Implementing Constraint Handlers in SCIP

Gregor Hendel
Ambros Gleixner, Felipe Serrano

September 30
Outline

Why use constraints? Motivation

Callback: Default Constraint Handlers of SCIP

Implementation hints for the Nosubtour Constraint Handler
Why use constraints? Motivation

Callback: Default Constraint Handlers of SCIP

Implementation hints for the Nosubtour Constraint Handler
Traveling Salesman Problem (TSP)

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

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

$K_8$
Traveling Salesman Problem (TSP)

**Definition**
Given a graph \( G = (V, E) \) with edge lengths \( c_e \).

Find a Hamiltonian cycle (cycle containing all nodes, tour) of minimum length.
Motivation For General Constraints:

MIP Formulation

\[
\begin{align*}
\min & \quad \sum_{e \in E} c_e x_e \\
\text{s.t.} & \quad \sum_{e \in \delta(v)} x_e = 2 \quad \forall v \in V \\
& \quad \sum_{e \in \delta(S)} x_e \geq 2 \quad \forall S \subset V, S \neq \emptyset \\
& \quad x_e \in \{0, 1\} \quad \forall e \in E.
\end{align*}
\]
Motivation For General Constraints:

**MIP Formulation**

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

**CIP Formulation**

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

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

Given 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 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
  $\iff \exists$ cut $\delta(S)$ with capacity $x^*(\delta(S)) < 2$. 
Separation Problem

Given 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 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
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 by deleting this edge from $T$
Outline

Why use constraints? Motivation

Callback: Default Constraint Handlers of SCIP

Implementation hints for the Nosubtour Constraint Handler
Default Constraint Handlers

Constraint Handler

- pseudo boolean
- quadratic
- sos1
- sos2
- super indicator
- var bound
- xor
- or
- non linear
- logicer
- linking
- linear
- knapsack
- indicator
- conjunctive
- cumulative
- bivariate
- abs power
- integral
- disjunction
- count
- sos
- bound
disjunction
min \quad x_1 + x_2 + x_3 + x_4 + x_5 + y_1 + y_2

s.t. \quad y_1 \leq 3 + 4 x_1 \\
\quad \quad y_2 \leq 4 + 5 x_2 \\
\quad \quad 3 x_1 + 2 x_2 + 4 x_3 \leq 8 \\
\quad \quad \quad \quad \quad x_2 + x_3 + x_4 = 1 \\
\quad \quad \quad \quad \quad 3 x_1 + 4.5 x_2 + 3.2 x_5 + 0.8 y_1 + 1.2 y_2 \geq 4 \\
\quad \quad x_1, x_2, x_3, x_4 \in \{0, 1\} \\
\quad \quad x_5 \in \mathbb{Z}_+ \\
\quad \quad y_1, y_2 \in \mathbb{R}_+

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

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

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

presolved problem has 7 variables (4 bin, 1 int, 0 impl, 2 cont) and 5 constraints
2 constraints of type <varbound>
1 constraints of type <knapsack>
1 constraints of type <setpcc>
1 constraints of type <linear>
min \quad x_1 + x_2 + x_3 + x_4 + x_5 + y_1 + y_2

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 \\

x_1, x_2, x_3, x_4 \in \{0, 1\} \\
\quad x_5 \in \mathbb{Z}_+ \\
\quad y_1, y_2 \in \mathbb{R}_+$
Special Linear Constraints

\[
\begin{align*}
\text{min} & \quad x_1 + x_2 + x_3 + x_4 + x_5 + y_1 + y_2 \\
\text{s.t.} & \quad y_1 \leq 3 + 4 x_1 \\
& \quad y_2 \leq 4 + 5 x_2 \\
& \quad 3 x_1 + 2 x_2 + 4 x_3 \leq 8 \\
& \quad x_2 + x_3 + x_4 = 1 \\
& \quad 3 x_1 + 4.5 x_2 + 3.2 x_5 + 0.8 y_1 + 1.2 y_2 \geq 4 \\
& \quad x_1, x_2, x_3, x_4 \in \{0, 1\} \\
& \quad x_5 \in \mathbb{Z}^+ \\
& \quad y_1, y_2 \in \mathbb{R}^+ 
\end{align*}
\]

presolved problem has 7 variables (4 bin, 1 int, 0 impl, 2 cont) and 5 constraints

2 constraints of type <varbound>
1 constraints of type <knapsack>
1 constraints of type <setppc>
1 constraints of type <linear>
Special Linear Constraints

\[ \text{min} \quad x_1 + x_2 + x_3 + x_4 + x_5 + y_1 + y_2 \]

s.t.

\[ y_1 \leq 3 + 4 x_1 \]
\[ y_2 \leq 4 + 5 x_2 \]
\[ 3 x_1 + 2 x_2 + 4 x_3 \leq 8 \]
\[ x_2 + x_3 + x_4 = 1 \]
\[ 3 x_1 + 4.5 x_2 + 3.2 x_5 + 0.8 y_1 + 1.2 y_2 \geq 4 \]

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

The presolved problem has 7 variables (4 bin, 1 int, 0 impl, 2 cont) and 5 constraints:
- 2 constraints of type <varbound>
- 1 constraints of type <knapsack>
- 1 constraints of type <setppc>
- 1 constraints of type <linear>
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: \( b \in \mathbb{Z}_+ \)
The integral constraint handler

- ensures the integrality of discrete problem variables in a solution
- does not need constraints because integrality restrictions are encoded in the type of a variable
- is the shortest example of a working constraint handler I know
Ingredients of a Constraint Handler

Callback Methods

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

- **Additional:**
  - CONSINIT..., CONSEXIT...
  - CONSSEEPALP, CONSEEPASOL
  - CONSPROP, CONSRESPROP, CONSRESOL
  - CONSACTIVE, CONSDEACTIVE, CONSENABLE, CONSDISABLE
  - CONSTRANS, CONSDELETE, CONSFREE
  - CONSPRINT, CONSPARSE, CONSCOPI
  - CONSDELVARS, CONSGETVARS, CONSGETNVARS
  - CONSGETDIVEBDCCHGS

Further Ingredients

- Private data
- Interface methods
- Properties/Parameters
Ingredients of a Constraint Handler

Callback Methods

- Fundamental:
  - CONSCHECK
  - CONSENFOLP, CONSENFOPS
  - CONSLOCK

- Additional:
  - CONSINIT..., CONSEXIT...
  - CONSSEPALP, CONSSSEPASOL
  - CONSPROP, CONSRESPROP, CONSPRESOL
  - CONSACTIVE, CONSDEACTIVE, CONSENABLE, CONSDISABLE
  - CONSTRANS, CONDELETE, CONSFREE
  - CONSPRINT, CONSPARSE, CONSCOPY
  - CONSDELVARS, CONSGETVARS, CONSGETNVARS
  - CONSGETDIVEBDCHGS

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
}
```
Creating a single knapsack constraint.

```c
SCIP_RETCODE SCIPcreateConsBasicKnapsack(
    SCIP* scip, /**< SCIP data structure */
    SCIP_CONS** cons, /**< pointer to hold the created constraint */
    const char* name, /**< name of constraint */
    int nvars, /**< number of items in the knapsack */
    SCIP_VAR** vars, /**< array with item variables */
    SCIP_Longint* weights, /**< array with item weights */
    SCIP_Longint capacity /**< capacity of knapsack */
)
{
    SCIP_CONSDATA* consdata;
    SCIP_CALL( consdataCreate(scip, &consdata, nvars, vars, weights, capacity) );
    SCIP_CALL( SCIPcreateConsBasic(scip, cons, name, conshdlr, consdata) );
    return SCIP_OKAY;
}
```
Including the knapsack constraint handler.

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

    SCIP_CALL( conshdlrdataCreate(scip, &conshdlrdata) );

    /* include constraint handler into SCIP */
    SCIP_CALL( SCIPincludeConshdlrBasic(scip, &conshdlr,
        CONSHDLR_NAME, CONSHDLR_DESC,
        [...] 
        consEnfolpKnapsack, consEnfopsKnapsack,
        consCheckKnapsack, consLockKnapsack,
        conshdlrdata) );

    /* set non–fundamental callbacks via specific setter functions */
    ...
}
```
Ingredients of a Constraint Handler

Callback Methods

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

- **Additional:**
  - CONSINIT..., CONSEXIT...
  - CONSESEPALP, CONSESEPASOL
  - CONSPROP, CONSRESPROP, CONSPRESOL
  - CONSACTIVE, CONSDEACTIVE, CONSENABLE, CONSDISABLE
  - CONSTRANS, CONSDELETE, CONSFREE
  - CONSPRINT, CONSPARSE, CONSCOPI
  - CONSDELVARS, CONSGETVARS, CONSGETNVARS
  - CONSGETDIVEBDCHGS

Further Ingredients

- Private data
- Interface methods
- Properties/Parameters
Callback Methods

What is a callback method?
Callback Methods

What is a **callback** method?

Nice quote from someone who cited an older Wikipedia version:

*In computer programming, a callback is a reference to executable code, or a piece of executable code, that is passed as an argument to other code. This allows a lower-level software layer to call a subroutine (or function) defined in a higher-level layer.*

source:

http://stackoverflow.com/questions/824234/what-is-a-callback-function
Callback Methods

What is a callback method?

Nice quote from someone who cited an older Wikipedia version:

In computer programming, a callback is a reference to executable code, or a piece of executable code, that is passed as an argument to other code. This allows a lower-level software layer to call a subroutine (or function) defined in a higher-level layer.

source:
http://stackoverflow.com/questions/824234/what-is-a-callback-function

All the functionality of constraint handlers (as well as all other plugin types of SCIP) is defined through callback functions.
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

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

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

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

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

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

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
Enforcement result of constraint handler:

- reduced domain
- added cut
- added constraint
- 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

Processing
- Presolving
- Node selection
- Conflict analysis
- Processing
- Branching
- Primal heuristics

Solve LP
- Domain propagation
- Solve LP
- Pricing
- Cuts
- Enforce constraints

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

- reduced domain
- added constraint
- added cut
- branched
- cutoff
- infeasible
- feasible
The integral constraint handler

- ensures the integrality of discrete problem variables in a solution
- does not need constraints because integrality restrictions are encoded in the type of a variable
- is the shortest example of a working constraint handler I know
The integral constraint handler

▶ ensures the integrality of discrete problem variables in a solution
▶ does not need constraints because integrality restrictions are encoded in the type of a variable
▶ is the shortest example of a working constraint handler I know

**Question**

How does the integral constraint handler enforce the LP solution?
The integral constraint handler

- ensures the integrality of discrete problem variables in a solution
- does not need constraints because integrality restrictions are encoded in the type of a variable
- is the shortest example of a working constraint handler I know

**Question**
How does the integral constraint handler enforce the LP solution?

**Answer**
It calls the branching rules to perform variable branching.
**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

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

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

**decreasing**: \(x_2\)
Ingredients of a Constraint Handler

Callback Methods

- **Fundamental:**
  - CONS_CHECK
  - CONSENSFOLP, CONSENSFOPS
  - CONSLOCK

- **Additional:**
  - CONSINIT..., CONSEXIT...
  - CONSEPALP, CONSEPASOL
  - CONSPROP, CONSRESPROP, CONSPRESOL
  - CONSACTIVE, CONSDAUTHOR, CONSENABLE, CONSDISABLE
  - CONSTRANS, CONSDELETE, CONSFREE
  - CONSPRINT, CONSPARSE, CONSCOPY
  - CONSDELVARS, CONSGETVARS, CONSGETNVARS
  - CONSGETDIVEBDCHGS

Further Ingredients

- Private data
- Interface methods
- Properties/Parameters
CONSINIT... , CONSEXIT...

CONSINIT and CONSEXIT:
- called after problem was transformed / before transformed problem is freed
- initialize and free statistics in SCIP_ConshdlrData
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
Ingredients of a Constraint Handler

Callback Methods

▶ Fundamental:
  ▶ CONSCHECK
  ▶ CONSENFOLP, CONSENFOPS
  ▶ CONSLOCK

▶ Additional:
  ▶ CONSINIT..., CONSEXIT...
  ▶ CONSSEPALP, CONSSEPASOL
  ▶ CONSPROP, CONSRESPROP, CONSRESOL
  ▶ CONSACTIVE, CONSDEACTIVE, CONSENABLE, CONSDISABLE
  ▶ CONSTRANS, CONSDELETE, CONSFREE
  ▶ CONSPRINT, CONSPARSE, CONSCOPY
  ▶ CONSDELVARS, CONSGETVARS, CONSGETNVARS
  ▶ CONSGETDIVEBDCHGS

Further Ingredients

▶ Private data
▶ Interface methods
▶ Properties/Parameters
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**

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

5\(x_1 + 6x_2 + 2x_3 + 2x_4 \leq 8 \)

**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 \text{ (decreasing order)}
2. Constraint handlers \text{ (decreasing order of CONSHDLR\_SEPA\_PRIORITY)}
3. Separators with \text{SEPA\_PRIORITY} < 0 \text{ (decreasing order)}
Ingredients of a Constraint Handler

Callback Methods

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

- **Additional:**
  - CONSINIT..., CONSEXIT...
  - CONSSESEPALP, CONSSEPASOL
  - CONSPROP, CONSRESPROP, CONSPRESOL
  - CONSACTIVE, CONSDEACTIVE, CONSENABLE, CONSDISABLE
  - CONSTRANS, CONSDELETE, CONSFREE
  - CONSPRINT, CONSPARSE, CONSCOPY
  - CONSDELVARS, CONSGETVARS, CONSGETNVARS
  - CONSGETDIVEBDCHGS

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

- **Additional:**
  - CONSINIT..., CONSEXIT...
  - CONSSSEPALP, CONSSSEPASOL
  - CONSPROP, CONSRESPROP, CONSPRESOL
  - CONSACTIVE, CONSDEACTIVE, CONSENABLE, CONSDISABLE
  - CONSTRANS, CONSDELETE, CONSFREE
  - CONSPRINT, CONSPARSE, CONSCOPY
  - CONSDELVARS, CONSGETVARS, CONSGETNVARS
  - CONSGETDIVEBDCHGS

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:**
  - CONSCHECK
  - CONSENFOLP, CONSENFOPS
  - CONSLOCK

- **Additional:**
  - CONSINIT..., CONSEXIT...
  - CONSSSEPALS, CONSEPSASOL
  - CONSPROP, CONSRESPROP, CONSPRESOL
  - CONSACTIVE, CONSDEACTIVE, CONSENABLE, CONSDISABLE
  - CONSTRANS, CONSDELETE, CONSFREE
  - CONSPRINT, CONSPARSE, CONSCOPY
  - CONSDELVARS, CONSGETVARS, CONSGETNVARS
  - CONSGETDIVEBDCHGS

- **Displaying, parsing problems**
- **Copying problems between different SCIP environments**
Outline

Why use constraints? Motivation

Callback: Default Constraint Handlers of SCIP

Implementation hints for the Nosubtour Constraint Handler
Exercise: TSP

**MIP Formulation**

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

**CIP Formulation**

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

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

- New Solution Event Handler
- TSP File Reader
- 2-Opt Heuristic
- MIP Default Plugins
- Conflict Analysis
- Cut Pool
- Solution Pool
- Implication Graph
- Presolve Management
- LP Relaxation
- Search Tree

Hendel, Gleixner, Serrano – Constraint Handler Implementation
Exercise: TSP

- New Solution Event Handler
- TSP File Reader
- Nosubtour Constraint Hdlr
- MIP Default Plugins
- 2-Opt Heuristic
- Conflict Analysis
- Cut Pool
- Solution Pool
- Implication Graph
- Presolve Management
- LP Relaxation
- Search Tree
Exercise: TSP

- New Solution Event Handler
- TSP File Reader
- Bonus: LP Greedy Heuristic
- Nosubtour Constraint Hdlr
- 2-Opt Heuristic
- MIP Default Plugins
- Conflict Analysis
- Cut Pool
- Solution Pool
- Implication Graph
- Presolve Management
- LP Relaxation
- Search Tree
Exercise: TSP

Callback Methods

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

- **Additional:**
  - CONSINIT..., CONSEXIT...
  - CONSEPAPALP, CONSEPASOL
  - CONSPROP, CONSRESPROP, CONSPRESOL
  - CONSACTIVE, CONSDEACTIVE, CONSENABLE, CONSDISABLE
  - CONSTRANS, CONSDELETE, CONSFREE
  - CONSPRINT, CONSPARSE, CONSCOPY
  - CONSDELVARS, CONSGETVARS, CONSGETNVARS
  - CONSGETDIVEBDCHGS

Further Ingredients

- Private data
- Interface methods
- Properties
Exercise: TSP

Callback Methods

- Fundamental:
  - CONSCHECK
  - CONSENFOLP, CONSENFOPS
  - CONSLOCK

- Additional:
  - CONSSEPALP, CONSSEPASOL

- CONSTRANS, CONSDELETE

Further Ingredients

- Private data
- Interface methods
- Properties
Feasibility Check

Given a point $y \in \mathbb{Q}^{|E|}$, decide whether $y$ satisfies all subtour elimination constraints.
Feasibility Check

Given a point \( y \in \mathbb{Q}^{|E|} \), decide whether \( y \) satisfies all subtour elimination constraints.

Check Priorities make this task easier:

- Constraint handlers check \( y \) in decreasing order of their check priority (CONSHDLR_CHECKPRIORITY).
- Check priority of \( \text{nosubtour}(G, x) \) smaller than linear and integral check priorities.
- The solution \( y \) already satisfies integrality and (linear) degree constraints.

\[
\begin{align*}
\text{min} & \quad \sum_{e \in E} c_e x_e \\
\text{s.t.} & \quad \sum_{e \in \delta(v)} x_e = 2 \quad \forall v \in V \\
\text{nosubtour}(G, x) & \\
& \quad x_e \in \{0, 1\} \quad \forall e \in E.
\end{align*}
\]
Feasibility Check

Given a point \( y \in \mathbb{Q}^{|E|} \), decide whether \( y \) satisfies all subtour elimination constraints.

Check Priorities make this task easier:

- Constraint handlers check \( y \) in decreasing order of their check priority (CONSHDLR_CHECKPRIORITY).
- Check priority of \( \text{nosubtour}(G, x) \) smaller than linear and integral check priorities
- The solution \( y \) already satisfies integrality and (linear) degree constraints.
Feasibility Check

Given a point $y \in \mathbb{Q}^{|E|}$, decide whether $y$ satisfies all subtour elimination constraints.

Check Priorities make this task easier:

- Constraint handlers check $y$ in decreasing order of their check priority (CONSHDLR_CHECKPRIORITY).
- Check priority of $\text{nosubtour}(G, x)$ smaller than linear and integral check priorities.
- The solution $y$ already satisfies integrality and (linear) degree constraints.

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

$\Rightarrow$ $y$ is the characteristic vector of a collection of cycles when passed to the $\text{nosubtour}$ constraint handler.
SCIP callback declarations are realized as macros:

What you see in `cons_nosubtour.c`:

```c
static
SCIP_DECL_CONSCHECK(consCheckNosubtour)
{
    // Your Sophisticated Code
}
```

What the compiler sees (after preprocessor and some formatting):

```c
static
SCIP_RETCODE consCheckNosubtour(SCIP* scip, SCIP_CONSHDLR* conshdlr,
    SCIP_CONS** conss, int nconss, SCIP_SOL* sol,
    SCIP_Bool checkintegrality, SCIP_Bool checklprows,
    SCIP_Bool printreason, SCIP_RESULT* result)
{
    // Your Sophisticated Code
}
```

`SCIP_DECL_CONSCHECK` is a macro defined in `type_cons.h`. It is responsible of expanding the callback function with the needed arguments and return value.
A compilable check callback

```c
static
SCIP_DECL_CONSCHECK(consCheckNosubtour)
{
    /* TODO: add some more code */
    *result = SCIP_INFEASIBLE;
    return SCIP_OKAY;
}
```

- all callbacks must return a SCIP_RETCODE! Use SCIP_OKAY if everything worked
- often, callbacks require to correctly assign a result-pointer, examples:
  - *result = SCIP_[IN]FEASIBLE
  - *result = SCIP_SEPARATED
  - *result = SCIP_DIDNOTFIND

Find the full callback definition and possible return types in type_<pluginname>.h, eg, type_cons.h for constraint handlers or type_heur.h for primal heuristics.
The graph data structure for this exercise is defined in GomoryHuTree.h:

typedef struct Graph
{
    int nuses;
    int nnodes; /\*\*< number of nodes of the graph */
    int nedges; /\*\*< number of edges */
    int nedgesnonzero;
    GRAPHNODE *nodes; /\*\*< array containing the nodes of the graph */
    GRAPHEDGE *edges; /\*\*< array containing all halfedges ... */
} GRAPH;
The Graph implementation

Every graph edge is a half-edge in GomoryHuTree.h:

```c
typedef struct GraphEdge {
    double cap;  /*< capacity used in maxflow */
    double rcap; /*< residual capacity used in maxflow */
    double length; /*< length of the edge measured by some fixed metric */
    struct GraphEdge *next; /*< in incidence list of node from which edge is emanating */
    struct GraphEdge *back; /*< pointer to reverse edge */
    GRAPHNODE *adjac; /*< pointer to adjacent node */
    SCIP_VAR* var;
} GRAPHEDGE;
```

Every edge has a variable pointer `edge->var`. 
The Graph implementation

Nodes of the graph only have two relevant members (for us):

typedef struct GraphNode
{
    int id;  /*< number of the node*/
    ...
    struct GraphEdge *first_edge;  /*< in list of incident edges */
    ...
} GRAPHNODE;
The Graph implementation

Question
How to traverse all incident edges of a node?

Answer:

```c
GRAPHEDGE* edge = node->first_edge;
while( edge != NULL )
{
    /* do something with this edge */
    edge = edge->next;
}
```
Only one method from GomoryHuTree.h is relevant for us: ghc_tree(). It has the following arguments:

- GRAPH* gr: the graph data structure
- SCIP_Bool** cuts: 2-dimensional array to return cuts, needs to be allocated by the user
- int* ncuts: pointer to store the number of violated cuts
- double minviol feasibility tolerance, use SCIPfeastol(scip)

The method returns TRUE if it found at least one violated cut, and false otherwise.

ghc_tree() fills cuts with all minimum \((s, t)\)-cuts with capacity \(\leq 2 - \text{minviol}\)
Documentation

- **Constraint Integer Programming**

- [http://scip.zib.de](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.