Branching, Presolving, and Constraint Handlers

CO@Work Berlin

Timo Berthold and Kati Wolter

09/26/2009
Outline

Branching Rules

Node Selection

Presolving

Constraint Handlers
Outline

Branching Rules

Node Selection

Presolving

Constraint Handlers
### How do we solve discrete optimization problems?

<table>
<thead>
<tr>
<th><strong>Mixed Integer Programming</strong></th>
<th><strong>SAT Isfiability problems</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ LP relaxation</td>
<td>▶ conflict analysis</td>
</tr>
<tr>
<td>▶ cutting planes</td>
<td>▶ periodic restarts</td>
</tr>
<tr>
<td>▶ branch-and-bound</td>
<td>▶ branch-and-bound</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Constraint Programming</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ domain propagation</td>
</tr>
<tr>
<td>▶ symmetry handling</td>
</tr>
<tr>
<td>▶ branch-and-bound</td>
</tr>
</tbody>
</table>
How do we solve discrete optimization problems?

<table>
<thead>
<tr>
<th>Mixed Integer Programming</th>
<th>SATisfiability problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>LP relaxation</td>
<td>conflict analysis</td>
</tr>
<tr>
<td>cutting planes</td>
<td>periodic restarts</td>
</tr>
<tr>
<td>branch-and-bound</td>
<td>branch-and-bound</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constraint Programming</th>
<th>Branch-and-bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>domain propagation</td>
<td>Land, Doig [1960], Dakin [1965]</td>
</tr>
<tr>
<td>symmetry handling</td>
<td>“smart” enumeration</td>
</tr>
<tr>
<td>branch-and-bound</td>
<td>use dual bounds</td>
</tr>
</tbody>
</table>
LP-based Branch-and-Bound (Algorithm)

### Steps

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

```plaintext
\[ \mathcal{L} \leftarrow \{P\}, \quad U \leftarrow \infty, \quad x^{IP} \leftarrow \text{NULL} \]

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 \( P_i \), \( \mathcal{L}_{loc} \leftarrow c(x^{LP}) \) or \( \infty \)

if \( \mathcal{L}_{loc} \geq U \) then goto line 2

if \( x^{LP} \in P \) then \( x^{IP} \leftarrow x^{LP}, \quad U \leftarrow \mathcal{L}_{loc}, \quad \text{goto line 2} \)

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

Steps

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

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

If $L = \emptyset$ then return $x^{IP}$ and $U$

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

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

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

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

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

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

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

*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 \( P_i \), \( L_{loc} \leftarrow c(x^{LP}) \) or \( \infty \)

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

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

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

#### Steps

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

\[
\begin{align*}
\mathcal{L} & \leftarrow \{P\}, \quad U \leftarrow \infty, \quad x^{\text{IP}} \leftarrow \text{NULL} \\
\text{if } \mathcal{L} = \emptyset & \text{ then return } x^{\text{IP}} \text{ and } U \\
\text{Select } P_i & \in \mathcal{L}, \quad \mathcal{L} \leftarrow \mathcal{L}\backslash\{P_i\} \\
\text{Solve LP relaxation of } P_i, & \quad L_{\text{loc}} \leftarrow c(x^{\text{LP}}) \text{ or } \infty \\
\text{if } L_{\text{loc}} & \geq U \text{ then goto line 2} \\
\text{if } x^{\text{LP}} & \in P \text{ then } x^{\text{IP}} \leftarrow x^{\text{LP}}, \quad U \leftarrow L_{\text{loc}}, \quad \text{goto line 2} \\
\text{Select } j & \in I: x_j^{\text{LP}} \notin \mathbb{Z}. \text{ Split } P_i \text{ into } P_{2i+1} := P_i \cup \{x_j \leq \lfloor x_j^{\text{LP}} \rfloor\}, \text{ and} \\
P_{2i+2} & := P_i \cup \{x_j \geq \lceil x_j^{\text{LP}} \rceil\}, \quad \mathcal{L} \leftarrow \mathcal{L} \cup \{P_{2i+1}, P_{2i+2}\}, \quad \text{goto line 2}
\end{align*}
\]
LP-based Branch-and-Bound (Algorithm)

Steps

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

\[
L \leftarrow \{P\}, \quad U \leftarrow \infty, \quad x^{\text{IP}} \leftarrow \text{NULL}
\]

if \( L = \emptyset \) then return \( x^{\text{IP}} \) and \( U \)

Select \( P_i \in L, \quad L \leftarrow L \setminus \{P_i\} \)

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

if \( L_{\text{loc}} \geq U \) then goto line 2

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

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 \( P_{2i+2} := P_i \cup \{x_j \geq \lceil x_j^{\text{LP}} \rceil\} \), \( L \leftarrow L \cup \{P_{2i+1}, P_{2i+2}\} \), 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

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

if \( \mathcal{L} = \emptyset \) then return \( x^{IP} \) and \( U \)

Select \( P_i \in \mathcal{L} \), \( \mathcal{L} \leftarrow \mathcal{L}\backslash\{P_i\} \)

Solve LP relaxation of \( P_i \), \( L_{loc} \leftarrow c(x^{LP}) \) or \( \infty \)

if \( L_{loc} \geq U \) then goto line 2

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

Select \( j \in I: x_j^{LP} \notin \mathbb{Z} \). Split \( P_i \) into \( P_{2i+1} := P_i \cup \{x_j \leq \lfloor x_j^{LP} \rfloor\} \), and \( P_{2i+2} := P_i \cup \{x_j \geq \lceil x_j^{LP} \rceil\} \), \( \mathcal{L} \leftarrow \mathcal{L} \cup \{P_{2i+1}, P_{2i+2}\} \), goto line 2
LP-based Branch-and-Bound (Algorithm)

Steps

|----------------------|-------------------|---------------------|-------------|----------------------|-------------|

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

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 \(P_i\), \(L_{loc} \leftarrow c(x^{LP})\) or \(\infty\)

if \(L_{loc} \geq U\) then goto line 2

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

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

[Image of a colorful diagram showing a polyhedron with nodes and constraints]
### Steps

<table>
<thead>
<tr>
<th></th>
<th>Abort criterion</th>
<th>Solve relaxation</th>
<th>Bounding</th>
<th>Feasibility check</th>
<th>Branching</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>Node selection</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

#### Steps

|---|-------------------|---------------------|----------------------|

---

*T. Berthold and K. Wolter (ZIB)*
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

![Diagram showing steps of LP-based Branch-and-Bound](image-url)
**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

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Abort criterion</td>
<td>3.</td>
<td>Solve relaxation</td>
</tr>
<tr>
<td>5.</td>
<td>Feasibility check</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.</td>
<td>Branching</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

T. Berthold and K. Wolter (ZIB)

Branching, Presolving, and Conshdlr

09/26/2009 6 / 60
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)

<table>
<thead>
<tr>
<th>Steps</th>
<th></th>
<th></th>
</tr>
</thead>
</table>

![Branching Tree Diagram]

\( \chi^{\text{IP}} \)
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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Abort criterion</td>
<td>3.</td>
</tr>
<tr>
<td>5.</td>
<td>Feasibility check</td>
<td>6.</td>
</tr>
</tbody>
</table>

**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

Branching, Presolving, and Conshdlr

T. Berthold and K. Wolter (ZIB)
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

|---|--------------------|---------------------|----------------------|

![Branching tree diagram]

T. Berthold and K. Wolter (ZIB)
LP-based Branch-and-Bound (Colorful Picture)

### Steps

<table>
<thead>
<tr>
<th></th>
<th>Abort criterion</th>
<th>Solve relaxation</th>
<th>Feasibility check</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Node selection</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Solve relaxation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Bounding</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Feasibility check</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Branching</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Branching Tree and Relaxation](image)

- ● Node selection
- X IP
- ∞ Node expansion
- x Abort criterion

T. Berthold and K. Wolter (ZIB)
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>

<table>
<thead>
<tr>
<th>node</th>
<th>left</th>
<th>LP iter</th>
<th>mem</th>
<th>dualbound</th>
<th>primalbound</th>
<th>gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>2756</td>
<td>40M</td>
<td>5.553544e+04</td>
<td>--</td>
<td>Inf</td>
</tr>
<tr>
<td>50</td>
<td>49</td>
<td>32939</td>
<td>41M</td>
<td>5.576070e+04</td>
<td>--</td>
<td>Inf</td>
</tr>
<tr>
<td>* 73</td>
<td>8</td>
<td>47394</td>
<td>41M</td>
<td>5.576165e+04</td>
<td>5.641900e+04</td>
<td>1.18%</td>
</tr>
<tr>
<td>76</td>
<td>9</td>
<td>47533</td>
<td>37M</td>
<td>5.576165e+04</td>
<td>5.640000e+04</td>
<td>1.14%</td>
</tr>
<tr>
<td>* 79</td>
<td>8</td>
<td>47636</td>
<td>37M</td>
<td>5.576165e+04</td>
<td>5.638500e+04</td>
<td>1.12%</td>
</tr>
<tr>
<td>* 82</td>
<td>7</td>
<td>48134</td>
<td>37M</td>
<td>5.580334e+04</td>
<td>5.636300e+04</td>
<td>1.00%</td>
</tr>
<tr>
<td>* 98</td>
<td>4</td>
<td>50423</td>
<td>36M</td>
<td>5.584525e+04</td>
<td>5.613800e+04</td>
<td>0.52%</td>
</tr>
<tr>
<td>100</td>
<td>4</td>
<td>50529</td>
<td>35M</td>
<td>5.584525e+04</td>
<td>5.613800e+04</td>
<td>0.52%</td>
</tr>
<tr>
<td>* 128</td>
<td>2</td>
<td>52075</td>
<td>35M</td>
<td>5.584525e+04</td>
<td>5.613700e+04</td>
<td>0.52%</td>
</tr>
<tr>
<td>150</td>
<td>2</td>
<td>54032</td>
<td>35M</td>
<td>5.593390e+04</td>
<td>5.613700e+04</td>
<td>0.36%</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Node</th>
<th>Left</th>
<th>Objective</th>
<th>IInf</th>
<th>Best Integer</th>
<th>Best Node</th>
<th>Gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1.0000</td>
<td>1</td>
<td></td>
<td></td>
<td>1.0000</td>
</tr>
<tr>
<td>*</td>
<td>0+</td>
<td>0</td>
<td>3.0000</td>
<td>1.0000</td>
<td></td>
<td>66.67%</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>1.0000</td>
<td>1</td>
<td>3.0000</td>
<td>1.0000</td>
<td>66.67%</td>
</tr>
</tbody>
</table>

...
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.

<table>
<thead>
<tr>
<th>Node</th>
<th>Left</th>
<th>Objective</th>
<th>IInf</th>
<th>Best Integer</th>
<th>Best Node</th>
<th>Gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1.0000</td>
<td>1</td>
<td>1.0000</td>
<td>1.0000</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>0+</td>
<td>3.0000</td>
<td>1</td>
<td>1.0000</td>
<td>66.67%</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>1.0000</td>
<td>1</td>
<td>3.0000</td>
<td>1.0000</td>
<td>66.67%</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>224000000</td>
<td>2</td>
<td>1.0000</td>
<td>1</td>
<td>3.0000</td>
<td>1.0000</td>
<td>66.67%</td>
</tr>
</tbody>
</table>

Solution pool: 1 solution saved.

MIP - Time limit exceeded, integer feasible: Objective = 3.000000000e+00
Current MIP best bound = 1.000000000e+00 (gap = 2, 66.67%)
Solution time = 3600.01 sec. Iterations = 224141556 Nodes = 224141558 (2)
Minimize
obj: x3
Subject To
\[ c1: 12 x_1 + 9 x_2 - x_3 = 0 \]
Bounds
-\( \infty \) <= x_1
-\( \infty \) <= x_2
1 <= x_3
General
x_1
x_2
x_3
End
Minimize
obj: x3
Subject To
c1: 12 x1 + 9 x2 - x3 = 0
Bounds
-\text{inf} \leq x1
-\text{inf} \leq x2
1 \leq x3
General
x1
x2
x3
End

Not a CPLEX issue!
Other codes do not perform better!
Branch-And-Bound (Algorithm) continued

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

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 \( P_i \), \( L_{loc} \leftarrow c(x^{LP}) \) or \( \infty \)

if \( L_{loc} \geq U \) then goto line 2

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

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

Two degrees of freedom

1. Node selection
2. Variable selection (branching rule)
\[ \mathcal{L} \leftarrow \{ P \}, \quad U \leftarrow \infty, \quad x^{IP} \leftarrow \text{NULL} \]

If \( \mathcal{L} = \emptyset \) then return \( x^{IP} \) and \( U \)

Select \( P_i \in \mathcal{L} \), \( \mathcal{L} \leftarrow \mathcal{L} \backslash \{ P_i \} \)

Solve LP relaxation of \( P_i \), \( L_{loc} \leftarrow c(x^{LP}) \) or \( \infty \)

If \( L_{loc} \geq U \) then goto line 2

If \( x^{LP} \in P \) then \( x^{IP} \leftarrow x^{LP} \), \( U \leftarrow L_{loc} \), goto line 2

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

Two degrees of freedom

1. Node selection
2. Variable selection (branching rule)
**Same algorithm, different scope**

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

<table>
<thead>
<tr>
<th>MIP</th>
<th>SAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ improve LP value</td>
<td>▶ detect infeasibilities</td>
</tr>
<tr>
<td>▶ reduce LP complexity</td>
<td>▶ generate short conflicts</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CP</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ enable propagations</td>
</tr>
<tr>
<td>▶ reduce domains</td>
</tr>
</tbody>
</table>
Same algorithm, different scope

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

<table>
<thead>
<tr>
<th><strong>MIP</strong></th>
<th><strong>SAT</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ improve LP value</td>
<td></td>
</tr>
<tr>
<td>▶ reduce LP complexity</td>
<td></td>
</tr>
<tr>
<td>▶ detect infeasibilities</td>
<td></td>
</tr>
<tr>
<td>▶ generate short conflicts</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>CP</strong></th>
<th><strong>But</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ enable propagations</td>
<td></td>
</tr>
<tr>
<td>▶ reduce domains</td>
<td></td>
</tr>
<tr>
<td>Also in MIP, sometimes…</td>
<td></td>
</tr>
<tr>
<td>▶ there’s no objective</td>
<td></td>
</tr>
<tr>
<td>▶ instances are infeasible</td>
<td></td>
</tr>
<tr>
<td>▶ combinatorial structure</td>
<td></td>
</tr>
</tbody>
</table>
Same algorithm, different scope

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

<table>
<thead>
<tr>
<th>MIP</th>
<th>SAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ improve LP value</td>
<td>▶ detect infeasibilities</td>
</tr>
<tr>
<td>▶ reduce LP complexity</td>
<td>▶ generate short conflicts</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CP</th>
<th>But</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ enable propagations</td>
<td>Also in MIP, sometimes…</td>
</tr>
<tr>
<td>▶ reduce domains</td>
<td>▶ there’s no objective</td>
</tr>
<tr>
<td></td>
<td>▶ instances are infeasible</td>
</tr>
<tr>
<td></td>
<td>▶ combinatorial structure</td>
</tr>
</tbody>
</table>

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

\[ \zeta - (x_3) = 4 - 2.7 = 2.0 \]

\[ \psi - (x_3) = \zeta - 1(x_3) + ... + \zeta - n(x_3) \]

\[ \psi - (x_3) \cdot \text{frac}(x_3) = 4 \cdot 0.2 = 0.8 \]

\[ x_3 = 7.4 \]

\[ c = 2 \]
What are pseudocosts?

Estimating the objective

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

- other values at other nodes:
  - average objective gain
  - \( \psi - (x_3) = \zeta - 1(x_3) + \ldots + \zeta - n(x_3) \)
  - \( n = 5 \)
  - \( = 4 \)

- estimate increase of objective by pseudocosts and fractionality:
  - \( \psi - (x_3) \cdot \text{frac}(x_3) = 4 \cdot 0.2 = 0.8 \)
  - \( \psi + (x_3)(1 - \text{frac}(x_3)) = 7.6 \)
What are pseudocosts?

Estimating the objective

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

\[ x_3 = 7.4 \]
\[ c = 2 \]
\[ x_3 \geq 8 \]
\[ c = 8 \]
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
    - $\psi^-_1(x_3) = 4$, $\psi^+_1(x_3) = 9.5$
  - 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:
  \( \psi^- (x_3) \cdot \text{frac}(x_3) \)
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:
  - $\psi^{-}(x_3) \cdot \text{frac}(x_3) = 4 \cdot 0.2 = 0.8,$
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
    - $\psi^-_3(x_3) = 4$, $\psi^+_3(x_3) = 9.5$

- **estimate** increase of objective by pseudocosts and fractionality:
  - $\psi^-_3(x_3) \cdot \text{frac}(x_3) = 4 \cdot 0.2 = 0.8$
  - and $\psi^+_3(x_3)(1 - \text{frac}(x_3)) = 7.6$
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

\[
\begin{align*}
  x_1 + x_2 &= 1 \\
  x_1 + x_3 + x_4 &\leq 1 \\
  -x_1 + z &\geq 3 \\
  z &\in \mathbb{Z}_+ \\
  x_i &\in \{0, 1\}
\end{align*}
\]

\[s_{j}^{\text{infer}}(-) = 2\]

\[x_1 = 0 \Rightarrow x_2 = 1 \Rightarrow z \geq 3\]
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

\[
\begin{align*}
    x_1 + x_2 &= 1 \\
    x_1 + x_3 + x_4 &\leq 1 \\
    -x_1 + z &\geq 3 \\
    z &\in \mathbb{Z}_+ \\
    x_i &\in \{0, 1\}
\end{align*}
\]

\[
\begin{align*}
    x_1 = 1 &\Rightarrow x_2 = 0 \\
    &\Rightarrow x_3 = 0 \\
    &\Rightarrow x_4 = 0 \\
    s_j^{\text{infer}}(+) = 3
\end{align*}
\]
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 (≈ 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

- Reliability (MIP)
- Inference (CP)
- VSIDS (SAT)

Hybrid Branching

- 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 $f: \mathbb{Q}_{\geq 0} \rightarrow [0, 1)$, $x \mapsto \frac{x}{x+1}$
- use a weighted sum of all criteria

In formulae:

$$s_j = \omega_{\text{pscost}} f \left( \frac{s_j}{s_{\emptyset}} \right) + \omega_{\text{infer}} f \left( \frac{s_j}{s_{\emptyset}} \right) + \omega_{\text{vsids}} f \left( \frac{s_j}{s_{\emptyset}} \right) + \omega_{\text{prune}} f \left( \frac{s_j}{s_{\emptyset}} \right) + \omega_{\text{conf}} f \left( \frac{s_j}{s_{\emptyset}} \right)$$
Hybrid branching

How the combination works:

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

In formulae:

$$s_j = \omega_{p\text{cost}} f \left( \frac{s_j}{s_{p\text{cost}}} \right) + \omega_{\text{infer}} f \left( \frac{s_j}{s_{\text{infer}}} \right) + \omega_{v\text{sid}s} f \left( \frac{s_j}{s_{v\text{sid}s}} \right) + \omega_{\text{prune}} f \left( \frac{s_j}{s_{\text{prune}}} \right) + \omega_{\text{conf}} f \left( \frac{s_j}{s_{\text{conf}}} \right)$$

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
  \[ \text{score}(x_j) = \max\{s_j^-, \epsilon\} \cdot \max\{s_j^+, \epsilon\} \quad (\epsilon = 10^{-6}) \]
- computational results: 10% faster
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

<table>
<thead>
<tr>
<th>test set</th>
<th>MIPLIB2003</th>
<th>Cor@l BP</th>
<th>Infeasible</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Nodes</td>
<td>Time</td>
</tr>
<tr>
<td>reliability</td>
<td>450.4</td>
<td>5091</td>
<td>803.6</td>
</tr>
<tr>
<td>hybrid</td>
<td>445.6</td>
<td>5051</td>
<td>735.0</td>
</tr>
<tr>
<td>ratio reli/hyb</td>
<td>1.01</td>
<td>1.01</td>
<td>1.09</td>
</tr>
</tbody>
</table>

**Result:** No difference / slight improvement for general MIPs

<table>
<thead>
<tr>
<th>test set</th>
<th>Cor@l BP</th>
<th>Infeasible</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Nodes</td>
</tr>
<tr>
<td>reliability</td>
<td>672.4</td>
<td>2145</td>
</tr>
<tr>
<td>hybrid</td>
<td>577.2</td>
<td>1681</td>
</tr>
<tr>
<td>ratio reli/hyb</td>
<td>1.16</td>
<td>1.28</td>
</tr>
</tbody>
</table>

**Result:** Medium / large improvement for special MIPs
Outline

Branching Rules

Node Selection

Presolving

Constraint Handlers
Node Selection

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

Techniques
- basic rules:
  - depth first search (DFS):
    → early feasible solutions
  - best bound search (BBS):
    → improve dual bound
  - best estimate search (BES):
    → 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 → subproblem processing is less expensive
- sibling node → 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)
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., $0^T x \leq b_i$, $b_i < 0 \Rightarrow$ 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
- ...

T. Berthold and K. Wolter (ZIB) Branching, Presolving, and Conshdlr 09/26/2009 29 / 60
Minimal and maximal activities

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

$$
\alpha_{\text{min}} := \min_{s.t. \ \ell \leq x \leq u} a^T x = \sum_{j, a_j > 0} a_j \ell_j + \sum_{j, a_j < 0} a_j u_j.
$$

the minimal activity of the linear constraint.
Minimal and maximal activities

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

$$\alpha_{\text{max}} := \max \sum_{j, a_j > 0} a_j u_j + \sum_{j, a_j < 0} a_j \ell_j.$$ 

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.

$$
\alpha_{\text{max}} := \max_{\substack{s.t. \quad \ell \leq x \leq u}} a^T x = \sum_{j,a_j>0} a_j u_j + \sum_{j,a_j<0} a_j \ell_j.
$$

the maximal activity of the linear constraint.

### First observation

- $\alpha_{\text{min}} > b \Rightarrow$ problem infeasible
- $\alpha_{\text{max}} \leq b \Rightarrow$ constraint redundant
Minimal and maximal activities

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

$$\alpha_{\text{min}} := \min_{a^T x \leq b, \ell \leq x \leq u} a^T x = \sum_{j, a_j > 0} a_j \ell_j + \sum_{j, a_j < 0} a_j u_j.$$  

the minimal activity of the linear constraint.

Bound strengthening

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

$$a^T x - a_k x_k + a_k x_k \leq b.$$
Minimal and maximal activities

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

$$\alpha_{\text{min}} := \min_{s.t. \ \ell \leq x \leq u} a^T x = \sum_{j, a_j > 0} a_j \ell_j + \sum_{j, a_j < 0} a_j u_j.$$  
the minimal activity of the linear constraint.

Bound strengthening

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

$$a^T x - a_k x_k + a_k x_k \leq b \iff x_k \leq \frac{b - (a^T x - a_k x_k)}{a_k}$$
Linear presolving

Minimal and maximal activities

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

\[
\alpha_{\min} := \min_{s.t. \ l \leq x \leq u} a^T x = \sum_{j,a_j>0} a_j \ell_j + \sum_{j,a_j<0} a_j u_j.
\]

the minimal activity of the linear constraint.

Bound strengthening

Let \( a_k > 0 \). For all feasible solutions \( x \), it holds that:

\[
a^T x - a_k x_k + a_k x_k \leq b \iff x_k \leq \frac{b - (a^T x - a_k x_k)}{a_k} \Rightarrow x_k \leq \frac{b - \alpha_{\min} + a_k \ell_k}{a_k}.
\]
Linear presolving

Minimal and maximal activities

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

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

the maximal activity of the linear constraint.

Coefficient tightening

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

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

can be reformulated as

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

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

\[
\alpha_{\text{max}} := \max_{s.t. \ \ell \leq x \leq u} a^T x = \sum_{j, a_j > 0} a_j u_j + \sum_{j, a_j < 0} a_j \ell_j.
\]

the maximal activity of the linear constraint.

Coefficient tightening

Consider \( 7x_1 + 8x_2 \leq 13 \). \( \alpha_{\text{max}} = 15 \)
Linear presolving

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

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

the maximal activity of the linear constraint.

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

$$(15 - 13)x_1 + 8x_2 \leq (15 - 7)$$
Minimal and maximal activities

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

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

the maximal activity of the linear constraint.

Coefficient tightening

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

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

Minimal and maximal activities

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

$$\alpha_{\text{max}} := \max_{a^T x \leq b} a^T x = \sum_{j,a_j > 0} a_j u_j + \sum_{j,a_j < 0} a_j \ell_j.$$ 

the maximal activity of the linear constraint.

Coefficient tightening

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

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

$$2x_1 + (10 - 8)x_2 \leq (10 - 8)$$
Mineral and maximal activities

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

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

the maximal activity of the linear constraint.

Coefficient tightening

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

- $2x_1 + 8x_2 \leq 8$  $\alpha_{\text{max}} = 10$
- $2x_1 + 2x_2 \leq 2$
Minimal and maximal activities

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

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

the maximal activity of the linear constraint.

Coefficient tightening

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

- $2x_1 + 8x_2 \leq 8 \quad \alpha_{\text{max}} = 10$
- $2x_1 + 2x_2 \leq 2$
- $x_1 + x_2 \leq 1$
Linear presolving

Minimal and maximal activities

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

$$\alpha_{\text{max}} := \max \left\{ a^T x \right\} \text{ s.t. } \ell \leq x \leq u = \sum_{j, a_j > 0} a_j u_j + \sum_{j, a_j < 0} a_j \ell_j.$$  
the maximal activity of the linear constraint.

Coefficient tightening

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

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

$2x_1 + 2x_2 \leq 2$

$x_1 + x_2 \leq 1$

setpack($x_1, x_2$)
probing: tentatively fix binary variables and propagate

dominance test: pairwise comparison of rows/columns

aggregation of equations with only two variables
\[ a_k x_k + a_j x_j = b \Rightarrow x_k = \frac{b}{a_i} - \frac{a_j}{a_k} x_j \]

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

Branching Rules

Node Selection

Presolving

Constraint Handlers
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

→ For each type of constraint, one constraint handler is responsible.
Default Plugins
Special Linear Constraints

\[
\begin{align*}
\text{min} & \quad x_1 + x_2 + x_3 + x_4 + x_5 + y_1 + y_2 \\
\text{s.t.} & \quad y_1 \leq 3 + 4x_1 \\
& \quad y_2 \leq 4 + 5x_2 \\
& \quad 3x_1 + 2x_2 + 4x_3 \leq 8 \\
& \quad x_2 + x_3 + x_4 = 1 \\
& \quad 3x_1 + 4.5x_2 + 3.2x_5 + 0.8y_1 + 1.2y_2 \geq 4 \\
\end{align*}
\]

\[
x_1, x_2, x_3, x_4 \in \{0, 1\} \\
x_5 \in \mathbb{Z}_+ \\
y_1, y_2 \in \mathbb{R}_+
\]

The 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>
min \ x_1 + x_2 + x_3 + x_4 + x_5 + y_1 + y_2

s.t.

\begin{align*}
    y_1 & \leq 3 + 4 x_1 \\
    y_2 & \leq 4 + 5 x_2 \\
    3 x_1 + 2 x_2 + 4 x_3 & \leq 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 & \geq 4
\end{align*}

x_1, x_2, x_3, x_4 \in \{0, 1\}

x_5 \in \mathbb{Z}_+

y_1, y_2 \in \mathbb{R}_+

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

\[
\begin{align*}
\min & \quad x_1 + x_2 + x_3 + x_4 + x_5 + y_1 + y_2 \\
\text{s.t.} & \quad y_1 \leq 3 + 4 x_1 \\
& \quad y_2 \leq 4 + 5 x_2 \\
& \quad 3 x_1 + 2 x_2 + 4 x_3 \leq 8 \\
& \quad x_2 + x_3 + x_4 = 1 \\
& \quad 3 x_1 + 4.5 x_2 + 3.2 x_5 + 0.8 y_1 + 1.2 y_2 \geq 4 \\
\end{align*}
\]

\[
\begin{align*}
x_1, x_2, x_3, x_4 & \in \{0, 1\} \\
x_5 & \in \mathbb{Z}_+ \\
y_1, y_2 & \in \mathbb{R}_+
\end{align*}
\]

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

\[
\begin{align*}
\text{min} & \quad x_1 + x_2 + x_3 + x_4 + x_5 + y_1 + y_2 \\
\text{s.t.} & \quad y_1 \leq 3 + 4x_1 \\
& \quad y_2 \leq 4 + 5x_2 \\
& \quad 3x_1 + 2x_2 + 4x_3 \leq 8 \\
& \quad x_2 + x_3 + x_4 = 1 \\
& \quad 3x_1 + 4.5x_2 + 3.2x_5 + 0.8y_1 + 1.2y_2 \geq 4 \\
\end{align*}
\]

\[x_1, x_2, x_3, x_4 \in \{0, 1\}\]
\[x_5 \in \mathbb{Z}_+\]
\[y_1, y_2 \in \mathbb{R}_+\]

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

\[
\begin{align*}
\text{min } & \quad x_1 + x_2 + x_3 + x_4 + x_5 + y_1 + y_2 \\
\text{s.t. } & \quad y_1 \leq 3 + 4 x_1 \\
& \quad y_2 \leq 4 + 5 x_2 \\
& \quad 3 x_1 + 2 x_2 + 4 x_3 \leq 8 \\
& \quad x_2 + x_3 + x_4 = 1 \\
& \quad 3 x_1 + 4.5 x_2 + 3.2 x_5 + 0.8 y_1 + 1.2 y_2 \geq 4 \\
\end{align*}
\]

\[
\begin{align*}
x_1, x_2, x_3, x_4 & \in \{0, 1\} \\
x_5 & \in \mathbb{Z}_+ \\
y_1, y_2 & \in \mathbb{R}_+
\end{align*}
\]

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

Feasible region of 0-1 knapsack problem:

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

- 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
- CONSENFLP, CONSENFOPS

▶ **Additional:**
- CONSINIT..., CONSEXIT...
- CONSSEPALP, CONSSEPEASOL
- CONSPROP, CONSRESPROP, CONSPRESOL
- CONSACTIVE, CONSDEACTIVE, CONSENABLE, CONSDISABLE, CONSTRANS, CONSDDELETE, 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

```c
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

```c
struct SCIP_ConshdlrData
{
    int maxrounds;       // max nr. of sepa rounds per node
    int maxsepacuts;     // max nr. of cuts per sepa round
};
```
Creating a single knapsack constraint.

```c
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, ...) );
...
}
```
Including the knapsack constraint handler.

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

    SCIP_CALL( conshdlrdataCreate(scip, &conshdlrdata) );

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

    ...
}
```
## 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
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

\[ 3x_1 - 5x_2 + 2x_3 \leq 7 \]

increasing: \(x_1\) and \(x_3\)

decreasing: \(x_2\)
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:

\[3x_1 + 6x_2 + 4x_3 \leq 8\]
\[2x_1 + 2x_3 \leq 3\]

Result: SCIP_INFEASIBLE
CONSENFOLP and CONSENFOPS

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

**LP solution**

- solution of LP relaxation

**Pseudo Solution**

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

\[
\begin{align*}
\text{min} & \quad x_1 - x_2 + x_3 \\
\text{s.t.} & \quad 3x_1 + 8x_2 + 4x_3 \leq 4 \\
& \quad x_1, x_2, x_3 \in \{0, 1\}
\end{align*}
\]

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

\[
\begin{align*}
\text{min} & \quad x_1 - x_2 + x_3 \\
\text{s.t.} & \quad x_1, x_2, x_3 \in \{0, 1\}
\end{align*}
\]

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

- Presolving
- Node selection
- Conflict analysis
- Processing
- Branching
- Primal heuristics
- Solve LP
- Domain propagation
- Pricing
- Cuts
- Enforce constraints

Enforcement result of constraint handler:
- reduced domain
- added cut
- cutoff
- added constraint
- branched
- infeasible
- feasible
CONSENFOLP and CONSENFOPS

Enforcement result of constraint handler:

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

T. Berthold and K. Wolter (ZIB)
CONSENFOLP and CONSENFOPS

Presolving

Node selection

Conflict analysis

Processing

Primal heuristics

Branching

Solve LP

Domain propagation

Pricing

Cuts

Enforce constraints

Enforcement result of constraint handler:

- reduced domain
- added cut
- cutoff
- infeasible
- added constraint
- branched
- feasible
CONSENFOLP and CONSENFOPS

Enforcement result of constraint handler:

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

T. Berthold and K. Wolter (ZIB)
Enforcement result of constraint handler:

- reduced domain
- added cut
- cutoff
- added constraint
- branched
- infeasible
- feasible
**CONSENFOLP and CONSENFOPS**

**Enforcement result of constraint handler:**

- reduced domain
- added cut
- cutoff
- infeasible
- added constraint
- branched
- feasible
Enforcement result of constraint handler:

- reduced domain
- added cut
- cutoff
- infeasible
- added constraint
- branched
- feasible
ENFORCEMENT RESULT OF CONSTRAINT HANDLER:

- reduced domain
- added cut
- cutoff
- added constraint
- branched
- 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, CONSDSDELETE, CONSFREE
- CONSPRINT, CONSCOPY, CONSPARSE

### Further Ingredients

- Private data
- Interface methods
- Properties/Parameters
CONSINIT...\hspace{1cm} \text{CONSEXIT...}

\begin{itemize}
\item[\textbullet]\linebreak called after problem was transformed / \\
before transformed problem is freed
\item[\textbullet]\linebreak initialize and free statistics in SCIP_ConshdlrData
\end{itemize}
CONSINIT... , CONSEXIT...

CONSINITPRE and CONSEXITPRE:

▶ called before presolving starts / after presolving is finished
▶ initialize and free presolving data
CONSINIT..., CONSEXIT...

Init

Problem

Transforming

Presolving

Init Solve

Transformed

Solving

Free Transform

Transformed

Free Solve

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
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:

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

**Minimal Cover:** \( C \subseteq N \)

- \( \sum_{j \in C} a_j > b \)
- \( \sum_{j \in C \setminus \{i\}} a_j \leq b \quad \forall \; i \in C \)

**Minimal Cover Inequality**

\[ \sum_{j \in C} x_j \leq |C| - 1 \]

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

Minimal cover inequality:
\[ x_2 + x_3 + x_4 \leq 2 \]

Separated in knapsack constraint handler
Separation is implemented in separators and constraint handlers.

1. Separators with $\text{SEPA\_PRIORITY} \geq 0$ (decreasing order)
2. Constraint handlers (decreasing order of $\text{CONSHDLR\_SEPA\_PRIORITY}$)
3. Separators with $\text{SEPA\_PRIORITY} < 0$ (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, CONSEPASOL
  - CONSPROP, CONSRESPROP, CONSPRESOL
  - CONSACTIVE, CONSDEACTIVE, CONSENABLE, CONSDISABLE
  - CONSTRANS, CONSDELETE, CONSFREE
  - CONSPRINT, CONSPARSE, CONSCOPI

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, CONSCOPI

▶ 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, CONSCOOPY

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.
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.

$K_8$
### MIP Formulation

\[
\text{min } \sum_{e \in E} c_e x_e \\
\text{s.t. } \sum_{e \in \delta(v)} x_e = 2 \quad \forall \ v \in V \\
\sum_{e \in \delta(S)} x_e \geq 2 \quad \forall \ S \subseteq V, S \neq \emptyset \\
x_e \in \{0, 1\} \quad \forall \ e \in E.
\]

### CIP Formulation

\[
\text{min } \sum_{e \in E} c_e x_e \\
\text{s.t. } \sum_{e \in \delta(v)} x_e = 2 \quad \forall \ v \in V \\
\text{nsubstour}(G, x) \\
x_e \in \{0, 1\} \quad \forall \ e \in E.
\]

\[
\text{nsubstour}(G, x) \iff \nexists \ C \subseteq \{e \in E | x_e = 1\} : \text{C is a cycle of length } |C| < |V|
\]
Exercise: TSP

- Search Tree
- LP Relaxation
- Presolve Management
- Implication Graph
- Solution Pool
- Cut Pool
- Conflict Analysis

T. Berthold and K. Wolter (ZIB)
Exercise: TSP

- Search Tree
- LP Relaxation
- Presolve Management
- Implication Graph
- Solution Pool
- Cut Pool
- Conflict Analysis
- MIP
  - Default Plugins
Exercise: TSP

- Search Tree
- LP Relaxation
- Presolve Management
  - Implication Graph
  - Solution Pool
  - Cut Pool
  - Conflict Analysis

- MIP Default Plugins
- 2-Opt Heuristic
- TSP File Reader
- New Solution Event Handler
Exercise: TSP

- Search Tree
- LP Relaxation
- Presolve Management
  - Implication Graph
  - Solution Pool
  - Cut Pool
  - Conflict Analysis

- MIP Default Plugins
- 2-Opt Heuristic
- Nosubtour Constraint Hdlr
- LP Greedy Heuristic
- TSP File Reader
- New Solution Event Handler

T. Berthold and K. Wolter (ZIB)
Exercise: TSP

- Search Tree
- LP Relaxation
- Presolve Management
  - Implication Graph
  - Solution Pool
  - Cut Pool
  - Conflict Analysis

- MIP Default Plugins
  - 2-Opt Heuristic

- Nosubtour Constraint Handler
  - LP Greedy Heuristic

- TSP File Reader
  - New Solution Event Handler

T. Berthold and K. Wolter (ZIB)
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

#### Further Ingredients

- Private data
- Interface methods
- Properties
Exercise: TSP

Callback Methods

▷ Fundamental:
  ▶ CONSLOCK
  ▶ CONSCHECK
  ▶ CONSENFOLP, CONSENFOPS

▷ Additional:
  ▶ CONSESEPALP, CONSESEPASOL

▷ Private data
  ▶ Interface methods
  ▶ Properties
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.
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.

### Trivial observation:

- Consider $G = (V, E)$ with edge capacities $x^*_e$.
- $x^*$ violates at least one subtour elimination constraint
  $\iff \exists \text{ cut } \delta(S) \text{ with capacity } x^*(\delta(S)) < 2.$
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 $\geq 2$, all subtour elimination constraints satisfied

$\rightarrow \left(\frac{|V|}{2}\right)$ 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 $\geq 2$, all subtour elimination constraints satisfied

$\rightarrow |V| - 1$ times MaxFlow-MinCut algo within 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 by 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 $\leq 2 - \text{minviol}$
Documentation

- **Constraint Integer Programming**

- **Primal Heuristics for Mixed Integer Programs**

- **Implementation of Cutting Plane Separators for Mixed Integer Programs**

- **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