Introduction to Constraint Integer Programming

Ambros M. Gleixner

Zuse Institute Berlin · MATHEON · Berlin Mathematical School

5th Porto Meeting on Mathematics for Industry, April 10–11, 2014, Porto
ZIB – Fast Algorithms, Fast Computers

Zuse Institute Berlin is a research institute and computing center of the State of Berlin with research units:

▶ Numerical Analysis and Modeling
▶ Visualization and Data Analysis
▶ Scientific Information Systems
▶ Computer Science and High Performance Computing

President: Prof. Dr. Dr. h.c. mult. Martin Grötschel
Zuse Institute Berlin is a research institute and computing center of the State of Berlin with research units:

- Numerical Analysis and Modeling
- Visualization and Data Analysis
- Scientific Information Systems
- Computer Science and High Performance Computing

President: Prof. Dr. Dr. h.c. mult. Martin Grötschel
Outline

Constraint Integer Programming

MIP + CP = CIP

Solving Constraint Integer Programs

The SCIP Optimization Suite

Using SCIP
Outline

**Constraint Integer Programming**

\[
\text{MIP} + \text{CP} = \text{CIP}
\]

Solving Constraint Integer Programs

The SCIP Optimization Suite

Using SCIP
Definition (TSP)

Given a complete graph $G = (V, E)$ and distances $d_e$ for all $e \in E$:

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

$K_8$
Definition (TSP)

Given a complete graph $G = (V, E)$ and distances $d_e$ for all $e \in E$:

Find a Hamiltonian cycle (cycle containing all nodes, tour) of minimum length.
An example: the Traveling Salesman Problem

Definition (TSP)

Given a complete graph $G = (V, E)$ and distances $d_e$ for all $e \in E$:

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

$\frac{(n-1)!}{2}$ possible solutions: finite, but enumeration intractable!
TSP – Integer Programming Formulation

Given

- complete graph $G = (V, E)$
- distances $d_e > 0$ for all $e \in E$

Binary variables

- $x_e = 1$ if edge $e$ is used
TSP – Integer Programming Formulation

Given

- complete graph \( G = (V, E) \)
- distances \( d_e > 0 \) for all \( e \in E \)

Binary variables

- \( x_e = 1 \) if edge \( e \) is used

\[
\begin{align*}
\text{min} & \quad \sum_{e \in E} d_e x_e \\
\text{subject to} & \quad \sum_{e \in \delta(v)} x_e = 2 \quad \forall v \in V \\
& \quad \sum_{e \in \delta(S)} x_e \geq 2 \quad \forall S \subset V, S \neq \emptyset \\
& \quad x_e \in \{0, 1\} \quad \forall e \in E
\end{align*}
\]
TSP – Integer Programming Formulation

Given

- complete graph \( G = (V, E) \)
- distances \( d_e > 0 \) for all \( e \in E \)

Binary variables

- \( x_e = 1 \) if edge \( e \) is used

\[
\min \sum_{e \in E} d_e x_e
\]

subject to

\[
\sum_{e \in \delta(v)} x_e = 2 \quad \forall v \in V \quad \text{node degree}
\]

\[
\sum_{e \in \delta(S)} x_e \geq 2 \quad \forall S \subset V, S \neq \emptyset
\]

\[
x_e \in \{0, 1\} \quad \forall e \in E
\]
TSP – Integer Programming Formulation

Given
- complete graph $G = (V, E)$
- distances $d_e > 0$ for all $e \in E$

Binary variables
- $x_e = 1$ if edge $e$ is used

$$\begin{align*}
\text{min} & \quad \sum_{e \in E} d_e x_e \\
\text{subject to} & \quad \sum_{e \in \delta(v)} x_e = 2 \quad \forall v \in V \\
& \quad \sum_{e \in \delta(S)} x_e \geq 2 \quad \forall S \subset V, S \neq \emptyset \\
& \quad x_e \in \{0, 1\} \quad \forall e \in E
\end{align*}$$

subtour elimination
TSP – Integer Programming Formulation

Given
- complete graph $G = (V, E)$
- distances $d_e > 0$ for all $e \in E$

Binary variables
- $x_e = 1$ if edge $e$ is used

Objective
\[
\min \sum_{e \in E} d_e x_e
\]

Subject to
\[
\sum_{e \in \delta(v)} x_e = 2 \quad \forall v \in V
\]
\[
\sum_{e \in \delta(S)} x_e \geq 2 \quad \forall S \subset V, S \neq \emptyset
\]
\[
x_e \in \{0, 1\} \quad \forall e \in E
\]
TSP – Integer Programming Formulation

Given

- complete graph \( G = (V, E) \)
- distances \( d_e > 0 \) for all \( e \in E \)

Binary variables

- \( x_e = 1 \) if edge \( e \) is used

\[
\begin{align*}
\text{min} & \quad \sum_{e \in E} d_e x_e \\
\text{subject to} & \quad \sum_{e \in \delta(v)} x_e = 2 & \forall v \in V \\
& \quad \sum_{e \in \delta(S)} x_e \geq 2 & \forall S \subset V, S \neq \emptyset \\
& \quad x_e \in \{0, 1\} & \forall e \in E
\end{align*}
\]
**TSP – Integer Programming Formulation**

**LP Relaxation**
- convex feasible region
- efficiently solvable
- dual bound on the IP optimum

\[
\begin{align*}
\min & \quad \sum_{e \in E} d_e x_e \\
\text{subject to} & \quad \sum_{e \in \delta(v)} x_e = 2 & \forall v \in V \\
& \quad \sum_{e \in \delta(S)} x_e \geq 2 & \forall S \subset V, S \neq \emptyset \\
& \quad x_e \in [0, 1] & \forall e \in E
\end{align*}
\]
Linear Programming Algorithms

Simplex algorithm  Ellipsoid method  Interior point
1947  Dantzig: Primal Simplex algorithm
1954  Lemke and Beale: Dual Simplex algorithm
       ▶ by far the most used algorithm to solve LPs
       ▶ worst case exponential running time
       ▶ fast reoptimization ↔ does not parallelize well

1979  Khachiyan: Ellipsoid Method
       ▶ first polynomial time algorithm
       ▶ not suited for practical application

1984  Karmarkar: Interior Point Method/Barrier Algorithm
1989  Kojima et al.: Primal-dual Interior Point Algorithm
       ▶ first practical polynomial algorithms
       ▶ for single LPs often faster then simplex
       ▶ good parallel speedup ↔ unsuited for reoptimization
1947  Dantzig: Primal Simplex algorithm
1954  Lemke and Beale: Dual Simplex algorithm
   ▶ by far the most used algorithm to solve LPs
   ▶ worst case exponential running time
   ▶ fast reoptimization ↔ does not parallelize well

1979  Khachiyan: Ellipsoid Method
   ▶ first polynomial time algorithm
   ▶ not suited for practical application

1984  Karmarkar: Interior Point Method/Barrier Algorithm
1989  Kojima et al.: Primal-dual Interior Point Algorithm
   ▶ first practical polynomial algorithms
   ▶ for single LPs often faster then simplex
   ▶ good parallel speedup ↔ unsuited for reoptimization
Cutting Plane Separation

**Task**
- strengthen relaxation
- add valid constraints
- generate on demand

**Techniques**
- **general cuts**
  - complemented MIR cuts
  - Gomory mixed integer cuts
  - strong Chvátal-Gomory cuts
  - implied bound cuts
  - reduced cost strengthening
- **problem specific cuts**
  - 0-1 knapsack problem
  - stable set problem
  - 0-1 single node flow problem
Branch-and-Bound (Land and Doig 1960)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching
Branch-and-Bound (Land and Doig 1960)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching
Branch-and-Bound (Land and Doig 1960)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching
Branch-and-Bound (Land and Doig 1960)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching
Branch-and-Bound (Land and Doig 1960)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching
Branch-and-Bound (Land and Doig 1960)

Steps

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

Diagram showing the process of branch-and-bound with nodes and constraints.
Branch-and-Bound (Land and Doig 1960)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching
Branch-and-Bound (Land and Doig 1960)

Steps

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


Branch-and-Bound (Land and Doig 1960)

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
Branch-and-Bound (Land and Doig 1960)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching
Branch-and-Bound (Land and Doig 1960)

Steps

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

Diagram: A tree structure with nodes and branches, representing the decision process of the Branch-and-Bound algorithm.
Branch-and-Bound (Land and Doig 1960)

Steps

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

\[ x^{IP} \]
Branch-and-Bound (Land and Doig 1960)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching
Branch-and-Bound (Land and Doig 1960)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching
Branch-and-Bound (Land and Doig 1960)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching
Branch-and-Bound (Land and Doig 1960)

Steps

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

\[ \begin{align*}
   \text{Abort criterion} & \quad \text{Node selection} \\
   \text{Solve relaxation} & \quad \text{Bounding} \\
   \text{Feasibility check} & \quad \text{Branching}
\end{align*} \]
Branch-and-Bound (Land and Doig 1960)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching
Branch-and-Bound (Land and Doig 1960)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching
Branch-and-Bound (Land and Doig 1960)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching
Branch-and-Bound (Land and Doig 1960)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching
Branch-and-Bound (Land and Doig 1960)

Steps

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

\[ \emptyset \]

\[ \infty \]
Branch-and-Bound (Land and Doig 1960)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching
Branch-and-Bound (Land and Doig 1960)

Steps

1. Abort criterion
2. Node selection
3. Solve relaxation
4. Bounding
5. Feasibility check
6. Branching
Branch-and-Bound (Land and Doig 1960)

**Steps**

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

![Branch-and-Bound Tree Diagram]
Branching Rules

Task
- divide into (disjoint) subproblems
- improve local bounds

Techniques
- branching on variables
  - most infeasible
  - least infeasible
  - random branching
  - strong branching
  - pseudocost
  - reliability
  - VSIDS
  - hybrid reliability/inference
- branching on constraints
  - SOS1
  - SOS2
Example: Pseudocost Branching

Estimating the objective

\[ \zeta - (x_3) = 4 - 2 = 2 \]

Pseudo costs:

Average objective gain

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

\[ n = 5 + 3 = 8 \]

Estimate increase of objective by pseudo costs and fractionality:

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

\[ \psi + (x_3)(1 - \text{frac}(x_3)) = 7.6 \]

\[ x_3 = 7.4 \]

\[ c = 2 \]
Example: Pseudocost Branching

Estimating the objective

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

\[
x_3 = 7.4 \\
c = 2 \\
x_3 \leq 7 \\
c = 4
\]
Example: Pseudocost Branching

Estimating the objective

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

Estimating the objective

- objective gain per unit:
  - $\zeta^{-}(x_3) = 5$, $\zeta^{+}(x_3) = 10$
Example: Pseudocost Branching

Estimating the objective

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

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

Diagram:
- $\zeta_1^-(x_3) = 5$
- $\zeta_2^-(x_3) = 3$
Example: Pseudocost Branching

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:
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:
    - $\psi^-(x_3) \cdot \text{frac}(x_3)$
Example: Pseudocost Branching

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:
    - $\psi^-(x_3) \cdot \text{frac}(x_3) = 4 \cdot 0.2 = 0.8$,
Example: Pseudocost Branching

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:
  $\psi^-(x_3) \cdot \text{frac}(x_3) = 4 \cdot 0.2 = 0.8$, and $\psi^+(x_3)(1 - \text{frac}(x_3)) = 7.6$
Primal Heuristics

**Task**
- improve primal bound
- effective on average
- guide remaining search

**Techniques**
- rounding
  - possibly solve final LP
- diving
  - least infeasible
  - guided
- objective diving
  - objective feasibility pump
- Large Neighborhood Search
  - RINS, local branching
  - RENS
- combinatorial
Example: The Feasibility Pump

Algorithm

1. Solve LP;
2. Round LP optimum;
3. If feasible:
   4. Stop!
5. Else:
   6. Change objective;
7. Go to 1;
Example: The Feasibility Pump

**Algorithm**

1. Solve LP;
2. Round LP optimum;
3. If feasible:
   4. Stop!
5. Else:
6. Change objective;
7. Go to 1;
Algorithm

1. Solve LP;
2. Round LP optimum;
3. If feasible:
   4. Stop!
5. Else:
6. Change objective;
7. Go to 1;
Algorithm

1. Solve LP;
2. Round LP optimum;
3. If feasible:
   4. Stop!
5. Else:
6. Change objective;
7. Go to 1;
Example: The Feasibility Pump

Algorithm

1. Solve LP;
2. Round LP optimum;
3. If feasible:
   4. Stop!
4. Else:
   5. Change objective;
7. Go to 1;
\[ \Delta(x, \tilde{x}) = \sum |x_j - \tilde{x}_j| \]
Example: The Feasibility Pump

Algorithm

1. Solve LP;
2. Round LP optimum;
3. If feasible:
4. Stop!
5. Else:
6. Change objective;
7. Go to 1;

$\Delta(x, \tilde{x}) = \sum |x_j - \tilde{x}_j|$
Algorithm

1. Solve LP;
2. Round LP optimum;
3. If feasible:
4. Stop!
5. Else:
6. Change objective;
7. Go to 1;

\[ \Delta(x, \tilde{x}) = \sum |x_j - \tilde{x}_j| \]
Algorithm

1. Solve LP;
2. Round LP optimum;
3. If feasible:
   4. Stop!
5. Else:
6. Change objective;
7. Go to 1;
Example: The Feasibility Pump

Algorithm

1. Solve LP;
2. Round LP optimum;
3. If feasible:
   4. Stop!
5. Else:
   6. Change objective;
7. Go to 1;
Example: The Feasibility Pump

Algorithm

1. Solve LP;
2. Round LP optimum;
3. If feasible:
   4. Stop!
5. Else:
6. Change objective;
7. Go to 1;

\[ \Delta(x, \tilde{x}) = \sum |x_j - \tilde{x}_j| \]
The largest solved instance of the traveling salesman problem consists of a tour through **85,900 cities** in a VLSI application that arose in Bell Laboratories in the late 1980s. The total amount of computer usage for the computations was appx. **136 CPU years**.
## MIP vs. CP

### Mixed Integer Program

**Objective function:**
- linear function

**Feasible set:**
- described by linear constraints

**Variable domains:**
- real or integer values

**General form:**
\[
\begin{align*}
\min & \quad c^T x \\
\text{s.t.} & \quad Ax \leq b \\
& (x_I, x_C) \in \mathbb{Z}^I \times \mathbb{R}^C
\end{align*}
\]

### Constraint Program

**Objective function:**
- arbitrary function

**Feasible set:**
- given by arbitrary constraints

**Variable domains:**
- arbitrary (usually finite)

**General form:**
\[
\begin{align*}
\min & \quad c(x) \\
\text{s.t.} & \quad C_k(x) \text{ for } k = 1, \ldots, m \\
& (x_I, x_N) \in \mathbb{Z}^I \times X
\end{align*}
\]
Given

- complete graph \( G = (V, E) \)
- for each \( e \in E \) a distance 
  \( d_e > 0 \)

Integer variables

- \( x_v \) position of \( v \in V \) in tour
Given

- complete graph $G = (V, E)$
- for each $e \in E$ a distance $d_e > 0$

Integer variables

- $x_v$ position of $v \in V$ in tour

\[
\min \quad \text{length}(x_1, \ldots, x_n) \\
\text{subject to} \quad \text{alldifferent}(x_1, \ldots, x_n) \\
x_v \in \{1, \ldots, n\} \quad \forall v \in V
\]
Constraint Integer Programming

- **Mixed Integer Programs**

Relation to CP and MIP

- Every MIP is a CIP. "\( \text{MIP} \subsetneq \text{CIP} \)"
- Every CP over a finite domain space is a CIP. "\( \text{FD} \subsetneq \text{CIP} \)"
Constraint Integer Programming

- **Mixed Integer Programs**
- **SATisfiability problems**

### Relation to CP and MIP

- Every MIP is a CIP. "\( MIP \subsetneq CIP \)"
- Every CP over a finite domain space is a CIP. "\( FD \subsetneq CIP \)"
Constraint Integer Programming

- Mixed Integer Programs
- SAT satisfiability problems
- Pseudo-Boolean Optimization

Relation to CP and MIP

- Every MIP is a CIP. “\( \text{MIP} \subsetneq \text{CIP} \)”
- Every CP over a finite domain space is a CIP. “\( \text{FD} \subsetneq \text{CIP} \)”
Constraint Integer Programming

- Mixed Integer Programs
- SAT isfiability problems
- Pseudo-Boolean Optimization
- Finite Domain

Relation to CP and MIP

- Every MIP is a CIP. “MIP ⊆ CIP”
- Every CP over a finite domain space is a CIP. “FD ⊆ CIP”
Constraint Integer Programming

- **Mixed Integer Programs**
- **SATisfiability problems**
- **Pseudo-Boolean Optimization**
- **Finite Domain**
- **Constraint Programming**

Relation to CP and MIP

- Every MIP is a CIP. “$\text{MIP} \subset \text{CIP}$”
- Every CP over a finite domain space is a CIP. “$\text{FD} \subset \text{CIP}$”
Constraint Integer Programming

- Mixed Integer Programs
- SATisfiability problems
- Pseudo-Boolean Optimization
- Finite Domain
- Constraint Programming
- Constraint Integer Programming

Relation to CP and MIP

- Every MIP is a CIP. “\( MIP \subsetneq CIP \)”
- Every CP over a finite domain space is a CIP. “\( FD \subsetneq CIP \)”
What is a Constraint Integer Program?

**Constraint Integer Program**

Objective function:
- **linear function**

Feasible set:
- described by *arbitrary* constraints

Variable domains:
- real or integer values

After fixing all integer variables:
- CIP becomes tractable (e.g. LP)

General form:
\[
\begin{align*}
\text{min} & \quad c^T x \\
\text{s.t.} & \quad C_k(x) \text{ for } k = 1, \ldots, m \\
& \quad (x_I, x_C) \in \mathbb{Z}^I \times \mathbb{R}^C
\end{align*}
\]

Remark:
- arbitrary objective or variables modeled by constraints
What is a Constraint Integer Program?

**Constraint Integer Program**

Objective function:
- linear function

Feasible set:
- described by arbitrary constraints

Variable domains:
- real or integer values

After fixing all integer variables:
- CIP becomes tractable (e.g. LP)

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

(CIP formulation of TSP)

Single nosubtour constraint rules out subtours (e.g. by domain propagation). It may also separate subtour elimination inequalities.
Constraint Integer Programming

MIP + CP = CIP

Solving Constraint Integer Programs

The SCIP Optimization Suite

Using SCIP
How do we solve CIPs?

MIP
- LP relaxation
- cutting planes

CP
- domain propagation

SAT
- conflict analysis
- periodic restarts

MIP, CP, and SAT
- branch-and-bound

SCIP
Conflict Analysis

Task

▶ Analyze infeasibility
▶ Derive valid constraints
▶ Help to prune other nodes

Techniques

▶ Analyze
  ▶ infeasible LPs
  ▶ bound-exceeding LPs
  ▶ propagation conflicts

▶ Reduce conflict by
  ▶ LP dual ray heuristic
  ▶ cut in conflict graph

▶ Apply conflicts
  ▶ for propagation
  ▶ as cutting planes
Domain Propagation

Task
- simplify model locally
- improve local dual bound
- detect infeasibility

Techniques
- Constraint-specific
  - each constraint handler may provide a propagation routine
  - reduced presolving (usually)
- Dual propagation
  - reduced cost strengthening
  - objective function
- Special global structures
  - variable bounds
  - cliques
Example: AND Constraints

AND

- \( y = \prod_{i \in J} x_i \), \( y, x_i \in \{0, 1\} \ \forall i \in J \)
- \( y \): resultant, \( x_i \): operand variables
- can be linearized
- linearization weak/big

Propagation rules

- \( \exists i: x_i = 0 \Rightarrow y = 0 \)
- \( x_i = 1 \ \forall i \in J \Rightarrow y = 1 \)
- \( y = 1 \Rightarrow x_i = 1 \ \forall i \in J \)
- \( y = 0 \ \wedge \exists k: x_i = 1 \ \forall i \in J \setminus \{k\} \Rightarrow x_k = 0 \)
Linearization of $y = x_1 \land \cdots \land x_n$

\[
\sum_{i=1}^{n} x_i - y \leq n - 1
\]
\[
\sum_{i=1}^{n} x_i - n y \geq 0
\]

- 2 constraints
  - contains fractional vertices 🔴

- n + 1 constraints
  - only integer vertices ⬤
Enforcement of AND Constraints

Only propagation

- **good**: fast subproblem processing
- **bad**: LP relaxation has no knowledge about the nonlinear structure

SCIP’s strategy

- propagation + dynamically decide between relaxation and separation
Enforcement of AND Constraints

Only propagation
- **good**: fast subproblem processing
- **bad**: LP relaxation has no knowledge about the nonlinear structure

Only relaxation – put the complete linearization into the LP
- **good**: LP relaxation contains complete problem
- **bad**: can blow up the LP
Enforcement of AND Constraints

Only propagation

▶ **good**: fast subproblem processing
▶ **bad**: LP relaxation has no knowledge about the nonlinear structure

Only relaxation – put the complete linearization into the LP

▶ **good**: LP relaxation contains complete problem
▶ **bad**: can blow up the LP

Only separation – generate the linearization as they are needed

▶ **good**: the LP only is fed with the import constraint (cuts)
▶ **bad**: LP relaxation has partial knowledge about the nonlinearities
Enforcement of AND Constraints

Only propagation
  ▶ good: fast subproblem processing
  ▶ bad: LP relaxation has no knowledge about the nonlinear structure

Only relaxation – put the complete linearization into the LP
  ▶ good: LP relaxation contains complete problem
  ▶ bad: can blow up the LP

Only separation – generate the linearization as they are needed
  ▶ good: the LP only is fed with the import constraint (cuts)
  ▶ bad: LP relaxation has partial knowledge about the nonlinearities

SCIP’s strategy
propagation + dynamically decide between relaxation and separation
SCIP’s Modular, Plugin-based Structure
SCIP’s Modular, Plugin-based Structure
SCIP’s Solving Loop

Start → Init → Presolving → Stop

Node selection → Conflict analysis → Primal heuristics

Branching → Processing

Domain propagation

Solve Relaxation → Pricing → Cuts

Enforce constraints

Constraint Integer Programming

MIP + CP = CIP

Solving Constraint Integer Programs

The SCIP Optimization Suite

Using SCIP
What is SCIP?

SCIP (Solving Constraint Integer Programs) . . .

▶ provides a full-scale MIP and MINLP solver,
▶ incorporates
  ▶ CP features (domain propagation),
  ▶ MIP features (cutting planes, LP relaxation), and
  ▶ SAT-solving features (conflict analysis, restarts),
▶ has an modular structure,
▶ can be extended via plugins,
▶ is a branch-cut-and-price framework,
▶ is free for academic purposes,
▶ and is available in source-code under http://scip.zib.de!
SCIP Optimization Suite = SCIP + SoPlex + ZIMPL

Toolbox for *generating* and *solving* constraint integer programs

### ZIMPL
- Zuse Institute Mathematical Programming Language
- Model and generate LPs, MIPs, and MINLPs

### SCIP
- MIP, MINLP and CIP solver, branch-cut-and-price framework
- ZIMPL models can directly be loaded into SCIP

### SoPlex
- Default LP solver within SCIP
- Revised primal and dual simplex algorithm
- Special support for rational LP solving
ZIMPL
- modeling language

SCIP
- MIP, MINLP and CIP solver, branch-cut-and-price framework

SoPlex
- LP solver

GCG
- generic branch-cut-and-price solver
- based on Dantzig-Wolfe decomposition

UG
- framework for parallelization of MIP and MINLP solvers
- shared and distributed memory capabilities
Performance

- one of the **fastest non-commercial MIP solvers**

![Graph showing performance comparison between different solvers](image1)

<table>
<thead>
<tr>
<th>Solver</th>
<th>Time in Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLPK 4.52</td>
<td>4.8x</td>
</tr>
<tr>
<td>Ipsove 5.5.2</td>
<td>4.08x</td>
</tr>
<tr>
<td>CBC 2.8.7</td>
<td>2.23x</td>
</tr>
<tr>
<td>SCIP 3.1.0 – CLP 1.15.6</td>
<td>2.02x</td>
</tr>
<tr>
<td>SCIP 3.1.0 – SoPlex 2.0.0</td>
<td>1.00x</td>
</tr>
<tr>
<td>Couenne 0.4</td>
<td>0.72x</td>
</tr>
<tr>
<td>Cplex 12.6.0</td>
<td>0.24x</td>
</tr>
<tr>
<td>Gurobi 5.6.0</td>
<td>0.19x</td>
</tr>
<tr>
<td>LindoAPI 7.0.1.497</td>
<td>0.18x</td>
</tr>
</tbody>
</table>

(data: Hans Mittelmann, graphics: ZIB)

- one of the **fastest MINLP solvers**

![Graph showing performance comparison between different solvers](image2)

<table>
<thead>
<tr>
<th>Solver</th>
<th>Time in Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCIP 3.0.0 – CPLEX 12.4.0</td>
<td>1.00x</td>
</tr>
<tr>
<td>BARON 11.3</td>
<td>1.82x</td>
</tr>
<tr>
<td>Couenne 0.4</td>
<td>2.05x</td>
</tr>
<tr>
<td>LindoAPI 7.0.1.497</td>
<td>2.40x</td>
</tr>
</tbody>
</table>

(results on MINLPLIB (August 2012), all instances that can be handled by all solvers)
History of SCIP and Friends

1996  SoPlex – Sequential obj. simPlex (R. Wunderling [now IBM Cplex])
1998  SIP – Solving Integer Programs (A. Martin [now Univ. Erlangen])
2002  Start of SCIP development (T. Achterberg [now Gurobi])
2003  Chipdesign verification ⇒ constraint programming
2005  First public version 0.80
2007  Start of exact integer programming development
2007  SCIP 1.0
2008  Pseudo-Boolean optimization and Generic Column Generation
2009  Mixed-integer nonlinear programming
2009  Beale-Orchard-Hays Prize (T. Achterberg)
2012  UG framework for parallelization
2014  SoPlex 2.0 with support for rational LP solving
Developers at ZIB

- Thorsten Koch
- Marc Pfetsch @ TU Darmstadt
- Gerald Gamrath
- Ambros Gleixner
- Matthias Miltenberger
- Felipe Serrano
- Yuji Shinano
- Kati Wolter
- Students: Gregor Hendel, Eva Ramlow, Michael Winkler, Benjamin Müller, Leif Naundorf, Luca Fabbri

*
Developers at ZIB

▶ Thorsten Koch
▶ Marc Pfetsch @ TU Darmstadt
▶ Gerald Gamrath
▶ Ambros Gleixner
▶ Matthias Miltenberger
▶ Felipe Serrano
▶ Yuji Shinano
▶ Kati Wolter
▶ Students: Gregor Hendel, Eva Ramlow, Michael Winkler, Benjamin Müller, Leif Naundorf, Luca Fabbri

Former developers
▶ Tobias Achterberg @ Gurobi
▶ Stefan Vigerske @ GAMS
▶ Timo Berthold @ FICO XPRESS
▶ Stefan Heinz @ FICO XPRESS
SCIP Facts

- more than 500,000 lines of C code, 25% documentation
  → 28,000 assertions, 4,000 debug messages
- 10 examples illustrating the use of SCIP
- HowTos: each plugin type, debugging, automatic testing, ...
- C++ wrapper classes, python interface, Java

- supports 10+ different input formats
- 7 interfaces to external LP solvers
  → CLP, CPLEX, Gurobi, Mosek, QSopt, SoPlex, XPRESS
- more than 1000 parameters, 15 “emphasis” settings
- active mailing list (200+ members)
- 8000+ downloads per year from 100+ countries

- free for academics, available in source code: http://scip.zib.de
- runs on Linux, Windows, Mac (Darwin+PPC), SunOS, ...
(Some) Universities and Institutes using SCIP

- RWTH Aachen
- Universität Bayreuth
- FU Berlin
- TU Berlin
- HU Berlin
- WIAS Berlin
- TU Braunschweig
- TU Chemnitz
- TU Darmstadt
- TU Dortmund
- TU Dresden
- Universität Erlangen-Nürnberg
- Leibniz Universität Hannover
- Universität Heidelberg
- Fraunhofer ITWM
- Kaiserslautern
- Universität Karlsruhe
- Christian-Albrechts-Universität zu Kiel
- Hochschule Lausitz
- OvGU Magdeburg
- TU München
- Universität Osnabrück
- Universität Stuttgart
- Aarhus Universitet
- The University of Adelaide
- Università dell’Aquila
- Arizona State University
- University of Assiut
- National Technical University of Athens
- Georgia Institute of Technology
- Indian Institute of Science
- Tsinghua University
- UC Berkeley
- Lehigh University
- University of Bristol
- Eötvös Loránd Tudományegyetem
- Universidad de Buenos Aires
- Institut Français de Mécanique Avancée
- Chuo University
- Clemson University
- University College Cork
- Danmarks Tekniske Universitet
- Syddansk Universitet
- Fuzhou University
- Jinan University
- Rijksuniversiteit Groningen
- Hanoi Institute of Mathematics
- The Hong Kong Polytechnic University
- University of Hyogo
- The Irkutsk Scientific Center
- University of the Witwatersrand
- Københavns Universitet
- Kunming Botany Institute
- École Poly. Fédérale de Lausanne
- Linköpings universitet
- Université catholique de Louvain
- Universidad Rey Juan Carlos
- Université de la Méditerranée Aix-Marseille
- University of Melbourne
- UNAM
- Politecnico di Milano
- Università degli Studi di Milano
- Monash University
- Ikerlan
- Université de Montréal
- NIISI RAS
- Université de Nantes
- The University of Newcastle
- University of Nottingham
- Universitetet i Oslo
- Università degli Studi di Padova
- L’Université Sud de Paris
- Brown University
- The University of Queensland
- IASI CNR
- Erasmus Universiteit Rotterdam
- Carnegie Mellon University
- Universidad Diego Portales
- University of Balochistan
- Universidad San Francisco de Quito
- Universidade Federal do Rio de Janeiro
- Universidade de São Paulo
- Fudan University
- University of New South Wales
- Tel Aviv University
- The University of Tokyo
- Politecnico di Torino
- University of Toronto
- NTNU i Trondheim
- The University of York
- University of Washington
- University of Waterloo
- Massey University
- Austrian Institute of Technology
- TU Wien
- Universität Wien
- Wirtschaftsuniversität Wien
- ETH Zürich
(Some) Universities and Institutes using SCIP
SCIP to go!
Constraint Integer Programming

MIP + CP = CIP

Solving Constraint Integer Programs

The SCIP Optimization Suite

Using SCIP
Install SCIP

- download SCIP Optimization Suite from http://scip.zib.de
- extract tarball
- compile the SCIP Optimization Suite with make
- test binary with make test

- download precompiled binaries from http://scip.zib.de
- start SCIP ./scip-3.1.0/bin/scip
- SCIP>
SCIP> read check/instances/MINLP/circle.cip
SCIP> optimize
SCIP> display solution
SCIP> display statistics
SCIP> help
SCIP> quit
Interactive Shell – Settings/Parameters

- SCIP comes with a bunch of predefined settings
  - cuts, heuristics, and presolving
    - aggressive
    - fast
    - off
  - emphasis
    - counter
    - cpsolver
    - easymip
    - feasibility
    - hardlp
    - optimality

- in the interactive shell you can modify parameters, for example,
  - each parameter is shown with a description and value range

SCIP> set {heuristics|...} emphasis {fast| ...}
SCIP> set emphasis {feasibility|...}
SCIP> set heuristics feaspump freq 1
Callable Library – Generate a Problem

Using SCIP within another code

- problem object
  SCIPcreate()
  SCIPcreateProb()

- problem variables
  SCIPcreateVar()
  SCIPaddVar()
  SCIPreleaseVar()

- problem constraints
  SCIPcreateConsLinear(), SCIPcreateConsQuadratic(), ...
  SCIPaddCoefLinear()
  SCIPaddCons()
  SCIPreleaseCons()

See: scip.h, pub_*.h, <plugin>.h (e.g., cons_linear.h)
Callable Library – Solve a Problem

- setting parameters
  SCIPsetBoolParam()
  SCIPsetIntParam()
  SCIPsetRealParam()

- solve problem
  SCIPpresolve()
  SCIPsolve()

- get solution
  SCIPgetBestSol()
  SCIPgetSolVals()

- free problem
  SCIPfree()

See: scip.h, pub_*.h, <plugin>.h (e.g. cons_linear.h)
SCIP as a Framework

- extend SCIP by adding plugins in C or C++
  - branching rule, constraint handler, dialog, display, heuristic, node selector, presolver, propagator, reader, relaxation, separator
  - there are template files for all plugin types, e.g., cons_xyz.{c,h}
- extended framework can be used via
  - interactive shell
  - as callable library
- the SCIP release comes with 10 examples
  - Binpacking – starting branch-and-price example
  - Coloring – elaborated branch-and-price example
  - LOP – constraint handler example, branch-and-cut
  - TSP – C++ branch-and-cut example, primal heuristics
  - MIPSolver – starting example
  - Queen – placing n queens on a chessboard, callable library
  - ...
TSP Plugins
TSP Plugins
TSP Plugins

SCIP

Primal Heuristic

farthest insert

diving

2-Opt

... obj

mcf

odd cycle

rapid learn

redcost

strong cg

default

Propagator

... obj

root

redcost

default

Relax

Primal Heuristic

branch

... obj

incident

default

Node selector

default

Preprocessing

default

Node selector

default

Display

default

Dialog

default

Cutpool

default

Conflict

default

Implied

default

Implications

default

Solver

default

Solver

default

Branch

default

Solver

default

Branch

default

Solver

default

Branch

default

Solver

default

Branch

default

Solver
### lines of C++ code in TSP-example

<table>
<thead>
<tr>
<th>Component</th>
<th>Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>main program</td>
<td>196</td>
</tr>
<tr>
<td>graph structure</td>
<td>80</td>
</tr>
<tr>
<td>TSP file reader</td>
<td>407</td>
</tr>
<tr>
<td>nosubtour constraint</td>
<td>793</td>
</tr>
<tr>
<td>farthest insert heuristic</td>
<td>354</td>
</tr>
<tr>
<td>2-Opt heuristic</td>
<td>304</td>
</tr>
<tr>
<td>Gomory-Hu algo</td>
<td>658</td>
</tr>
<tr>
<td>altogether</td>
<td>2134</td>
</tr>
</tbody>
</table>
Cutting circles from area-minimizing rectangles

Task
Given \( n \) circles of radii \( r_1, \ldots, r_n > 0 \), find a rectangle of width \( w \) and height \( h \) such that all circles fit into the rectangle without overlap and the area \( hw \) is minimized.

Download
▷ SCIP binary from http://scip.zib.de
Cutting circles from area-minimizing rectangles

Task
Given \( n \) circles of radii \( r_1, \ldots, r_n > 0 \), find a rectangle of width \( w \) and height \( h \) such that all circles fit into the rectangle without overlap and the area \( hw \) is minimized.

Download
▷ SCIP binary from http://scip.zib.de
Take-away messages

- MIP solving is a bag of tricks
- you can add your own ones (framework)

Thank you very much for your attention!

Muito obrigado!
Take-away messages

- MIP solving is a bag of tricks
- you can add your own ones (framework)

- there are various interesting extensions of MIP
- you can add your own ones (framework)
Take-away messages

- MIP solving is a bag of tricks
- you can add your own ones (framework)

- there are various interesting extensions of MIP
- you can add your own ones (framework)

- SCIP is a fast, stable, non-commercial MIP solver
- SCIP is not only a MIP solver
Take-away messages

- MIP solving is a bag of tricks
- you can add your own ones (framework)

- there are various interesting extensions of MIP
- you can add your own ones (framework)

- SCIP is a fast, stable, non-commercial MIP solver
- SCIP is not only a MIP solver

Thank you very much for your attention!
Muito obrigado!
Introduction to Constraint Integer Programming

Ambros M. Gleixner

Zuse Institute Berlin · MATHEON · Berlin Mathematical School

5th Porto Meeting on Mathematics for Industry, April 10–11, 2014, Porto