Cloud Branching

Timo Berthold

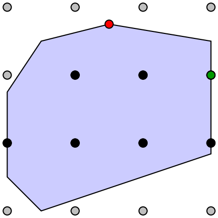
Zuse Institute Berlin

joint work with Domenico Salvagnin (Università degli Studi di Padova)

DFG Research Center MATHEON
Mathematics for key technologies

21/May/13, CPAIOR 2013, IBM T. J. Watson Research Center

$$\begin{aligned} \min \quad & c^T x \\ s.t. \quad & Ax \leq b \\ & x \in \mathbb{Z}^I_{\geq 0} \times \mathbb{R}^C_{\geq 0} \end{aligned}$$



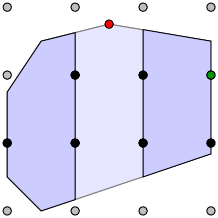
Mixed Integer Program:

- linear objective & constraints
- integer variables
- continuous variables

Branching for MIP:

- based on LP relaxation
- fractional variables
- tries to improve dual bound

$$\min \quad c^T x$$
$$s.t. \quad Ax \leq b$$
$$x \in \mathbb{Z}^I_{\geq 0} \times \mathbb{R}^C_{\geq 0}$$



Mixed Integer Program:

- linear objective & constraints
- integer variables
- continuous variables

Branching for MIP:

- based on LP relaxation
- fractional variables
- tries to improve dual bound

$$x_i \leq \lfloor x_i^* \rfloor \vee x_i \geq \lceil x_i^* \rceil \text{ for } i \in I \text{ and } x_i^* \notin \mathbb{Z}$$

Most infeasible branching

- often referred to as a simple, standard rule
- computationally as bad as random branching!

Strong branching [ApplegateEtAl1995]

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

Pseudocost branching [BenichouEtAl1971]

- try to estimate LP values, based on history information
- effective, cheap, but weak in the beginning
- can be combined with strong branching

Most infeasible branching

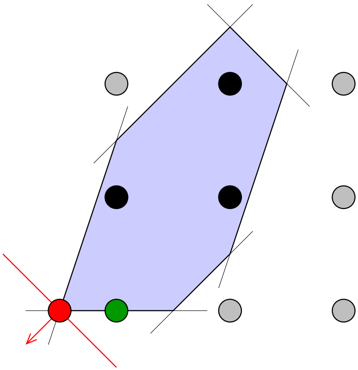
- often referred to as a simple, standard rule
- computationally as bad as random branching!

Strong branching [ApplegateEtAl1995]

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

Pseudocost branching [BenichouEtAl1971]

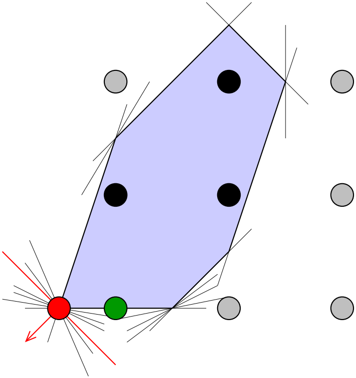
- try to estimate LP values, based on history information
- effective, cheap, but weak in the beginning
- can be combined with strong branching



naïvely:

- 1 optimal solution, determined by n constraints

Degeneracy

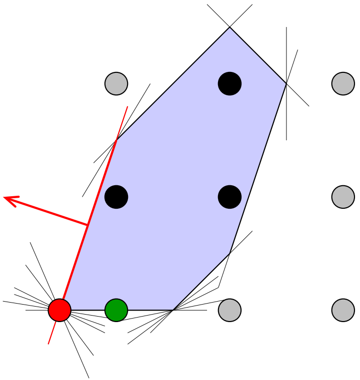


naïvely:

▷ 1 optimal solution, determined by n constraints

at a second thought:

▷ 1 optimal solution, $k > n$ tight constraints



naïvely:

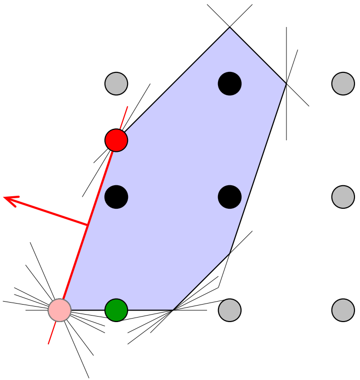
- 1 optimal solution, determined by n constraints

at a second thought:

- 1 optimal solution, $k > n$ tight constraints

but really:

- an optimal polyhedron



naïvely:

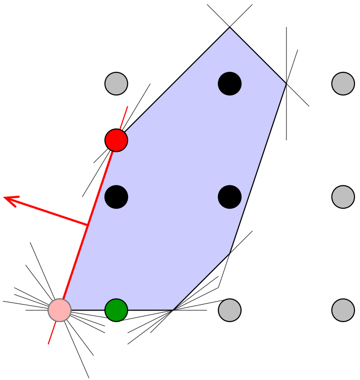
- 1 optimal solution, determined by n constraints

at a second thought:

- 1 optimal solution, $k > n$ tight constraints

but really:

- an optimal polyhedron



naïvely:

- 1 optimal solution, determined by n constraints

at a second thought:

- 1 optimal solution, $k > n$ tight constraints

but really:

- an optimal polyhedron

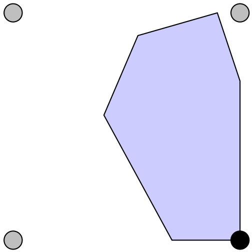
Goal of this talk:
branch on a set (a cloud) of solutions

Goal of this talk:
branch on a cloud of solutions

1. How do we get extra optimal solutions?
2. Why should that be a good idea anyway?

1. How do we get extra optimal solutions?

- restrict LP to optimal face
- min/max each variable (OBBT) or
- feasibility pump objective (pump&reduce [Achterberg2010])



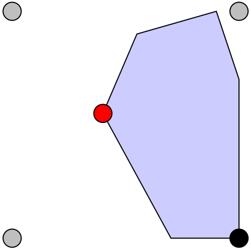
1. How do we get extra optimal solutions?

- restrict LP to optimal face
- min/max each variable (OBBT) or
- feasibility pump objective (pump&reduce [Achterberg2010])

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

$x_1 = 0.4$

$x_2 = 0.55$



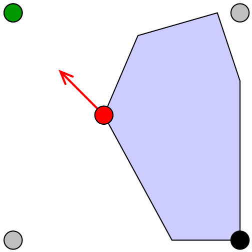
1. How do we get extra optimal solutions?

- restrict LP to optimal face
- min/max each variable (OBBT) or
- feasibility pump objective (pump&reduce [Achterberg2010])

$$\Delta(x, \tilde{x}) = \sum |x_j - \tilde{x}_j| \; = -x_1 + x_2$$

$x_1 = 0.4$

$x_2 = 0.55$



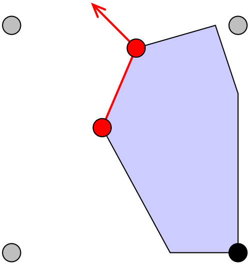
1. How do we get extra optimal solutions?

- restrict LP to optimal face
- min/max each variable (OBBT) or
- feasibility pump objective (pump&reduce [Achterberg2010])

$$\Delta(x, \tilde{x}) = \sum |x_j - \tilde{x}_j| \;= -x_1 + x_2$$

$x_1 \in [0.4, 0.55]$

$x_2 \in [0.55, 0.9]$



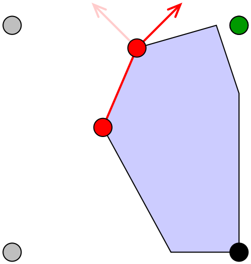
1. How do we get extra optimal solutions?

- restrict LP to optimal face
- min/max each variable (OBBT) or
- feasibility pump objective (pump&reduce [Achterberg2010])

$$\Delta(x, \tilde{x}) = \sum |x_j - \tilde{x}_j| \;\; = +x_1 + x_2$$

$x_1 \in [0.4, 0.55]$

$x_2 \in [0.55, 0.9]$



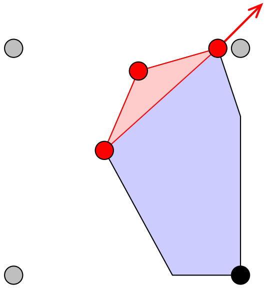
1. How do we get extra optimal solutions?

- restrict LP to optimal face
- min/max each variable (OBBT) or
- feasibility pump objective (pump&reduce [Achterberg2010])

$\Delta(x, \tilde{x}) = \sum |x_j - \tilde{x}_j| \; = +x_1 + x_2$

$x_1 \in [0.4, 0.9]$

$x_2 \in [0.55, 1.0]$



1. How do we get extra optimal solutions?

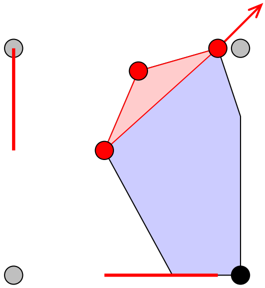
- restrict LP to optimal face
- min/max each variable (OBBT) or
- feasibility pump objective (pump&reduce [Achterberg2010])

$$\Delta(x, \tilde{x}) = \sum |x_j - \tilde{x}_j| \;\; = +x_1 + x_2$$

$x_1 \in [0.4, 0.9]$

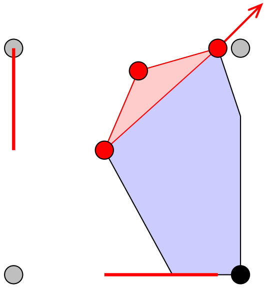
$x_2 \in [0.55, 1.0]$

$\rightsquigarrow$ intervals instead of values



1. How do we get extra optimal solutions?

- restrict LP to optimal face
- min/max each variable (OBBT) or
- feasibility pump objective (pump&reduce [Achterberg2010])



$$\Delta(x, \tilde{x}) = \sum |x_j - \tilde{x}_j| \;\; = +x_1 + x_2$$

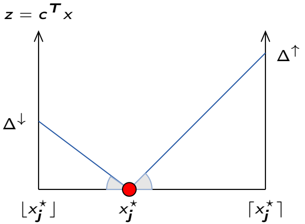
$x_1 \in [0.4, 0.9]$

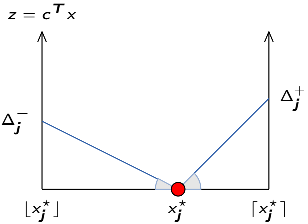
$x_2 \in [0.55, 1.0]$

$\rightsquigarrow$ intervals instead of values

stalling is cheap!

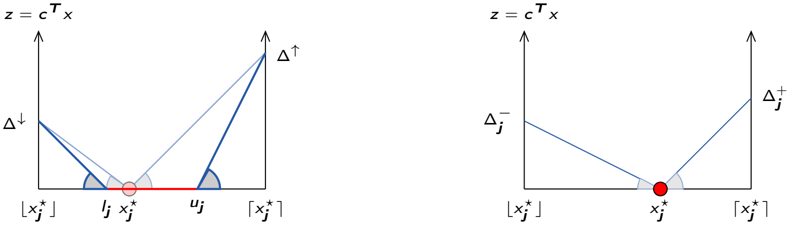
2. Why should that be a good idea anyway?





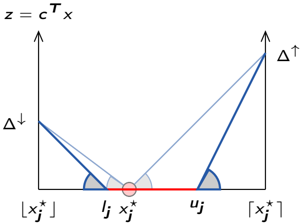
- pseudocost update
  - $\varsigma_j^+ = \frac{\Delta^\uparrow}{\lceil x_j^\star \rceil - x_j^\star}$ and $\varsigma_j^- = \frac{\Delta^\downarrow}{x_j^\star - \lfloor x_j^\star \rfloor}$
- pseudocost-based estimation
  - $\Delta_j^+ = \Psi_j^+(\lceil x_j^\star \rceil - x_j^\star)$ and $\Delta_j^- = \Psi_j^-(x_j^\star - \lfloor x_j^\star \rfloor)$

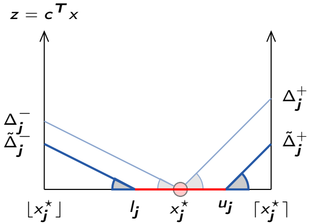
2. Why should that be a good idea anyway?



- pseudocost update
  - $\varsigma_j^+ = \frac{\Delta^{\uparrow}}{\lceil x_j^\star \rceil - x_j^\star}$ ... better: $\tilde{\varsigma}_j^+ = \frac{\Delta^{\uparrow}}{\lceil x_j^\star \rceil - u_j}$
- pseudocost-based estimation
  - $\Delta_j^+ = \Psi_j^+(\lceil x_j^\star \rceil - x_j^\star)$ and $\Delta_j^- = \Psi_j^-(x_j^\star - \lfloor x_j^\star \rfloor)$

2. Why should that be a good idea anyway?





- pseudocost update
  - $\varsigma_j^+ = \frac{\Delta^{\uparrow}}{\lceil x_j^\star \rceil - x_j^\star}$ ... better: $\tilde{\varsigma}_j^+ = \frac{\Delta^{\uparrow}}{\lceil x_j^\star \rceil - u_j}$
- pseudocost-based estimation
  - $\Delta_j^+ = \Psi_j^+(\lceil x_j^\star \rceil - x_j^\star)$ ... better: $\tilde{\Delta}_j^+ = \Psi_j^+(\lceil x_j^\star \rceil - u_j)$

Lemma

Let $x^\star$ be an optimal solution of the LP relaxation at a given branch-and-bound node and $\lfloor x_j^\star \rfloor \leq l_j \leq x_j^\star \leq u_j \leq \lceil x_j^\star \rceil$. Then

1. for fixed $\Delta^\uparrow$ and $\Delta^\downarrow$, it holds that $\tilde{\varsigma}_j^+ \geq \varsigma_j^+$ and $\tilde{\varsigma}_j^- \geq \varsigma_j^-$, respectively;
2. for fixed Ψ_j^+ and Ψ_j^-, it holds that $\tilde{\Delta}_j^+ \leq \Delta_j^+$ and $\tilde{\Delta}_j^- \leq \Delta_j^-$, respectively.

2. Why should that be a good idea anyway?

Full strong branching:

- solves 2·#frac. var's many LPs
- uses product of improvement values as branching score
  - improvement on both sides better than on one

Exploiting cloud points (strong branch)

2. Why should that be a good idea anyway?

Full strong branching:

- solves $2 \cdot \#$frac. var's many LPs
- uses product of improvement values as branching score
  - improvement on both sides better than on one

Benefit of cloud intervals:

- frac. var. gets integral in cloud point $\rightsquigarrow$ one LP spared
- cloud branching acts as a filter
- new frac. var.'s $\rightsquigarrow$ new candidates (one side known)

Cloud strong branching algorithm

Idea: Use 3-partition F_2, F_1, F_0 of branching candidates

- if strict improvements in both directions for F_2, disregard $F_1 \cup F_0$
- if strict improvement in one direction for $F_2 \cup F_1$, disregard F_0

Cloud strong branching algorithm

Idea: Use 3-partition F_2, F_1, F_0 of branching candidates

- ▷ if strict improvements in both directions for F_2, disregard $F_1 \cup F_0$
- ▷ if strict improvement in one direction for $F_2 \cup F_1$, disregard F_0
- ▷ alternatively: Only use F_2

Cloud strong branching algorithm

Idea: Use 3-partition F_2, F_1, F_0 of branching candidates

- if strict improvements in both directions for F_2, disregard $F_1 \cup F_0$
- if strict improvement in one direction for $F_2 \cup F_1$, disregard F_0
- alternatively: Only use F_2

- stop pump&reduce procedure when new cloud point does not imply new integral bound

Idea: Use 3-partition F_2, F_1, F_0 of branching candidates

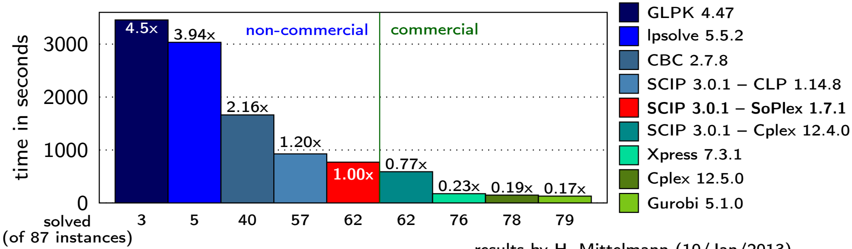
- if strict improvements in both directions for F_2, disregard $F_1 \cup F_0$
- if strict improvement in one direction for $F_2 \cup F_1$, disregard F_0
- alternatively: Only use F_2

- stop pump&reduce procedure when new cloud point does not imply new integral bound

Note: In our experiments, we do not use cloud points for anything else (heuristics, cuts)

SCIP: Solving Constraint Integer Programs

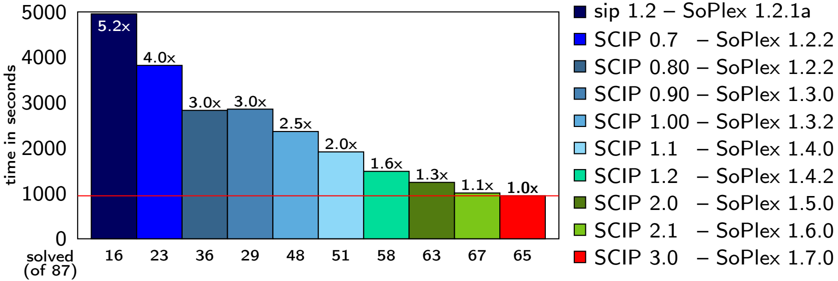
- standalone solver / branch-cut-and-price-framework
- modular structure via plugins
- free for academic use: `http://scip.zib.de`
- very fast non-commercial MIP and MINLP solver



results by H. Mittelmann (10/Jan/2013)

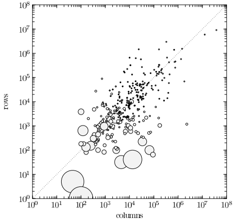
SCIP: Solving Constraint Integer Programs

- better support of MINLP
- new presolvers and propagators
- AMPL and MATLAB interface (beta)
- first releases of GCG and UG



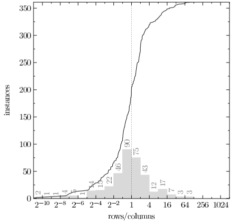
MMM

- MIPLIB 3.0, MIPLIB 2003, MIPLIB2010
- industry and academics
- 168 instances from diverse applications



Cor@l

- huge collection of 350 instances
- many combinatorial ones
- mainly collected from NEOS server



	cloud statistics			
Test set	%Succ	Pts	LPs	%Sav
MMM	12.2	2.19	74.34	21.7
COR@L	40.8	2.71	70.97	51.8

- applicable on some MMM, but many COR@L instances
- only few cloud points on average
- significants amount of LPs saved (if affected)

Test set	strong branch Nodes	strong branch Time (s)	cloud branch Nodes	cloud branch Time (s)
MMM	691	72.0	661	68.2
COR@L	593	157.3	569	118.3

- little less nodes
- 5.5% faster on MMM (few affected instances)
- 30 % faster on COR@L

Conclusion: Cloud branching...

- exploits knowledge of alternative relaxation optima
- can help to improve pseudocost predictions
- makes full strong branching up to 30% faster

Outlook

- pseudocost, reliability, hybrid branching
- cloud points from alternative relaxations (MINLP!)
- nonchimerical + cloud + propagation = ?

Cloud Branching

Timo Berthold

Zuse Institute Berlin

joint work with Domenico Salvagnin (Università degli Studi di Padova)

DFG Research Center MATHEON
Mathematics for key technologies