacity
- If all cut capacities ≥ 2, all subtour elimination constraints satisfied

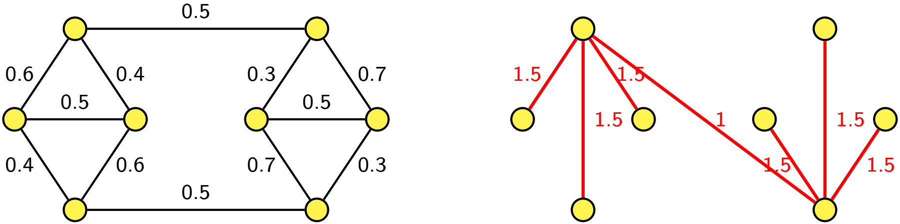
$\longrightarrow$ $|V| - 1$ times MaxFlow-MinCut algo within Gomory-Hu-Algorithm

Result of Gomory-Hu-Algorithm

Gomory-Hu-Tree T

For all $s, t \in V$:

- capacity of minimum (s, t)-cut in G:
  minimum label f_e of all edges e in unique (s, t)-path in T
- minimum (s, t)-cut in G:
  bipartition of V obtained be deleting this edge from T

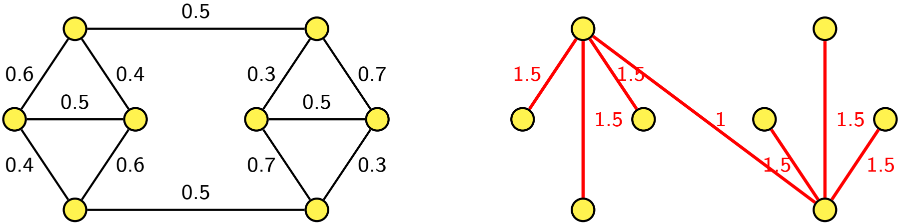


Result of Gomory-Hu-Algorithm

Gomory-Hu-Tree T

For all $s, t \in V$:

- capacity of minimum (s, t)-cut in G:
  minimum label f_e of all edges e in unique (s, t)-path in T
- minimum (s, t)-cut in G:
  bipartition of V obtained be deleting this edge from T



`ghc_tree()` returns all minimum (s, t)-cuts with capacity $\leq 2 - \texttt{minviol}$

Documentation

- Constraint Integer Programming
  Tobias Achterberg, Ph.D. dissertation, TU Berlin, 2007
- Primal Heuristics for Mixed Integer Programs
  Timo Berthold, Diploma thesis, TU Berlin, 2006
- Implementation of Cutting Plane Separators for Mixed Integer Programs
  Kati Wolter, Diploma thesis, TU Berlin, 2006

- http://scip.zib.de
  Doxygen documentation, HowTos, FAQ
- source code
  `scip.h`, `pub_*.h`, `type_*.h`

Please, . . .

- build groups of 3 people
  - laptop owner
  - C expert
  - IP expert
- raise your hand, if you get stuck.

Branching, Presolving, and Constraint Handlers

CO@Work Berlin

Timo Berthold and Kati Wolter

09/26/2009

Berlin Mathematical School

DFG Research Center MATHEON
Mathematics for key technologies