SCIP

. . . und Algorithm Engineering

Marc Pfetsch

Institut für Mathematische Optimierung
TU Braunschweig

Überblick

1. Einführung
2. Branch-And-Cut und Constraint Programming
3. SCIP
   - SCIP Plugins
   - Presolving
   - Lösen
   - Dokumentation
4. Algorithm Engineering

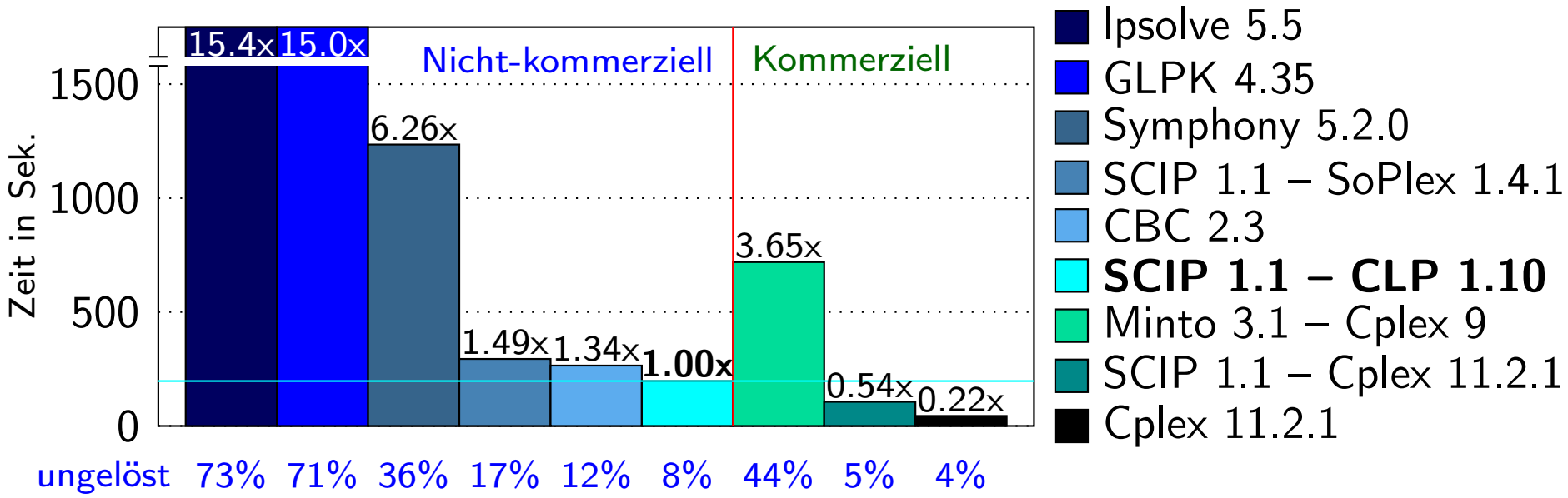
Werbefolie

SCIP (Solving Constraint Integer Programs) ...

- ist ein Branch-and-Cut-and-Price Framework,
- beinhaltet einen schnellen Löser für Gemischt-Ganzzahlige Programme (Mixed Integer Programs – MIPs),
- kann auch Constraint Programs (CP) lösen,
- beinhaltet einige SAT-Löser-Techniken (Konflikt-Analyse, Restarts),
- ist frei für akademische Zwecke,
- und ist im Source-Code erhältlich!

Vergleich von MIP-Lösern

Werbefolie



- Quelle: Hans Mittelmann (`http://plato.asu.edu/ftp/milpf.html`)
- MIP-Löser ungefähr so schnell wie CPLEX 9.0
- LP-Löser werden als „Black-Box“ verwendet.

SCIP Geschichte

1998	SIP – Solving Integer Programs (Alexander Martin)
10/2002	Beginn der SCIP-Entwicklung (Tobias Achterberg)
02/2003	Erste Version die MIPs lösen kann
05/2003	Gomory Cuts
08/2003	Konflikt-Analyse
03/2004	Knapsack Cover Separator (Kati Wolter)
04/2004	Restarts, c-MIR Cuts
01/2005	Feasibility Pump (Timo Berthold)
09/2005	Erste öffentliche Version 0.80
09/2006	Version 0.90
09/2007	Version 1.00
09/2008	Version 1.1.0

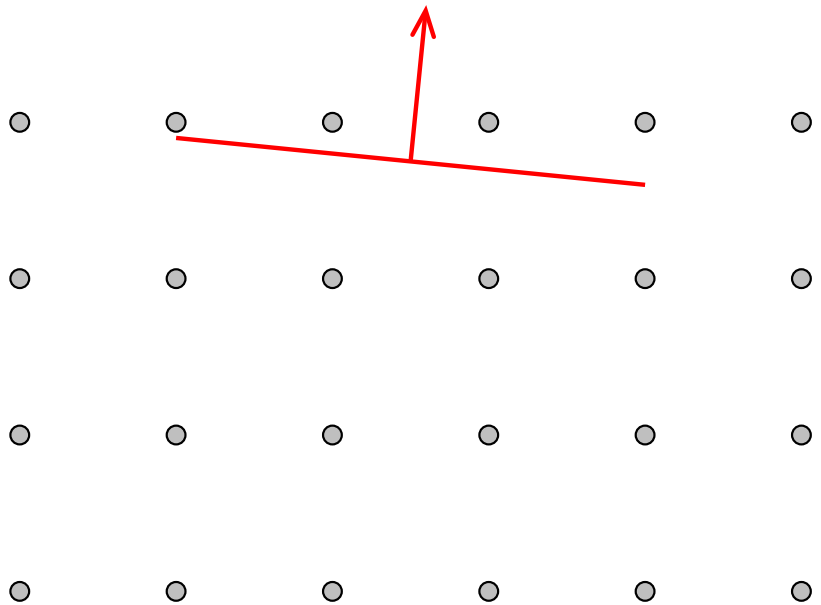
SCIP Informationen

- Ungefähr 220.000 Zeilen C-Code
- Hauptentwickler: Tobias Achterberg → ILOG/IBM
  Timo Berthold (Heuristiken), Kati Wolter (Cuts)
- Entwickelt am Zuse-Institut Berlin (ZIB)
  zusammen mit der TU Darmstadt.
- Frei für akademische Nutzung (wie SoPlex)
- Verfügbar auf `http://scip.zib.de`
- ZIB Optimization Suite = SCIP + SoPlex + ZIMPL
  `http://zibopt.zib.de`

Definition: Mixed Integer Program (MIP)

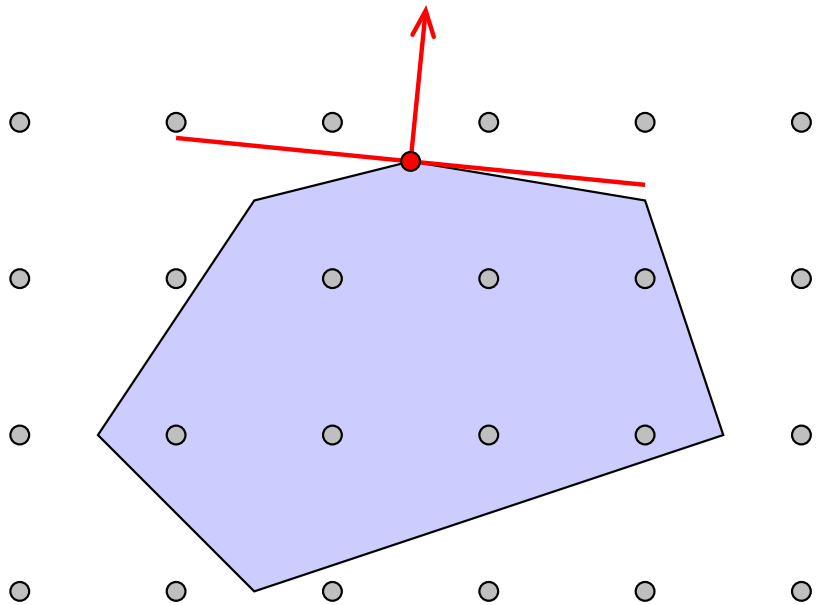
Definition: Mixed Integer Program (MIP)

$$\min \quad c^T x$$



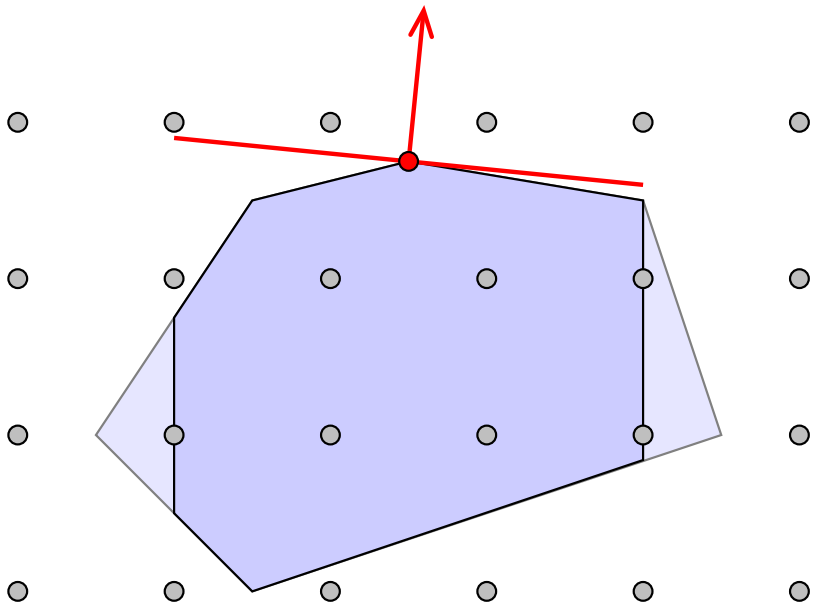
Definition: Mixed Integer Program (MIP)

$$\begin{aligned} \min \quad & c^T x \\ s.t. \quad & Ax \leq b \end{aligned}$$



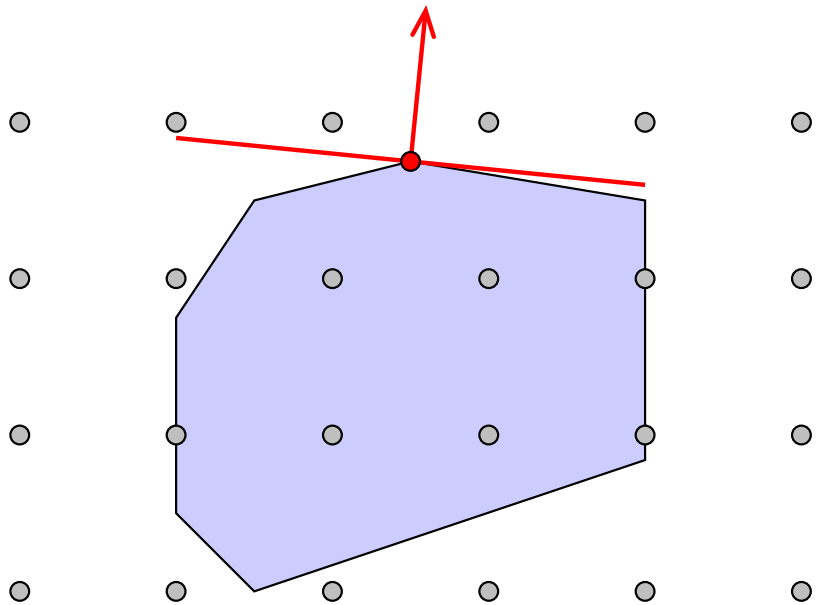
Definition: Mixed Integer Program (MIP)

$$\begin{aligned} \min \quad & c^T x \\ s.t. \quad & Ax \leq b \\ & l \leq x \leq u \end{aligned}$$



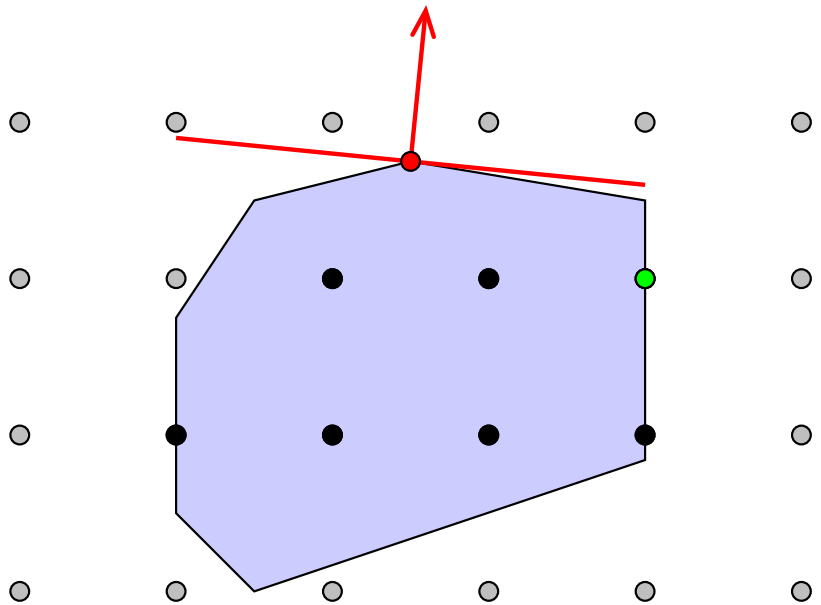
Definition: Mixed Integer Program (MIP)

$$\begin{aligned} \min \quad & c^T x \\ s.t. \quad & Ax \leq b \\ & l \leq x \leq u \end{aligned}$$



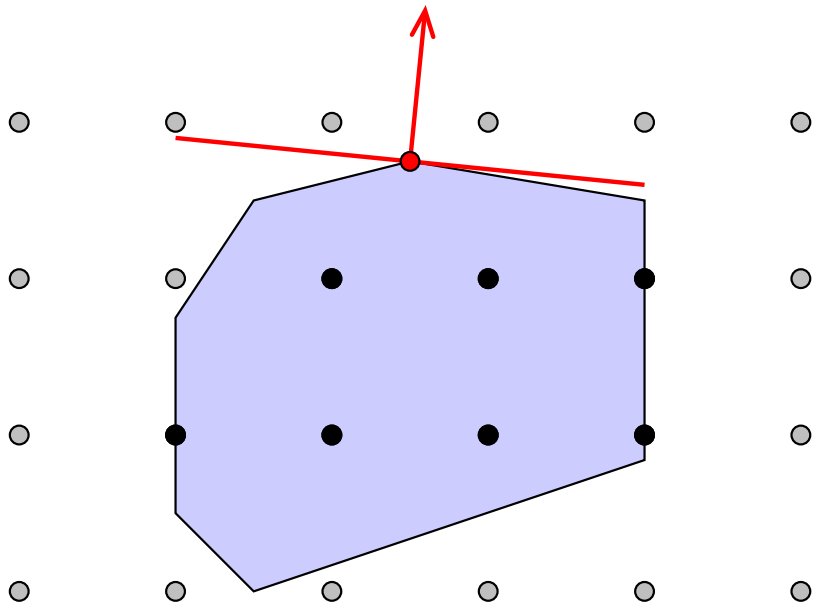
Definition: Mixed Integer Program (MIP)

$$
\begin{aligned}
\min \quad & c^T x \\
s.t. \quad & Ax \leq b \\
& l \leq x \leq u \\
& x_j \in \mathbb{Z} \quad j \in J
\end{aligned}
$$



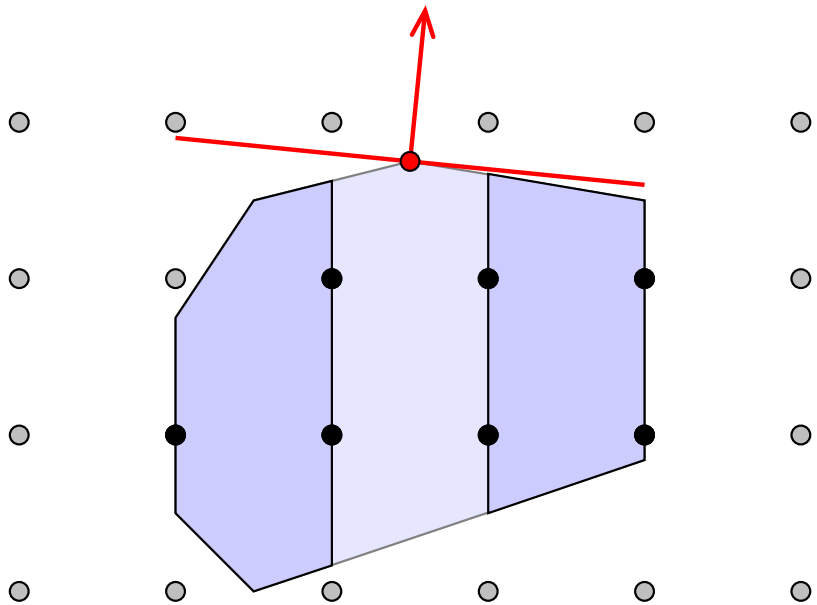
Kombination von

- Branch-and-Bound
- Schnittebenen (Cutting Planes)



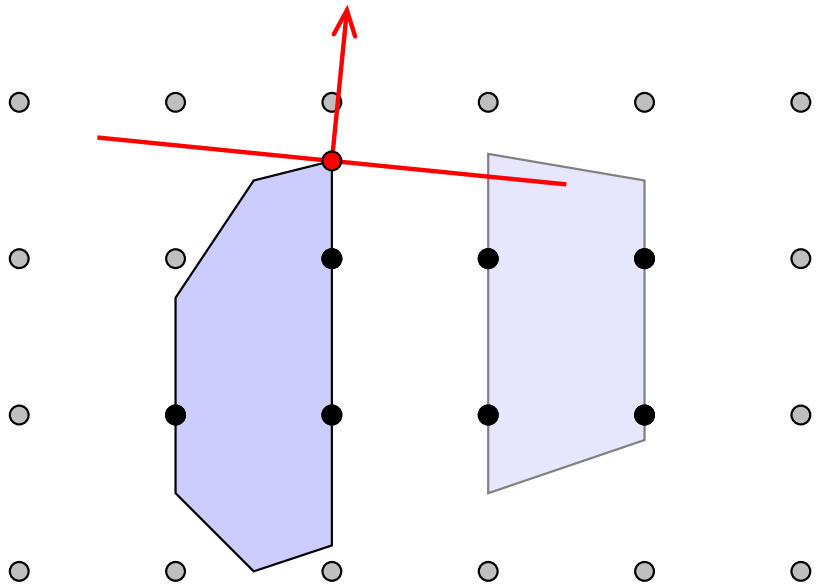
Kombination von

- Branch-and-Bound
- Schnittebenen (Cutting Planes)



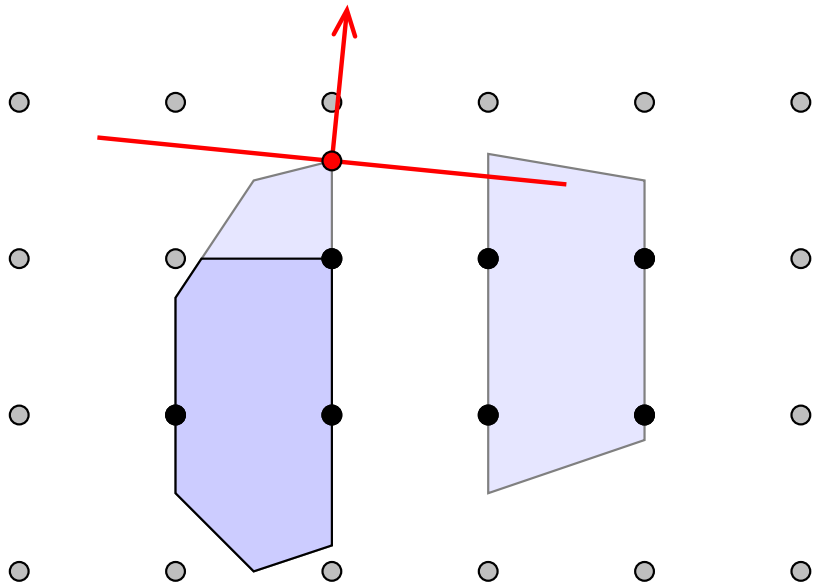
Kombination von

- Branch-and-Bound
- Schnittebenen (Cutting Planes)



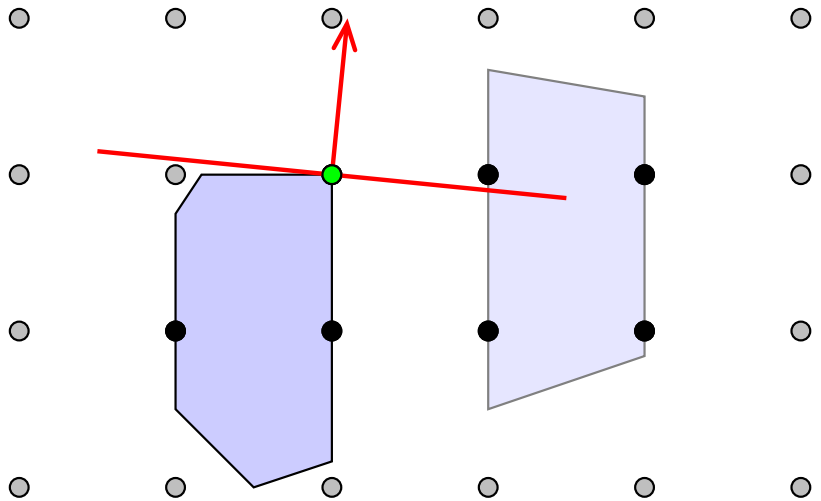
Kombination von

- Branch-and-Bound
- Schnittebenen (Cutting Planes)



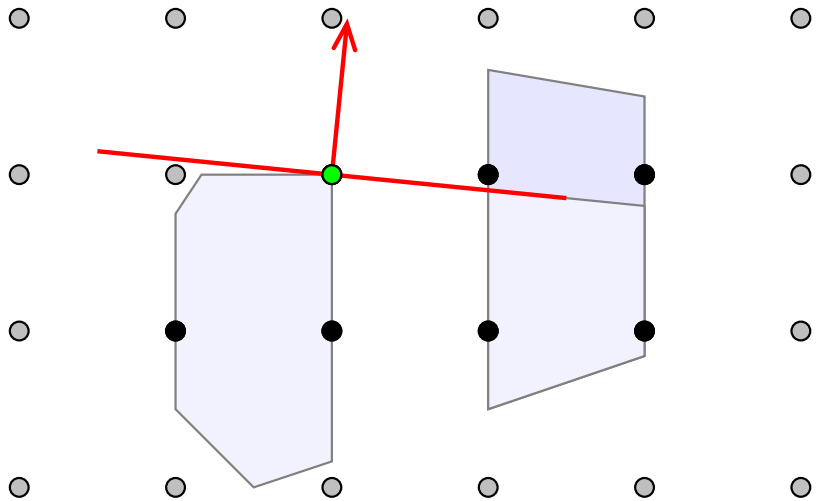
Kombination von

- Branch-and-Bound
- Schnittebenen (Cutting Planes)



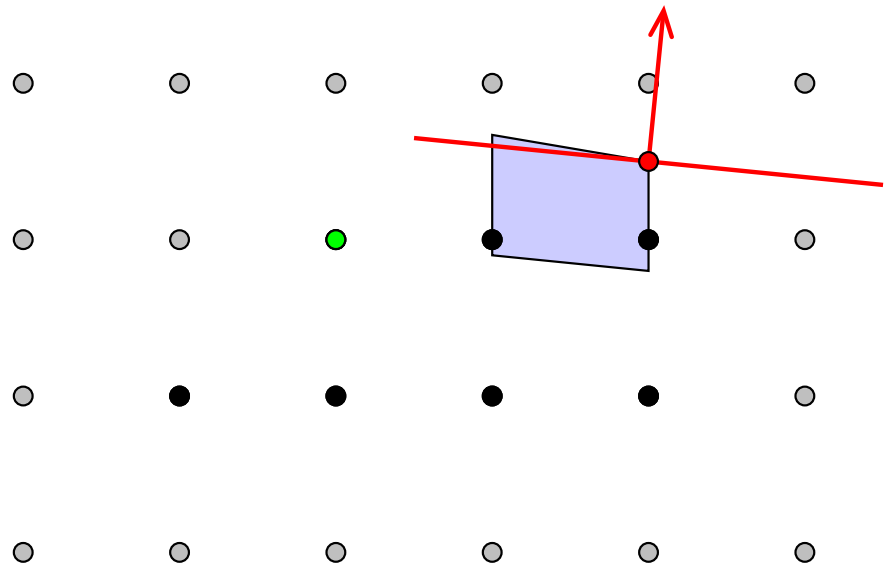
Kombination von

- Branch-and-Bound
- Schnittebenen (Cutting Planes)



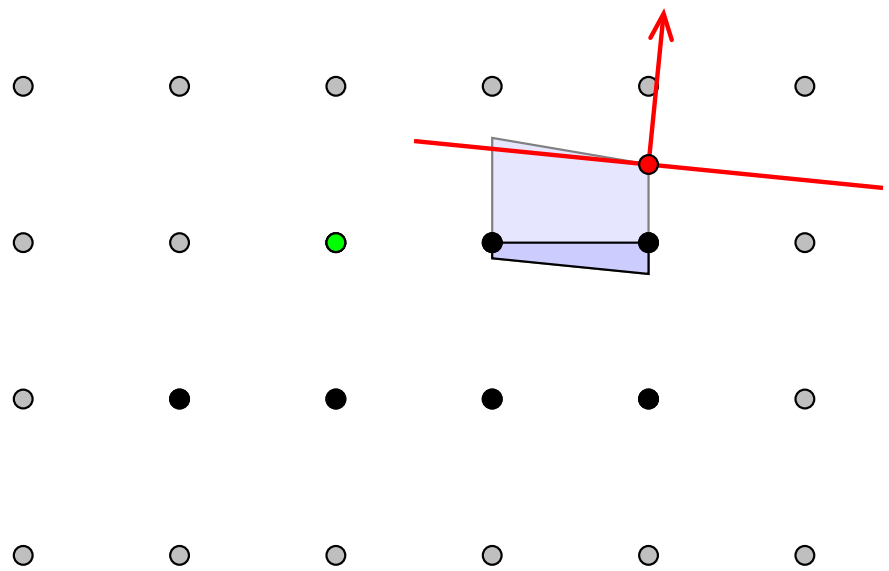
Kombination von

- Branch-and-Bound
- Schnittebenen (Cutting Planes)



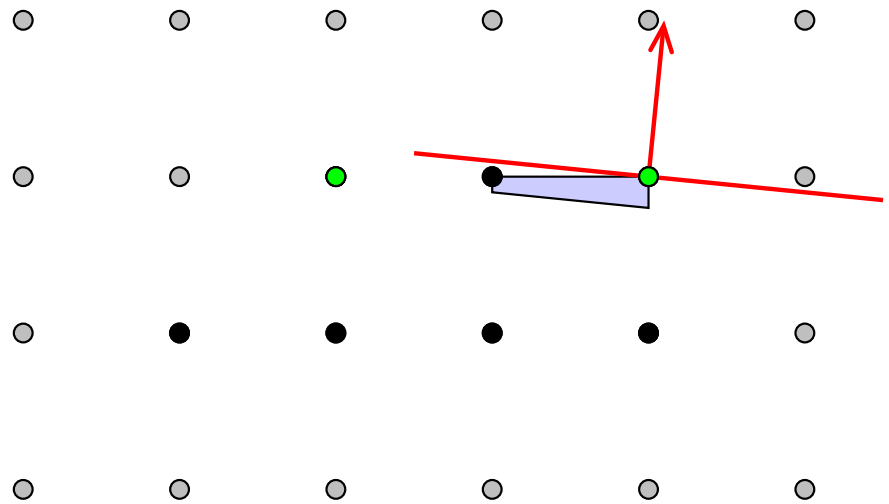
Kombination von

- Branch-and-Bound
- Schnittebenen (Cutting Planes)



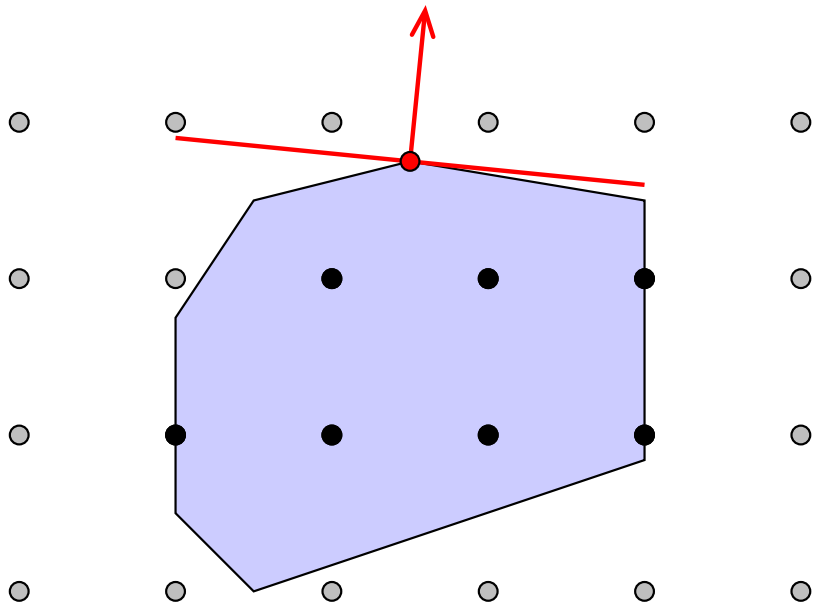
Kombination von

- Branch-and-Bound
- Schnittebenen (Cutting Planes)



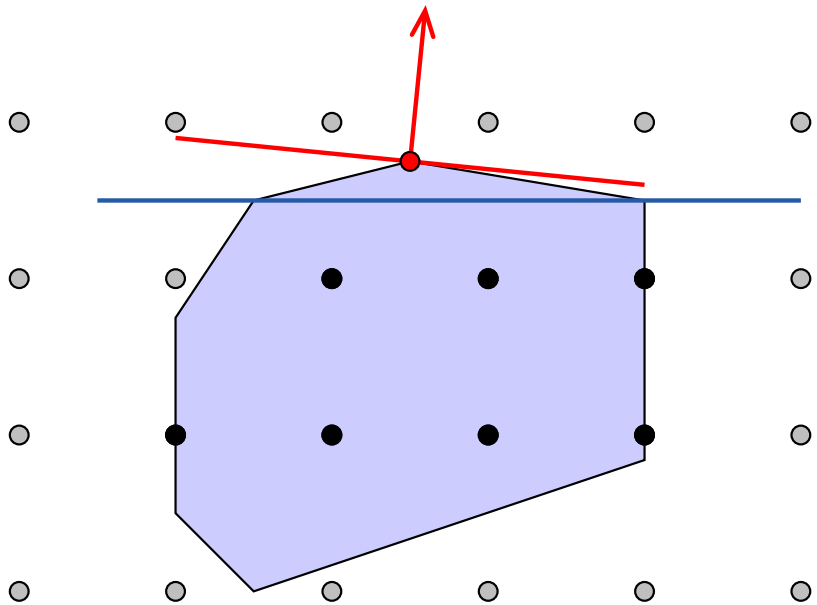
Kombination von

- Branch-and-Bound
- Schnittebenen (Cutting Planes)



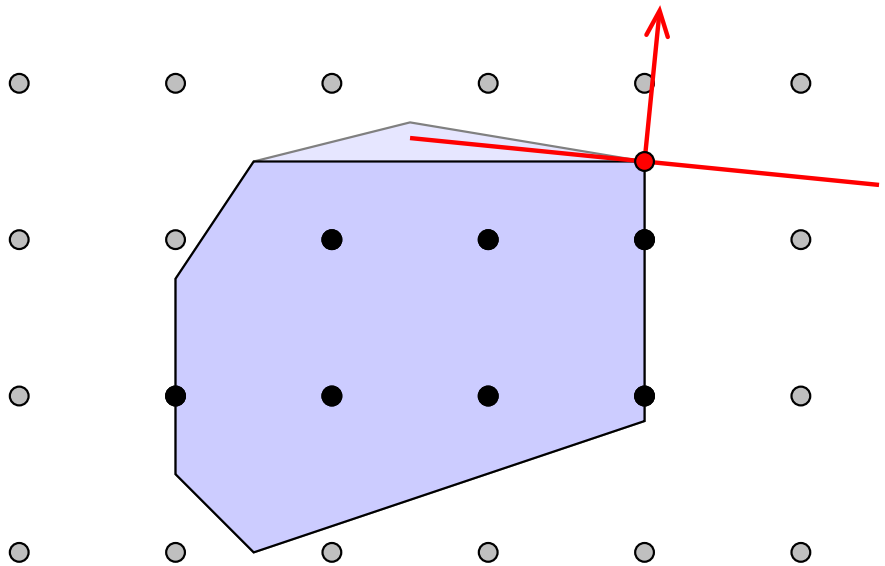
Kombination von

- Branch-and-Bound
- Schnittebenen (Cutting Planes)



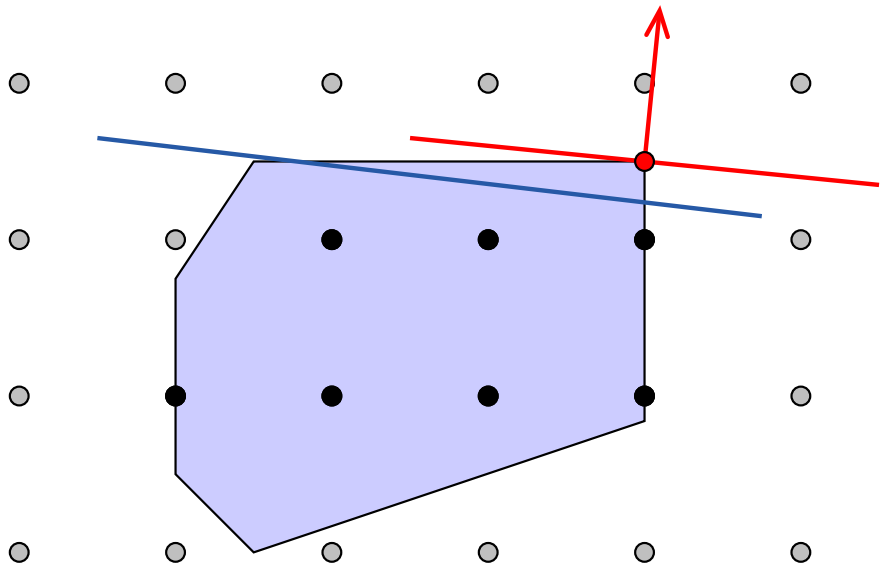
Kombination von

- Branch-and-Bound
- Schnittebenen (Cutting Planes)



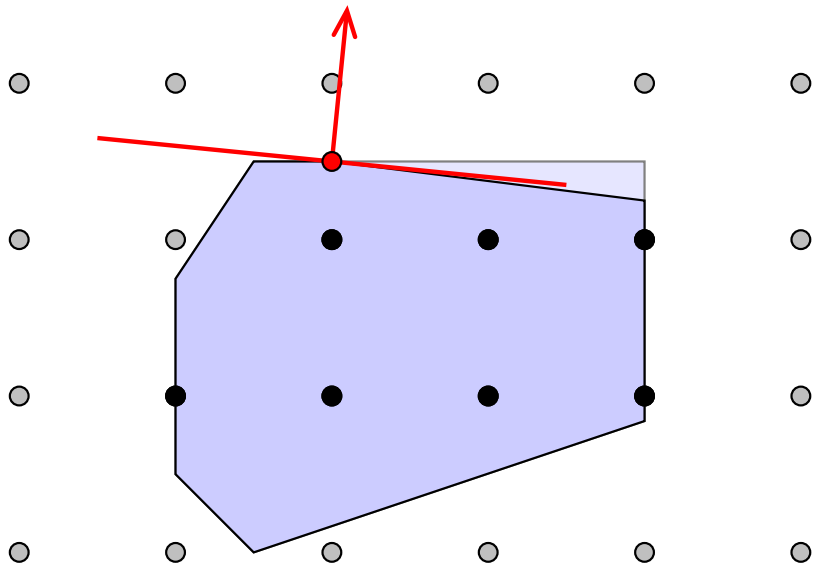
Kombination von

- Branch-and-Bound
- Schnittebenen (Cutting Planes)



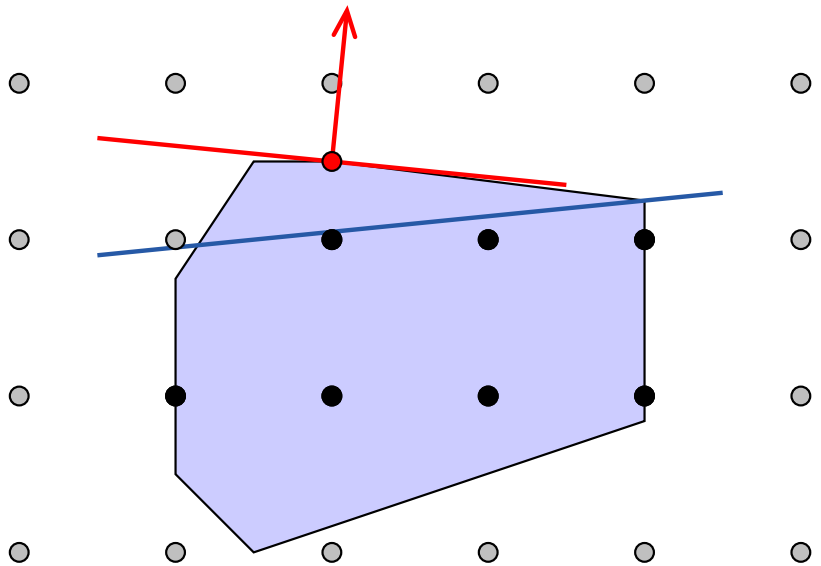
Kombination von

- Branch-and-Bound
- Schnittebenen (Cutting Planes)



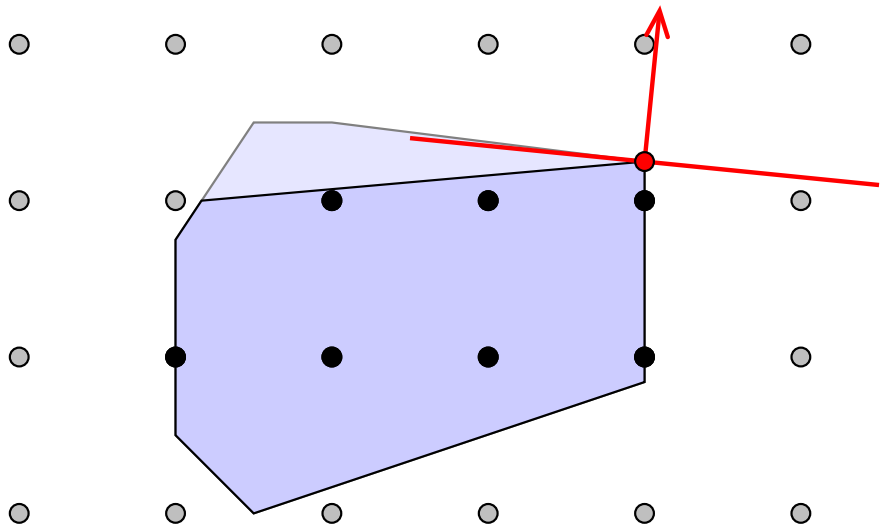
Kombination von

- Branch-and-Bound
- Schnittebenen (Cutting Planes)



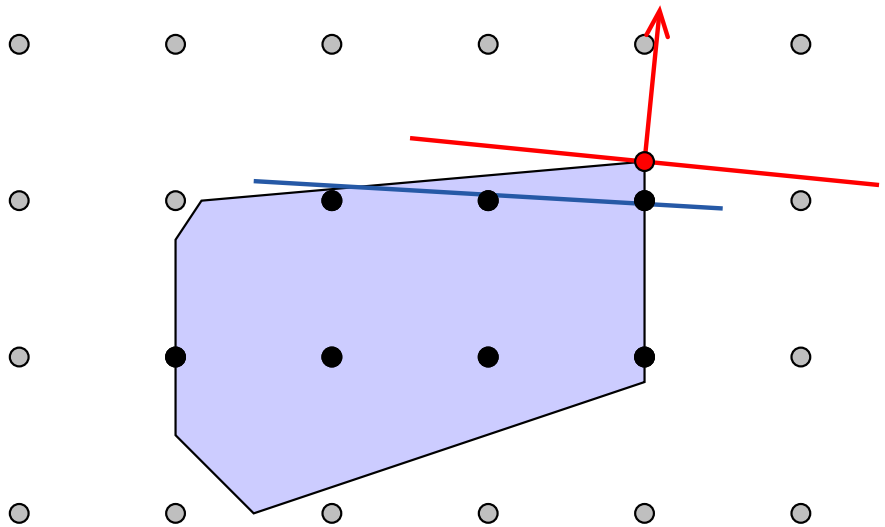
Kombination von

- Branch-and-Bound
- Schnittebenen (Cutting Planes)



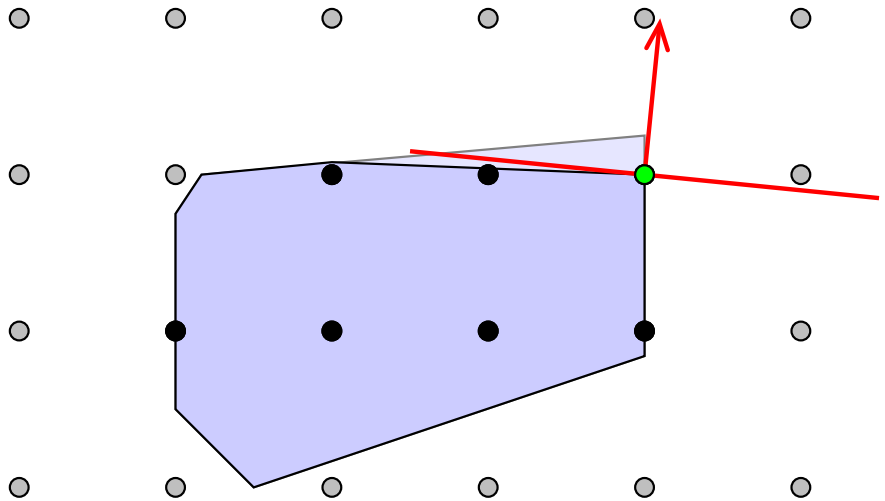
Kombination von

- Branch-and-Bound
- Schnittebenen (Cutting Planes)



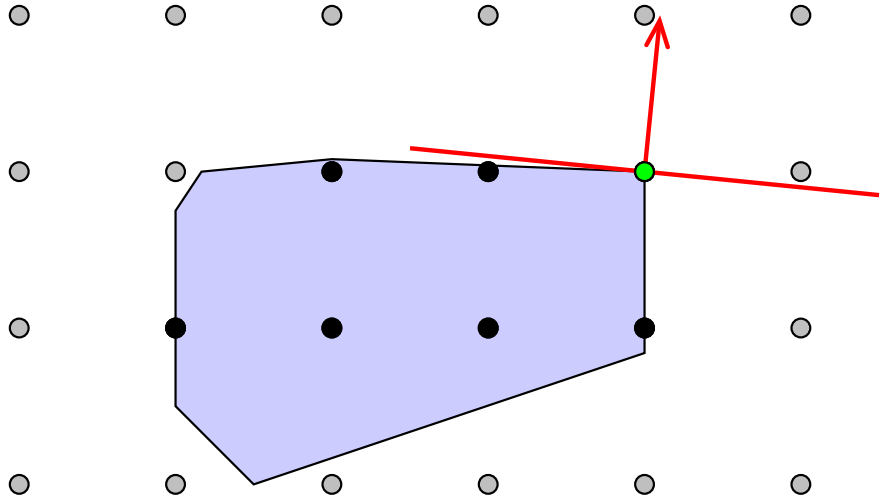
Kombination von

- Branch-and-Bound
- Schnittebenen (Cutting Planes)



Kombination von

- Branch-and-Bound
- Schnittebenen (Cutting Planes)



Constraint Program (CP)

- Zulässige Bereiche (Domains) der Variablen = (endliche) Menge
- Nebenbedingungen (Constraints) = Teilmengen des zul. Bereichs

Beispiel:

- Variablen x und y
- Domains: $D_x = D_y = \{0, 1\}$
- Constraint: $\{(0, 1), (1, 0)\} \subset D_x \times D_y \Leftrightarrow x = 1 - y$.

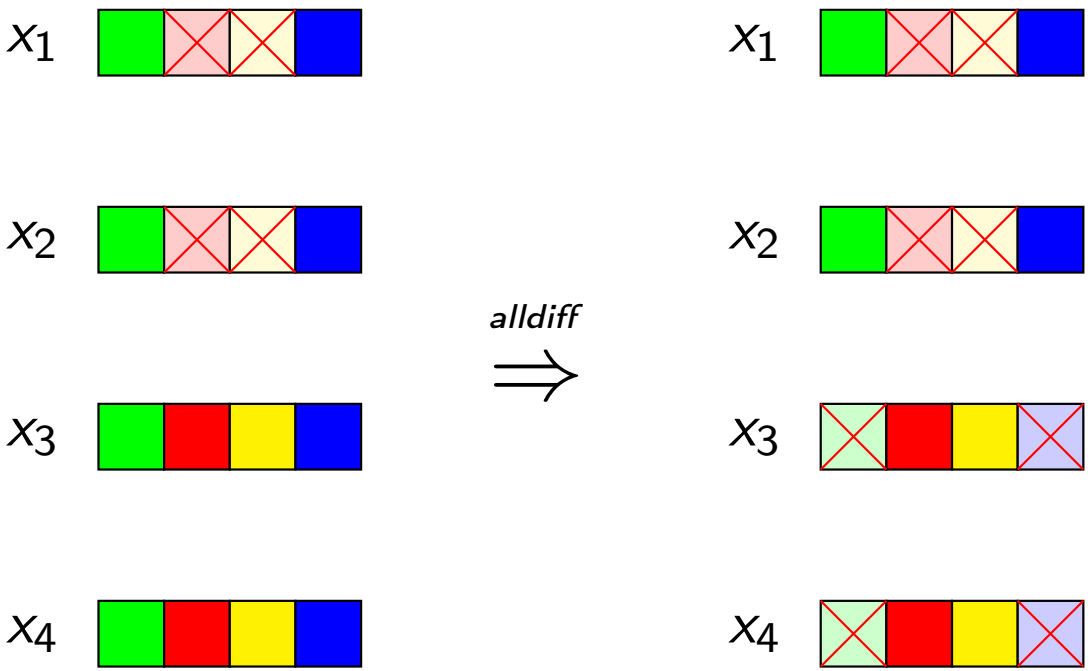
Constraint Integer Program (CIP)

Fixieren der ganzzahligen Variablen ergibt lineares Programm (LP).

- Branchen:
  - Teile in Subprobleme auf, löse rekursiv
- Domain Propagierung:
  - Reduktion der Variablen-Domains
  - Z. B.: $x_1 + 2x_2 \geq 5,\ x_1 \leq 2 \ \Rightarrow\ x_2 \geq 1.5$

Alldifferent Constraint:

$\{x_1, x_2, x_3, x_4\} \in \{1, 2, 3, 4\}$ paarweise verschieden.



- SCIP ist Constraint-basiert
  - Vorteil: Flexibilität
  - Nachteil: keine duale Repräsentierung
- *Ein Constraint kennt seine Variablen, aber eine Variable kennt die Constraints nicht in denen sie vorkommt.*
- Ein Constraint implementiert einen Typ von Ungleichungen, nicht eine einzelne Ungleichung.

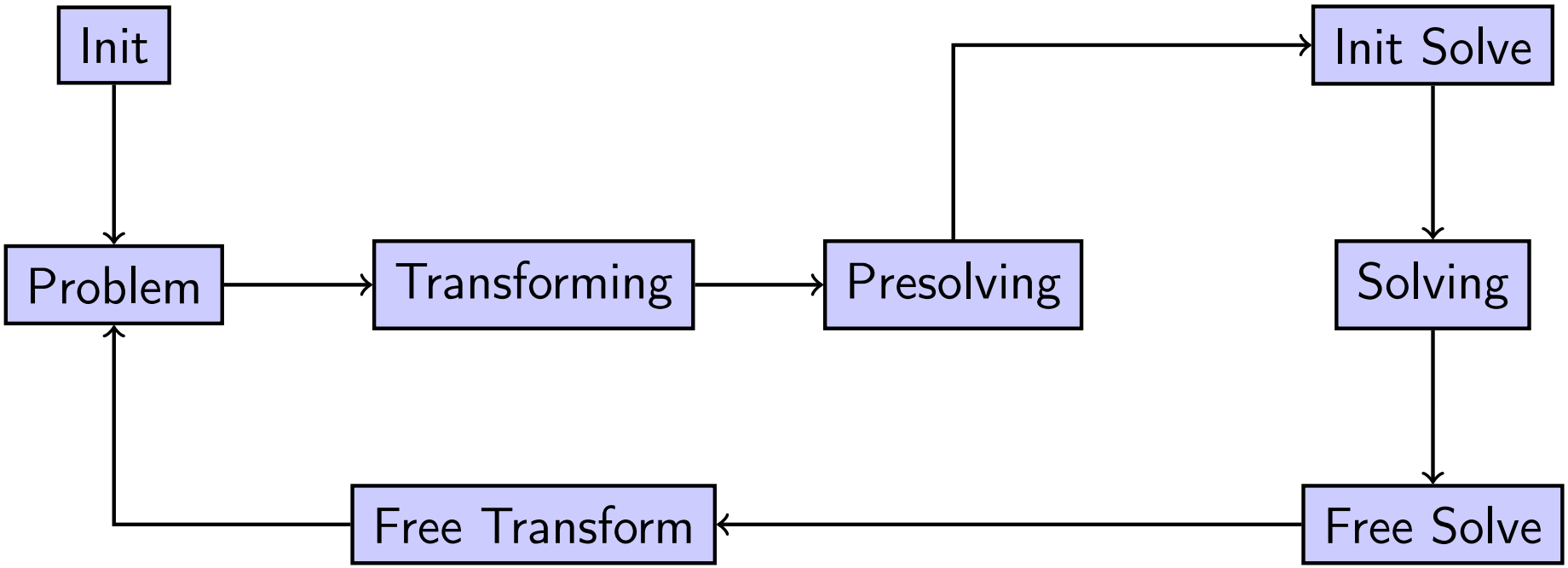
Traveling Salesman problem (TSP)

Gegeben ein Graph $G = (V, E)$ und Distanzen d_e, finde eine kürzeste Rundtour durch alle Knoten.

IP-Formulierung:

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

Beispiel in SCIP verfügbar, inklusive Visualisierung.


Init
Problem
Transforming
Presolving
Init Solve
Solving
Free Solve
Free Transform

Hauptinterface zu SCIP: via Plugins.

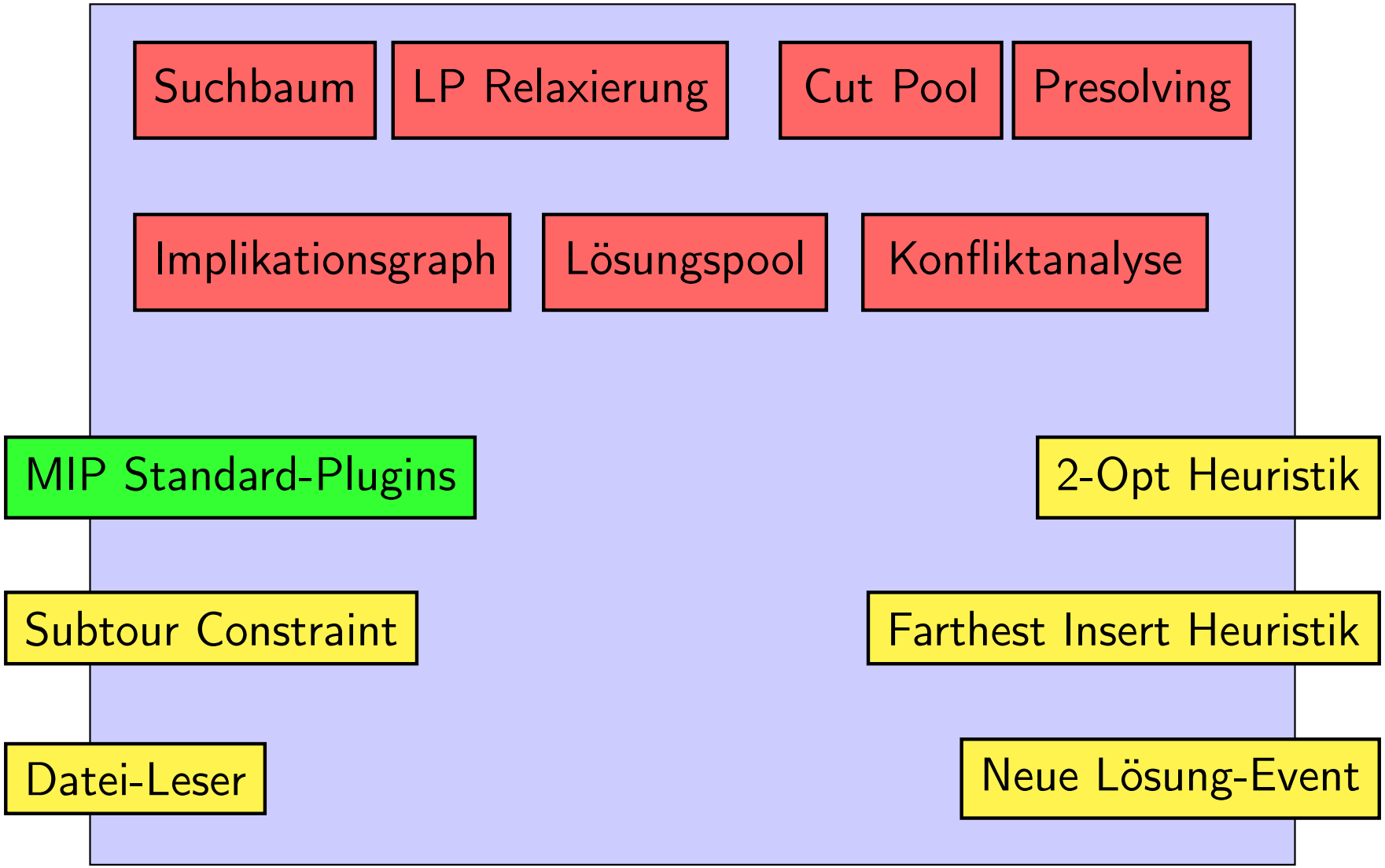
- Führen alle problemspezifischen Aktionen durch.
- Jeder Schritt ruft vom Benutzer definierte Plugins auf.
- SCIP kennt die Plugins durch "Anmelde"-Funktionen.
- Plugins können eigene (private) Daten haben.
- $\longrightarrow$ modulare Struktur.
- MIP-Löser ist eine Ansammlung von Plugins.

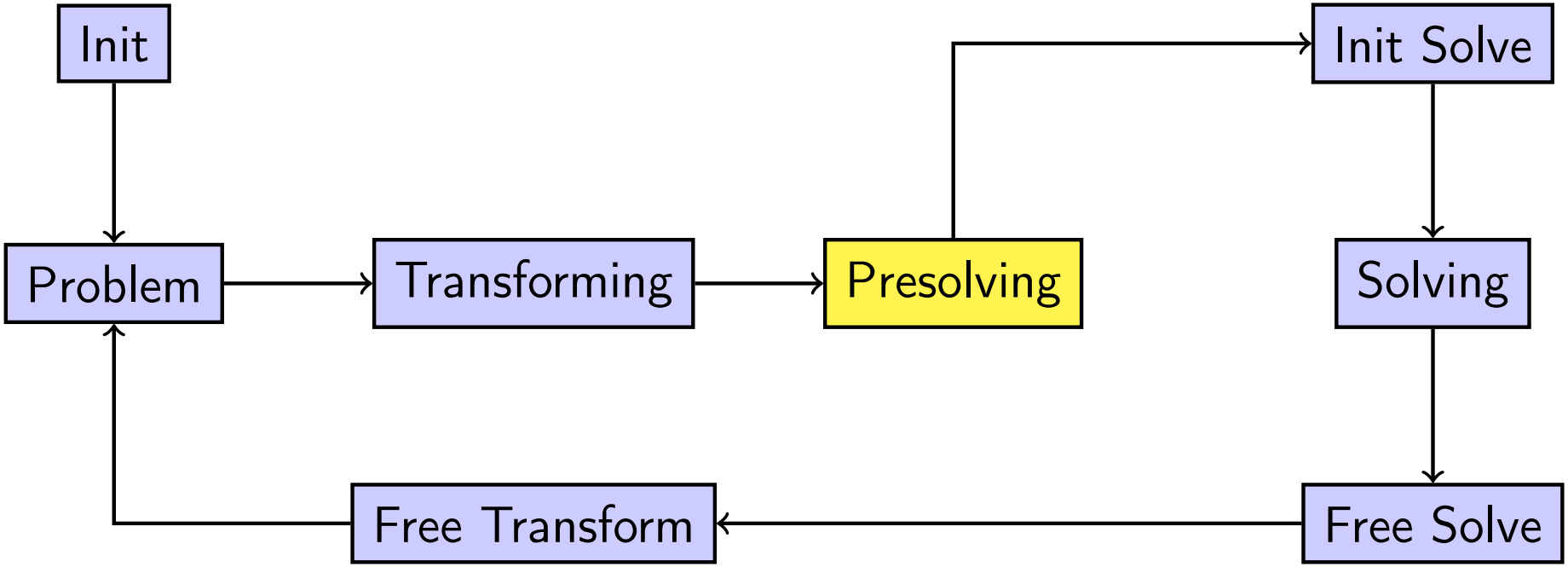
Typen von Plugins

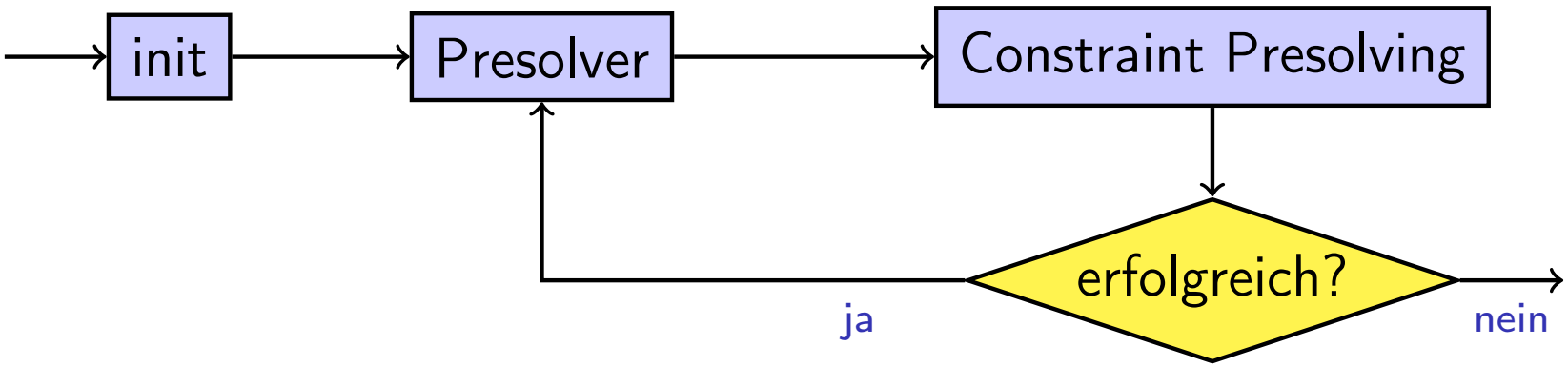
- Constraint Handler: stellt Zulässigkeit sicher, verstärkt Formulierung
- Separator: fügt Schnittebenen hinzu
- Pricer: Spaltengenerierung
- Heuristik: sucht zulässige Lösungen
- Branching-Regel: unterteilt das Problem
- Knotenauswahl: wählt nächstes Teilproblem (Knoten) aus
- Presolver: vereinfacht das Problem
- Propagator: vereinfacht das Problem (im Baum)
- Reader: liest Daten (verschiedene Formate)
- Event Handler: fängt Ereignisse ab (Bound-Änderungen, neue Lösungen, . . .)

5	Presolver
5	Knotenauswahlregeln
14	Constraint Handler
8	Separatoren
8	Branching-Regeln
2	Propagatoren
23	Heuristiken

TSP-Plugins


Suchbaum
LP Relaxierung
Cut Pool
Presolving
Implikationsgraph
Lösungspool
Konfliktanalyse
MIP Standard-Plugins
2-Opt Heuristik
Subtour Constraint
Farthest Insert Heuristik
Datei-Leser
Neue Lösung-Event


Init
Problem
Transforming
Presolving
Init Solve
Solving
Free Transform
Free Solve



- Presolver ermöglichen globales Presolving (z. B. duales Fixieren)
- Constraint Handler ermöglichen Constraint-spezifisches Presolving, z. B.:
  - Domain-Verkleinerung,
  - Koeffizienten-Modifikation,
  - Löschen von redundanten Constraints,
  - „Constraint Upgrading“.

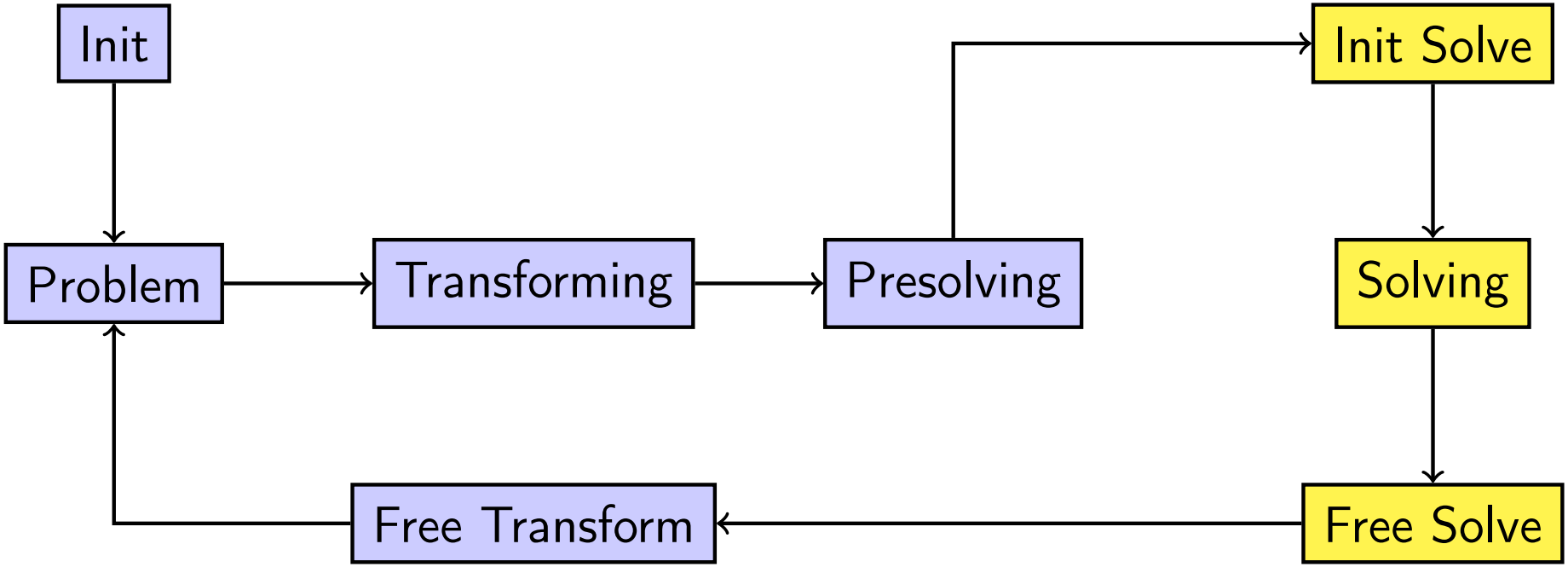
Probing:

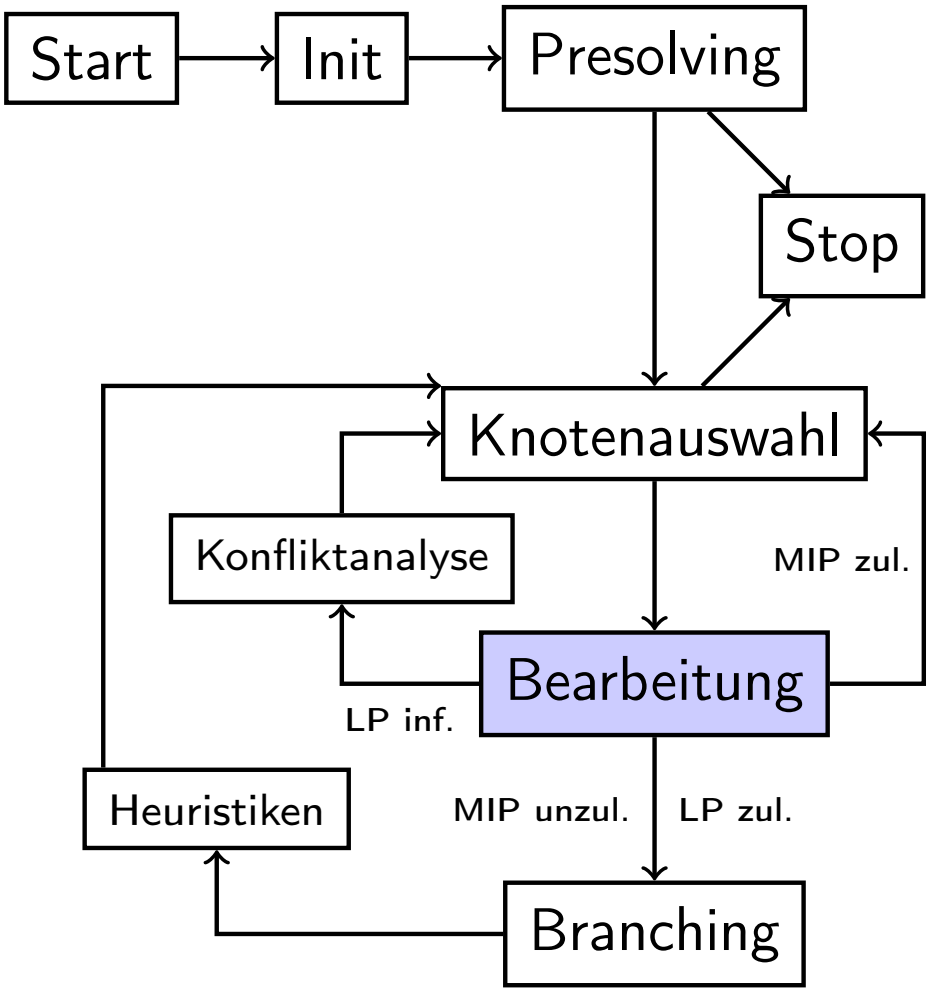
- Fixiere vorläufig Binärvariablen auf 0 bzw. 1.
- Führe Domain Propagierung der Constraints aus.
- Versuche globale Fixierungen bzw. Implikationen abzuleiten.

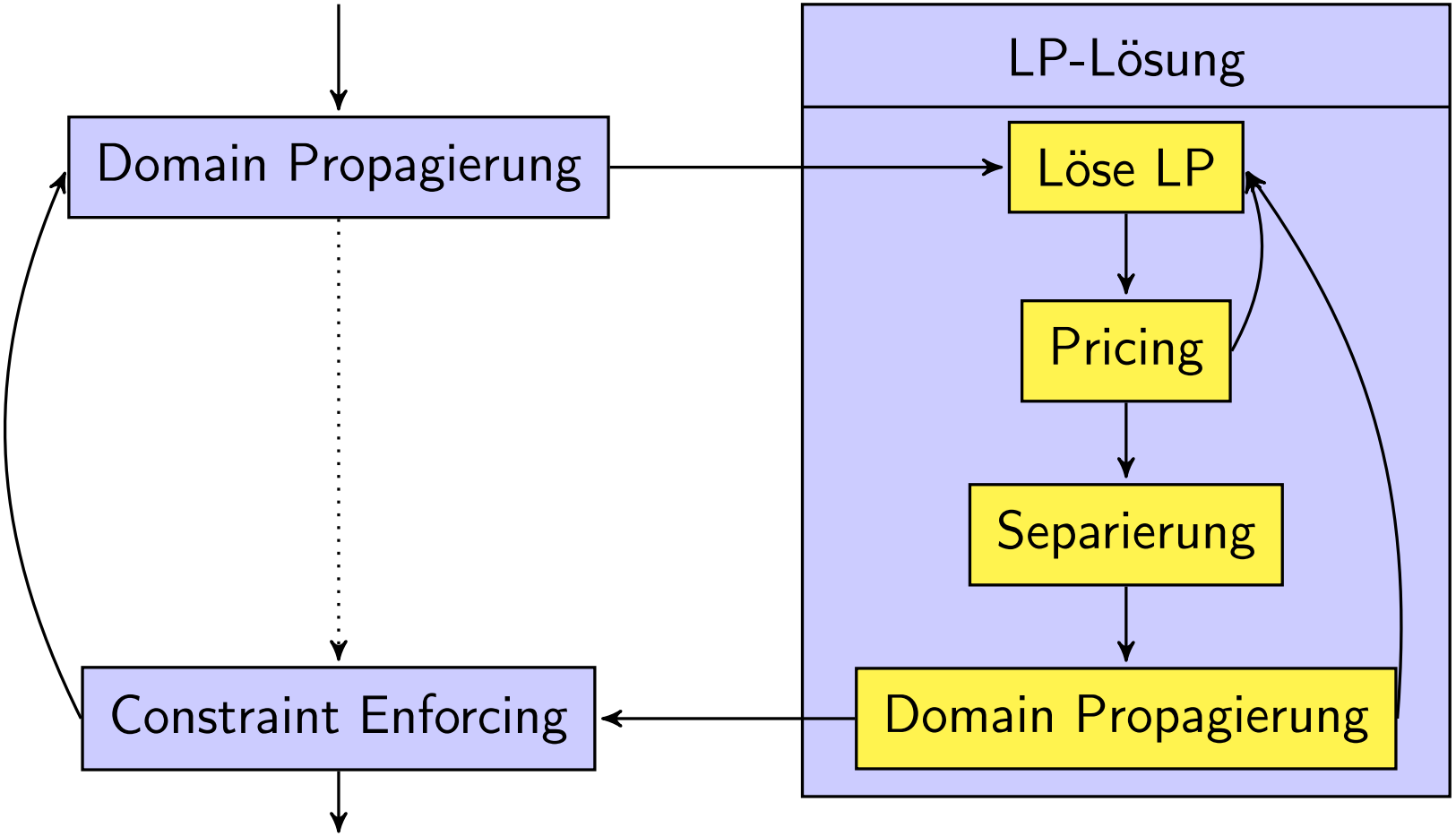
Beispiel:

$$\begin{aligned} x - z &\leq 0, \\ -y + z &\leq 0, \\ x + y &\leq 1 \end{aligned}$$

$$\Rightarrow x = 0$$


Init
Problem
Transforming
Presolving
Init Solve
Solving
Free Solve
Free Transform


Start
Init
Presolving
Stop
Knotenauswahl
Konfliktanalyse
MIP zul.
Bearbeitung
LP inf.
Heuristiken
MIP unzul.
LP zul.
Branching


LP-Lösung
Domain Propagierung
Löse LP
Pricing
Separierung
Domain Propagierung
Constraint Enforcing

TSP IP-Formulierung:

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

Keine-Subtour-Constraint: $nosub(x)$
x darf keine Subtour enthalten.

TSP CP-Formulierung

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

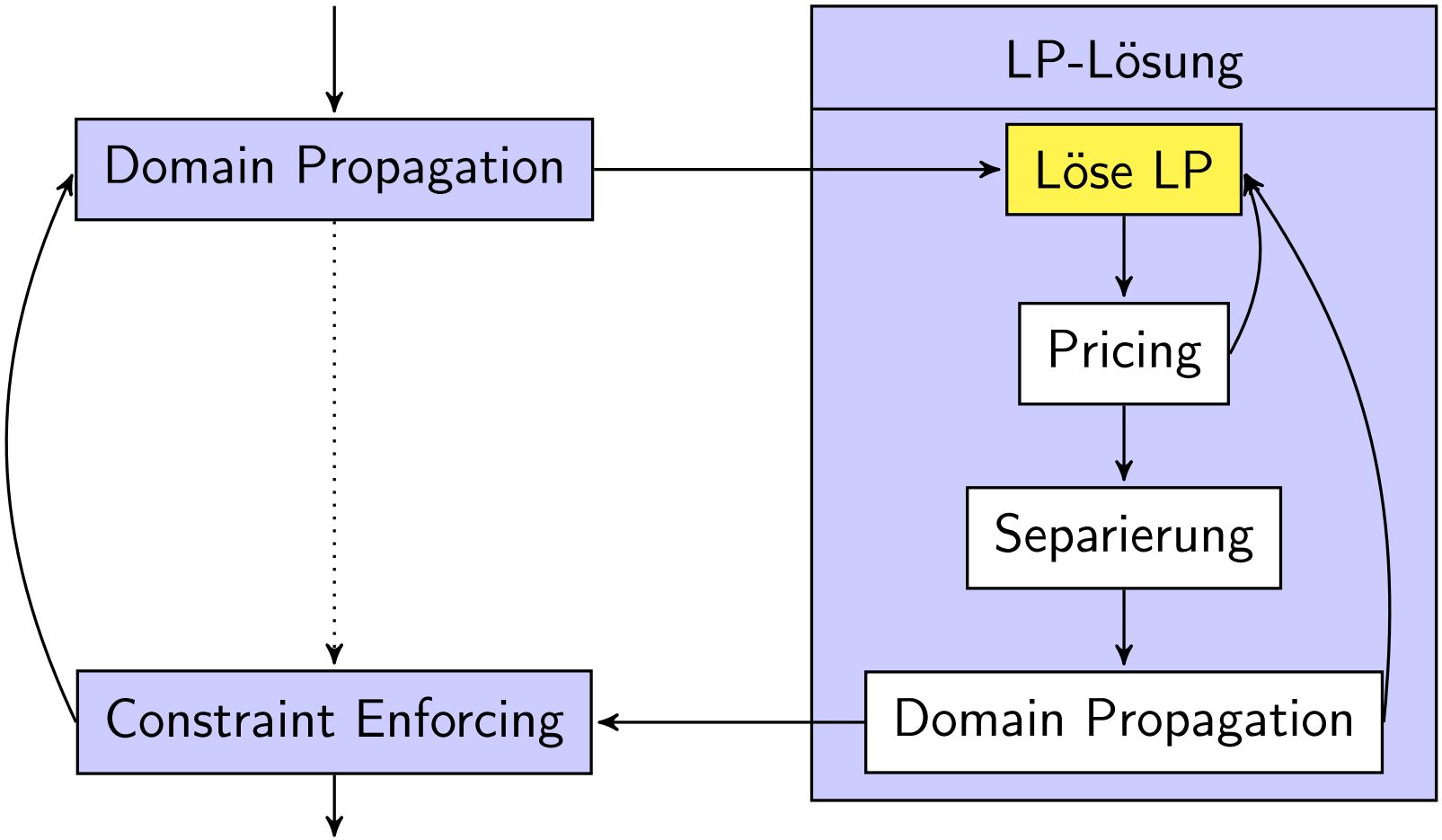
$$\sum_{e \in \delta(S)} x_e \geq 2 \qquad \forall S \subset V, S \neq \varnothing$$

Keine-Subtour-Constraint Handler:

- Separierung: separiere Ungleichungen für das LP
- Enforcing: separiere Ungleichungen
- Checking: überprüfe ob ganzzahlige Lösung (erzeugt von Heuristik) zulässig ist.
- Propagierung: –
- Sonstiges: (De)Initialisierung, “Locking”, Transforming, Ausgabe

lineare Constraints: $\alpha \leq a^T x \leq \beta$

- Hauptbaustein für das MIP-Lösen
- Ein MIP besteht aus linearen Constraints
- Kann zu anderen Constraints aufgewertet werden (z. B. Knapsack)
- Der Constraint Handler implementiert:
  - Presolving Methoden
  - Separiert die zugehörigen Ungleichungen
  - Propagiert die Variablen-Schranken.


LP-Lösung
Domain Propagation
Löse LP
Pricing
Separierung
Domain Propagation
Constraint Enforcing

LP-Lösung

- LP-Interface (`LPI`) zu verschiedenen LP-Lösern:
  SoPlex, CPLEX, XPress, CLP, ...
- Primaler oder dualer Simplex
- Barrier mit oder ohne Crossover
- LP-Stabilitätstest:
  Problemlösung durch Veränderung der Parameter:
  Skalierung, Toleranzen, Neuberechnung des LPs, andere Simplex-Methode.

Dokumentation

- Constraint Integer Programming
  Dissertation von Tobias Achterberg, 2007.
- SCIP – a framework to integrate Constraint and Mixed Integer Programming
  Tobias Achterberg, ZIB-Report 04-19, 2004.
- Primal Heuristics for Mixed Integer Programs
  Timo Berthold, Diplomarbeit, TU Berlin, 2006
- Implementation of Cutting Plane Separators for Mixed Integer Programs
  Kati Wolter, Diplomarbeit, TU Berlin, 2006

- `http://scip.zib.de`
  Doxygen Dokumentation, HowTos, FAQ

Algorithm Engineering Aspekte von SCIP

Zwei Algorithm Engineering Aspekte:

- MIP-Lösertechnologie
- Software-Entwicklung

MIP-Lösertechnologie

- Dramatische Fortschritte in der praktischen Lösbarkeit von MIPs: Seit den 1980ern gab es eine Beschleunigung um
  - mehr als einen Faktor 1000 durch Algorithmische Entwicklungen,
  - mehr als einen Faktor 1000 durch Hardwarebeschleunigung.
- Hauptkomponenten:
  - LP-Lösung
  - Schnittebenen
  - Preprocessing
- Weite Praxisverbreitung
  Simplex-Algorithmus einer der Algorithmen mit dem höchsten Rechenzeitverbrauch.
- Aber: MIP-Lösen ist immer noch NP-schwer!

Zusammenspiel von Mathematik/Informatik, Implementierungen und Praxisanforderungen

Beispiel: Gomory-Cuts

- Die Geschichte des Integer Programming startet 1958 mit der Entwicklung der Schnittebenen von Gomory
  (1960: Gomory Mixed Integer Cuts).
- Wurden in praktischen Berechnungen verwendet:
  1960: TSP-Berechnungen
- Dann verworfen wegen numerischer Schwierigkeiten.
- „Wiederentdeckt“ 1998
- Seitdem: Hauptbaustein von modernen MIP-Lösern.

Zusammenspiel

- Viele Verbesserungen verwenden Heuristiken oder im Grunde einfache Ideen.
- Einige Ideen haben weitreichende Folgen, z. B. mathematische ($\rightarrow$ polyedrische Kombinatorik).
- Viele Verbesserungen sind inspiriert durch Forschungsergebnisse.
- Praxisbeispiele machen Implementierungen stabil.
- Praxisbeispiele erfordern die Entwicklung neuer Methoden.

Das ist ist Algorithm Engineering!

Software-Entwicklung

- Reifegrad der MIP-Löser erfordert ein Minimum an Software-Engineering. Banale Beispiele:
  - Test-Bibliothek
  - Dokumentation/Asserts
  - Sorgfältiges Arbeiten
- Besondere Schwierigkeiten bei akademischen Codes: starke Veränderung der beteiligten Personengruppe
- Lizenzen: Freie Software vs. Kommerziell
  Beispiele: COIN-OR (Clp, CBC), SCIP/SoPlex, CPLEX

1. Einführung
2. Branch-And-Cut und Constraint Programming
3. SCIP
   - SCIP Plugins
   - Presolving
   - Lösen
   - Dokumentation
4. Algorithm Engineering

Programmhinweis

Workshop:

Combinatorial Optimization at Work

Martin Grötschel

21.9. bis 9.10.2009 an der TU Berlin (auf Englisch)

Inhalt:

- Zimpl, Soplex, SCIP
- viele Anwendungen (VLSI Design, Verkehr, Logistik, Telekommunikation, ...),
- Vorlesungen von Bob Bixby,
- Programmieren in SCIP

SCIP

. . . und Algorithm Engineering

Marc Pfetsch

Institut für Mathematische Optimierung
TU Braunschweig